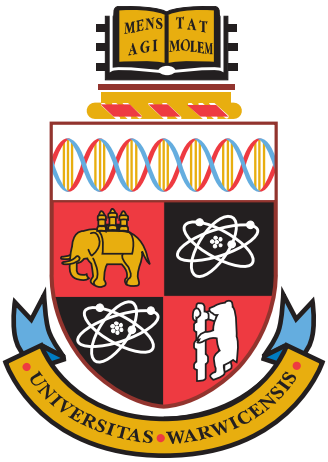


Graph-Based Segmentation and Scene Understanding for Context-Free Point Clouds

Sandro Spina
B.Sc.(Hons), M.Sc.

A thesis submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy in Engineering

School of Engineering
University of Warwick
2015

Contents

List of Figures

List of Tables

List of Publications

Papers

Spina, S., Debattista, K., Bugeja, K. & Chalmers, A. (2011). Point cloud segmentation for cultural heritage sites, *12th International conference on Virtual Reality, Archaeology and Cultural Heritage*, The Eurographics Association, pp. 41-48.

Spina, S., Debattista, K., Bugeja, K. & Chalmers, A. (2012). Fast scalable k-NN computation for very large point clouds, *Theory and Practice of Computer Graphics*, The Eurographics Association, pp. 85-92.

Spina, S., Debattista, K., Bugeja, K. & Chalmers, A. (2014). Scene semgmentation and understanding for context-free point clouds, *Pacific Conference on Computer Graphics and Applications - Short Papers*, The Eurographics Association, pp. 7-12.

Publications under preparation

PaRSe - A graph-based segmentation method for raw generic point clouds (Chapter 5, extended), for *Computer Graphics and Application*

Acknowledgements

I would first like to express my appreciation and gratitude to my advisors Kurt and Alan; without your guidance, motivation and continuous support this thesis would not have been possible. Many thanks for the opportunities given and for always encouraging my research.

Thanks also go to those in the Visualisation Group. Special thanks to Kurt, Vedad, Belma and Elmedin who have tremendously helped me during my many research visits by always welcoming me in their homes.

I would also like to thank my colleagues at the Department of Computer Science. In particular, Keith and Kevin, for voluntarily taking my teaching load during this final year, and thus allowing me to keep focused on this thesis.

Keith, your input has massively contributed to raising the quality of this work. My heartfelt thanks!

My most profound gratitude goes to all my family, especially my parents Rita and Frans, my brother Claudio, various uncles and aunties, for always being there and supporting me throughout.

Finally, and most importantly, I would like to thank my beloved wife and daughters. Stefania, without your patience and support this thesis would not have been possible. Eliana and Clara, you were a continuous source of energy; thanks for filling my sails when the ship was at a standstill.

Declaration

The work in this thesis is original and no portion of this work has been submitted in support of an application for another degree or qualification at this university or at another university or institution of learning.

Sandro Spina

Abstract

The acquisition of 3D point clouds representing the surface structure of real-world scenes has become common practice in many areas including architecture, cultural heritage and urban planning. Improvements in sample acquisition rates and precision are contributing to an increase in size and quality of point cloud data. The management of these large volumes of data is quickly becoming a challenge, leading to the design of algorithms intended to analyse and decrease the complexity of this data. Point cloud segmentation algorithms partition point clouds for better management, and scene understanding algorithms identify the components of a scene in the presence of considerable clutter and noise. In many cases, segmentation algorithms operate within the remit of a specific context, wherein their effectiveness is measured. Similarly, scene understanding algorithms depend on specific scene properties and fail to identify objects in a number of situations. This work addresses this lack of generality in current segmentation and scene understanding processes, and proposes methods for point clouds acquired using diverse scanning technologies in a wide spectrum of contexts. The approach to segmentation proposed by this work partitions a point cloud with minimal information, abstracting the data into a set of connected segment primitives to support efficient manipulation. A graph-based query mechanism is used to express further relations between segments and provide the building blocks for scene understanding. The presented method for scene understanding is agnostic of scene-specific context and supports both supervised and unsupervised approaches. In the former, a graph-based object descriptor is derived from a training process and used in object identification. The latter approach applies pattern matching to identify regular structures. A novel external memory algorithm based on a hybrid spatial subdivision technique is introduced to handle very large point clouds and accelerate the computation of the k-nearest neighbour function. Segmentation has been

successfully applied to extract segments representing geographic landmarks and architectural features from a variety of point clouds, whereas scene understanding has been successfully applied to indoor scenes on which other methods fail. The overall results demonstrate that the context-agnostic methods presented in this work can be successfully employed to manage the complexity of ever growing repositories.

Keywords: computer graphics, point cloud processing, segmentation, scene understanding, object tagging

CHAPTER 1

Introduction

Over the past decade, digital photography has been adopted by a large segment of the population with digital cameras becoming less expensive, more advanced and ubiquitous. This has led to the creation of massive repositories of digital content in the form of images and videos. To address this continuous increase in content, research has looked into ways of automatically organising this data by using information contained within these images. In particular, computer vision and image processing techniques have been designed to reason about the content in this data. For instance, face recognition algorithms have experienced rapid advances and are now employed on social networks and digital cameras to associate people to photos in which they are visible. The application of these algorithms adds semantic layers to digital content, which otherwise would simply consist of a set of coloured pixels when processed by a computer system.

More recently, another digital representation is growing in popularity, one that describes the geometric surface structure of real-world objects. Instead of using colour sensors to capture the appearance, 3D cameras or scanners use depth sensors to measure the distance to the objects in view and produce a set of points in space, a point cloud. Both images and point clouds are necessary to capture different aspects of a scene. In a similar fashion to digital cameras, continuous improvements in 3D scanning technologies resulting in improved precision and higher acquisition rates, have been contributing towards an increase in the creation and size of point cloud content. This growth is accentuated by the widespread popularity of commodity hardware which is primarily intended for the acquisition of small indoor environments. As a result, applications making use of point clouds have increased, with the acquisition of 3D point information becoming common practice in many areas including architecture, cultural her-

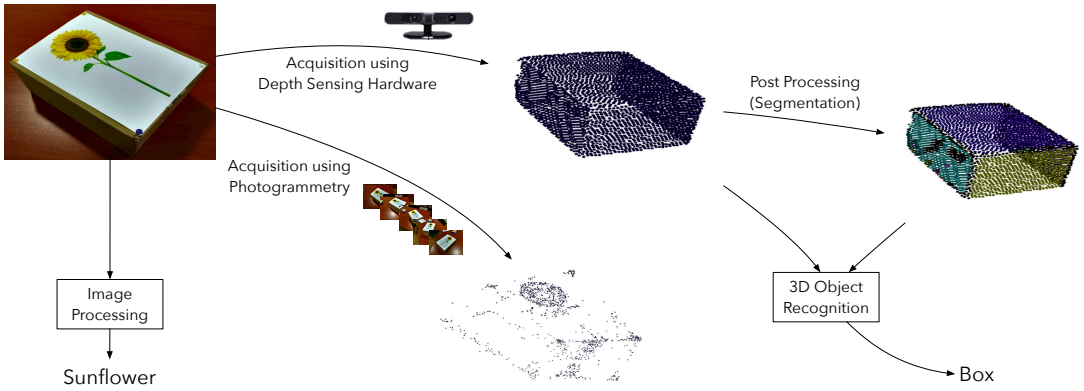


Figure 1.1: Acquisition of a scene into a digital representation, either as a 2D image or 3D point cloud. In the latter case, different methods can be used to scan the scene resulting in different results. The identification of objects in the scene depends on the digital representation and algorithms used. Segmentation plays an important part in the recognition process.

itage (CH), manufacturing and urban planning. A common underlying problem faced within these different domains is the increased complexity in handling these large data sets, typically ranging from a few thousand to several million points, which has resulted in the need for automated mechanisms to organise and hence facilitate the manipulation of point clouds.

While it may be a trivial task for a person to recognise objects and structures in an image or a point cloud, this is not a straightforward task for a computer system. This is particularly challenging in scenes comprised of an unknown number of different objects, where the complexity of the identification task is augmented with the added challenge of determining object boundaries, with objects which may only be partially visible to the sensor due to inter-object occlusions. Typically, using both appearance and shape information increases the potential for a correct interpretation of a scene. Figure 1.1 illustrates an example, where image processing techniques may correctly identify a sunflower and 3D object recognition techniques may determine the presence of a box. The combination of these results may contribute to a better interpretation of the scene, such that the computer system can now deduce that the scene contains a box with a sunflower picture on one of its sides. In many cases, however, information related to the appearance of a scene is not available with point clouds. For instance, in many point clouds used as case studies in this thesis the acquisition process is carried out by third parties, and only position information per point is made available.

Points from different scanning views can be combined in a common coordi-

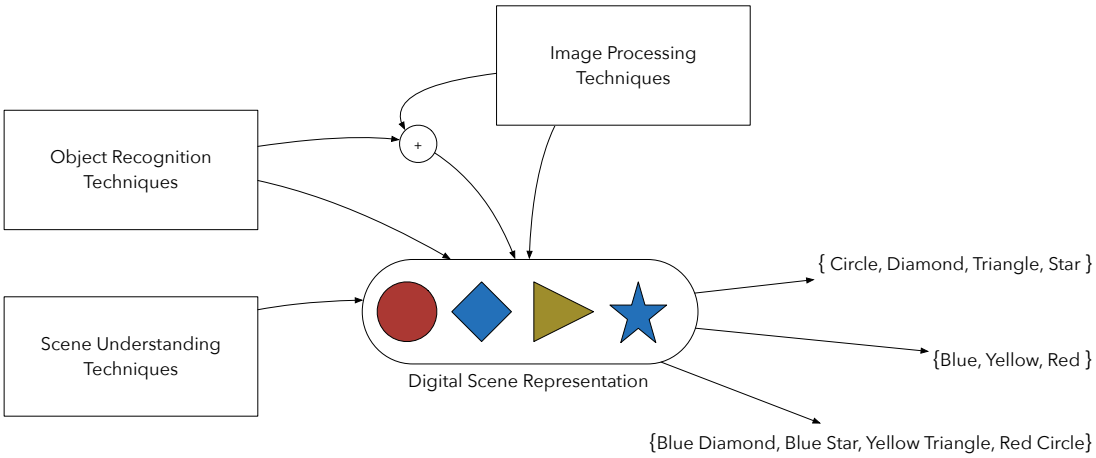


Figure 1.2: Given a digital representation of a scene, image processing, object recognition and scene understanding techniques have been used to identify the entities/objects contained in the scene. The output is an association between elements of the digital representation and a semantic interpretation, in this case for coloured shapes.

nate space to create a higher quality point cloud, with each depth sensor view contributing surface points which may not be visible from the other views. In particular, Iterative Closest Point (ICP) algorithms are used to combine individual scanner views into one consistent point cloud, and Simultaneous Localization and Mapping (SLAM) techniques are used to construct a point cloud of a scene by continuously keeping track of the scanner's location within the scene. Figure 1.1 illustrates the application of these techniques to produce a point cloud which closely matches the shape of the box in the image using depth sensing hardware. Following acquisition, a segmentation process can optionally be applied to partition the point cloud into groups of related points, where in this case, the groups, rendered using different colours, represent the four sampled sides of the box. This additional information may be used by a 3D object recognition algorithm to deduce that the point cloud represents a box. Analogous to a programming language compiler front-end, segmentation may be viewed as the tokenizer (or lexer) of the input stream, whereas 3D object recognition may be viewed as the parser, which groups together tokens representing specific constructs in the language. In the case of point clouds, these groups of points, referred to as segments, may be subsequently used to carry out a variety of tasks, for instance removing objects from a scene to improve visualisation, or measuring distances between the boundaries of a room.

Figure 1.2 illustrates the high-level approaches used for object identification

from point clouds, which include 3D object recognition and scene understanding techniques. In both cases, the result consists of a mapping between point subsets of the scene point cloud and semantic labels. Scene understanding algorithms have traditionally been employed to classify images of scenes into semantic categories; for instance, an input data set classified as an office space. The many different techniques can be divided into two main categories, namely scene-centred approaches and object-centred approaches. In the former, characteristics of the scene such as clutter and symmetry are used for classification, whereas in the latter, classification depends on the identification of objects such as plants, chairs and tables. These algorithms find application in numerous fields by contributing additional semantic information, particularly when the input is a set of images, but also when the input is a point cloud. Figure 1.3 illustrates a point cloud of a typical scene acquired using triangulation-based (§2.3.2) depth sensors. An autonomous navigation unit moving around such an environment would certainly benefit from the understanding of surrounding structures, for instance, if given the task of locating an object on a shelve or a table. In these scenarios, the majority of samples are initially taken from the surfaces of the larger objects in the scene, such as the table, chairs and shelving. Since fewer samples cover small objects that lie on the table or the floor, identification of these elements becomes more challenging. Proper identification of these smaller objects requires a separate acquisition step which only considers a small portion of the room such as the table top, or shelve. If an autonomous navigation system can properly interpret the room, then it can focus its scanners on the smaller section of the room to locate the required object. 3D object recognition techniques (§4.1) have predominantly been used to identify these small objects using a myriad of point-based object descriptors. On the other hand, scene understanding techniques (§4.2) have been employed to identify the more prominent components of a scene by making use of segment-based scene descriptors, which encode via a training process the objects making up a scene. This approach has been shown to be very sensitive to changes in the object's poses between trained and unseen scenes. For instance, toppled or inclined chairs similar to those in the office point cloud of Figure 1.3 are not correctly classified by any of the state of the art indoor scene understanding techniques. The chair on a table example highlights another shortcoming of these techniques, in that many work on the assumption that objects are always located at pre-established distances from a user-defined ground. The work presented in this thesis seeks to address these limitations.

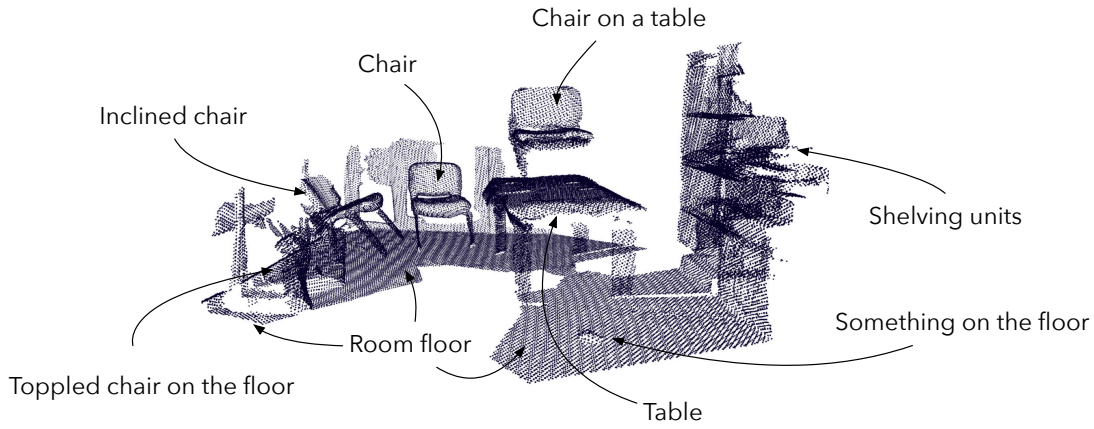


Figure 1.3: Scene understanding of indoor scenes from point clouds acquired using commodity triangulation-based hardware are typically noisy and cluttered. Moreover these may contain static objects in different poses (chairs) and varying structures (shelving). In order to cover the entire room, the point cloud resolution is not sufficient to identify small objects (object on the floor and shelving units).

1.1 Structure and Object Identification from Point Clouds

Figure 1.4 illustrates point clouds scanned using a variety of scanning technologies. These include long-range time-of-flight laser scanners to acquire the Mnajdra and Tarxien pre-historic sites over a number of hours (second row), and commodity hardware such as the triangulation-based Asus Xtion scanner, to acquire the office scene in under 1 minute (first row). Segmentation algorithms (Chapter 3) are necessary when working on very large point clouds, for tasks such as visualisation, editing and storage. In some cases, segmentation can be as straightforward as partitioning the points into equally sized regions, as shown in Figure 1.4 (bottom row). Other more complex tasks, such as object recognition or distance measurements, require the use of a more elaborate segmentation process, where the output segments correspond to some meaningful concept. For instance, in the case of object recognition, segments could represent tables and chairs, and in the case of distance measurements, segments could represent structures such as floors, walls and ceilings. General purpose segmentation adopts two principal approaches, namely using processes that fit primitive geometric shapes, such as spheres and cylinders, to the input point cloud, and region-growing processes which expand segments from seed points by following some surface property criteria such as curvature. Both approaches make a number of assumptions about the input data; in the first case, that the points can actually fit the set of primi-

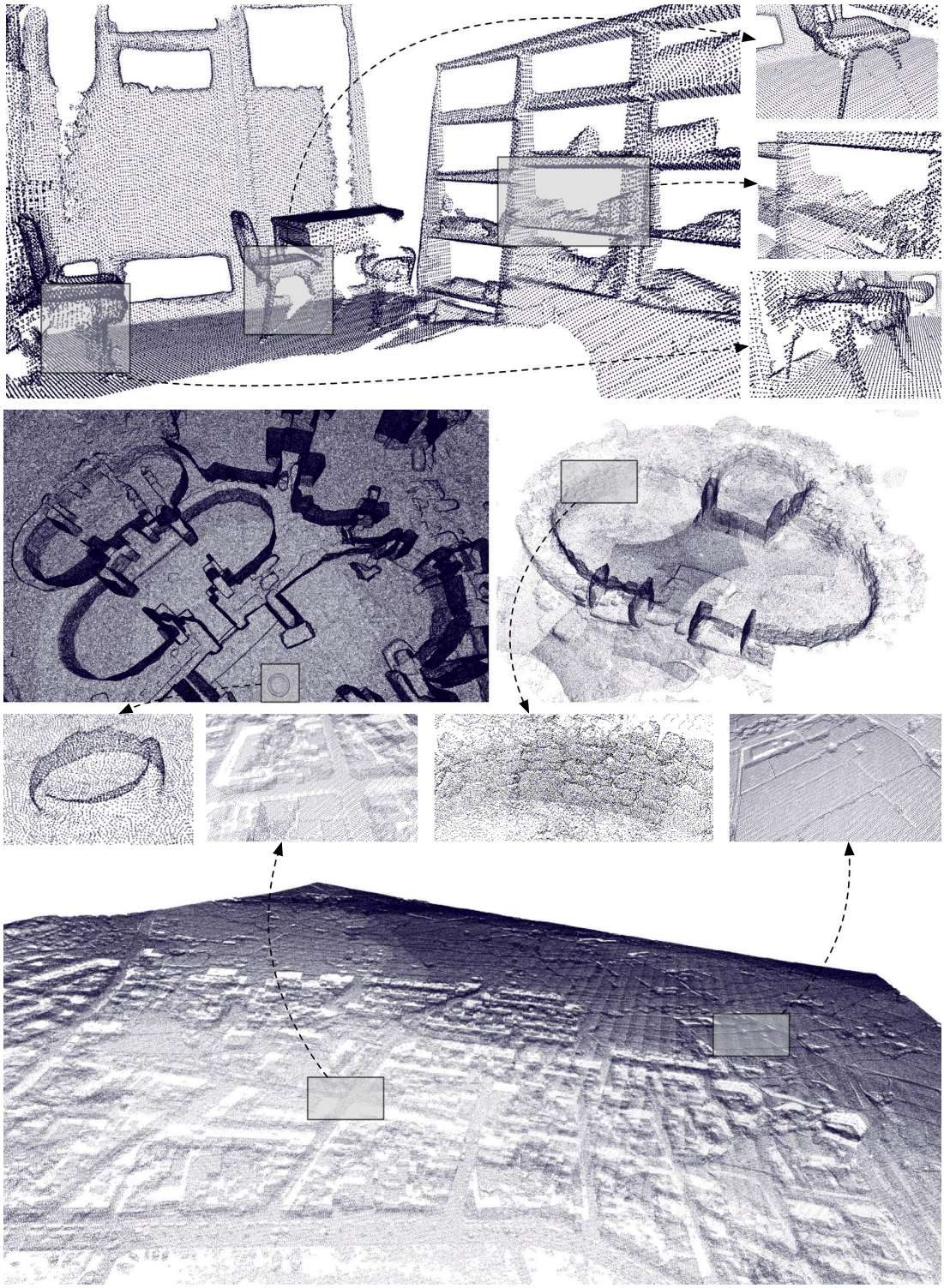


Figure 1.4: Point clouds acquired using different scanners and used in a variety of domains. Top row shows office scene scanned using a triangulation-based scanner (Asus Xtion); second row illustrates two point clouds scanned using time-of-flight laser scanners; third row illustrates a point cloud acquired using airborne LiDaR over the Maltese archipelago.

tive shapes used, and in the second, that the points are sampled from a relatively smooth surface with minimal sensor noise. Within these two approaches, processes have been tailored to suit specific scenarios, for instance segmentation of point clouds representing buildings, trees or industrial objects. The identification of objects and scene structures such as the floor, stairs or shelving, heavily depends on the segments produced. Whereas segmentation algorithms group points into related clusters, they do not provide a semantic interpretation for the segments. Instead, the grouping and labelling of these segments is the remit of scene understanding and object recognition algorithms. For instance, a number of segments corresponding to the steps of a flight of stairs, are grouped together and labelled appropriately as stairs. In addition to correct segmentation results, many indoor scene understanding methods also rely on scene-specific parameters; for example, the upward direction of the scene and distances between an object and the ground. This leads to scenarios where slightly changing the size, pose, or vertical position of an object (e.g. Figure 1.3) renders the method ineffective.

1.1.1 Applications

Many tasks carried out in a variety of fields can benefit from segmentation, 3D object recognition and scene understanding methods from point clouds.

Cultural Heritage

Many institutions are engaged in the acquisition of point clouds representing sites and objects of significant CH importance. Segmentation is necessary for documentation and preservation of a site or object, and enables realistic virtual reconstructions and dissemination to the general public (Yastikli, 2007; Rinaudo *et al.*, 2010).

Architecture

Semantically rich digital facility models are usually produced from computer aided design (CAD) models of a building. Given the variance between CAD models and what is actually built, indoor scene understanding techniques have recently emerged which use point clouds acquired using laser scanners to automatically synthesise building information models (BIM) (Tang *et al.*, 2010).

Planning

3D building data is an integral part of many large-scale urban models,

with data acquired using a variety of sensors and acquisition techniques. In particular, airborne LiDaR scanners producing accurate terrain 3D information greatly simplifies large-scale urban modelling (Hu *et al.*, 2003). Segmentation and object recognition techniques have been used to accelerate the post-processing effort by automatically partitioning the data into urban entities.

Robotics

Autonomous robot localisation and navigation greatly benefits from the availability of sensors capable of capturing depth information, for instance to prevent collisions and to locate specific objects (Biswas & Veloso, 2012). Indoor scene understanding techniques allow these autonomous robots the possibility of reasoning about their surroundings.

Manufacturing

Advances in 3D printing technologies have brought about a paradigm shift in the manufacturing process of small objects. Scanning technologies are used in order to manufacture replicas of real-world objects, which can then be accordingly modified using appropriate segmentation algorithms and 3D printed (Lipson & Kurman, 2013).

1.2 Research Aim

The research aim of this thesis is the advancement of techniques intended to facilitate the reasoning about and management of point clouds. Both segmentation and scene understanding methods contribute towards this goal. Previous indoor scene understanding methods, using both supervised and unsupervised approaches, have shown merit in reasoning about indoor scenes, but have so far depended on scene-specific context in their interpretation process. This thesis looks into filling this gap by designing a context-free scene understanding framework which is able to, without using scene-specific parameters, provide a valid interpretation for scenes such as the office in Figure 1.3. Segmentation is a critical component of this pipeline. Therefore, this work also looks at the development of a general purpose segmentation algorithm, which is able to reliably partition low quality point clouds acquired using commodity depth sensors for indoor scene understanding purposes, but which is also suited to partition point clouds acquired using a variety of other scanners for application in different fields (Figure

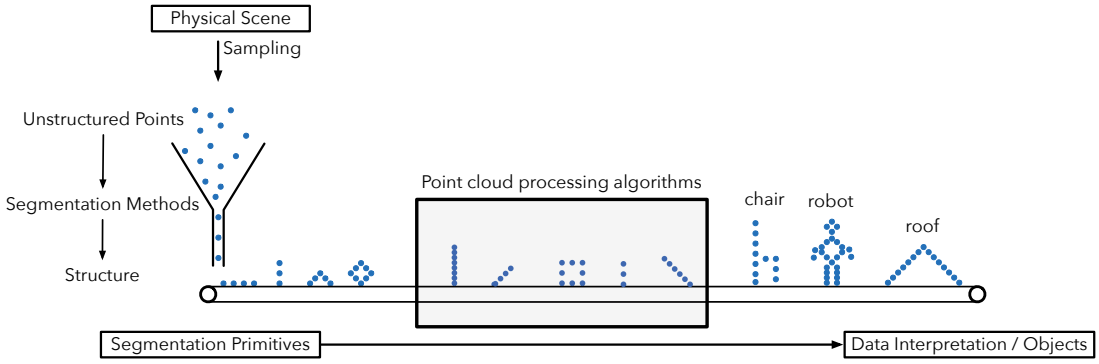


Figure 1.5: Rather than producing multiple point cloud processing pipelines where segmentation and task related processing methods are purposely designed to fit specific tasks, a context-free point cloud processing pipeline tries to minimise the set up required to address tasks from a variety of domains.

1.5). For this purpose, this thesis looks into the design of a general-purpose point cloud segmentation process which combines the advantages of both shape fitting and region-growing approaches.

1.2.1 PaRSe - Graph-based point cloud segmentation

For most tasks, manipulation of a point cloud usually requires extensive expertise in the use of CAD and modelling software. In this work, automated mechanisms which alleviate some of these tasks are presented in the form of a graph-based point cloud segmentation process, referred to as PaRSe, which combines the benefits of region-growing and primitive fitting approaches. Rather than just partitioning the input into a list of segments, a structure graph is built during the segmentation process which describes connectivity information between segment primitives. A variety of point clouds are used to evaluate the generality of the approach in supporting different tasks.

Objectives

- to design a general purpose segmentation algorithm
- to demonstrate its applicability on a variety of inputs, in particular point clouds representing CH sites

1.2.2 Processing of very large point clouds

In some cases, the size of the point cloud acquired is so large that it does not entirely fit in main memory. The majority of post-processing algorithms, such as segmentation, work under the assumption that the data sets operated on can fit in main memory, while others take into account the size of the data sets and are thus designed to keep data on disk. For many post-processing algorithms, a considerable amount of time is spent searching for the k-nearest neighbour (k-NN) of each point. Optimal performance results are achieved when k-NN computation is carried out in-core, i.e. when both points and acceleration structure are stored in main memory. On the other hand out-of-core techniques take into account the size of the points but are much slower due to overheads related to disk I/O. A novel out-of-core algorithm is presented in this thesis, which maximizes processor utilization while keeping I/O overheads to a minimum.

Objectives

- to enable the execution of point cloud segmentation on devices with limited memory
- to design an out-of-core k-NN process with similar running times to an in-memory approach

1.2.3 CoFFrS - Context-free scene understanding framework

The method to scene understanding presented in this thesis adopts a supervised approach. However, rather than using a training set of scenes to synthesise a scene descriptor and thus limiting its applicability to very similar unseen scenes, individual descriptors representing generic objects in the scene are synthesised using PaRSe and the inclusion of additional shape-related information. Searches for specific segment patterns in the input point cloud are used to recognise structures such as room boundaries and shelving. A novel scene understanding framework is introduced, referred to as CoFFrS, which first identifies scene structures by searching for specific segment patterns and then associates the remaining segments to previously trained objects.

Objectives

- to design a scene understanding method that is not sensitive to changes in object pose and scene parameters
- to determine, using a qualitative approach, its effectiveness against scenes from previous literature and new ones

1.3 Thesis Outline

This thesis is organised as follows:

Chapter 2: Preliminaries provides a comprehensive overview of concepts, definitions and notation used throughout the rest of the thesis.

Chapter 3: Segmentation of Point Clouds provides a detailed literature review of the various segmentation methods used on point clouds.

Chapter 4: Object Recognition and Indoor Scene Understanding provides a detailed literature review on the methods used for 3D object recognition and indoor scene understanding from point clouds.

Chapter 5: Point Cloud Structure Graphs presents a general purpose graph-based segmentation algorithm and outlines its utility in a variety of tasks.

Chapter 6: Fast Scalable k-NN Searches for Very Large Point Clouds presents a novel out-of-core algorithm which enables devices with limited memory to carry out point cloud segmentation processes.

Chapter 7: Structure Graphs for Indoor Scene Understanding presents a novel pipeline for scene understanding tasks which does not rely on a specific scene context.

Chapter 8: Conclusions concludes the dissertation, discussing contributions and limitations of this work and presenting potential avenues for future work.

CHAPTER 2

Preliminaries

This chapter provides a comprehensive overview of concepts, definitions and notation used throughout the rest of the thesis. The notation used for sets and operations on them is first defined, followed by a description of point clouds in terms of this notation, together with a number of properties and operations generally associated with them. The acceleration structures used to speed-up computations on point clouds are then briefly outlined, followed by a description of a number of operations on points which take advantage of these acceleration structures.

2.1 Collections

An important concept in mathematics and computer science is that of a collection of objects with similar type. Within these collections, both order and repetition may or may not be important. In this section *sets* are defined, which are collections in which neither order nor repetition is important.

2.1.1 Set Comprehensions

The simplest way to define a set is by listing all elements in the collection. For example, the set of days in the week $D = \{Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday\}$. One important set is the one which contains no elements, the empty set: $\emptyset$. In general, it is useful to define sets in terms of properties that their elements are expected to satisfy, with properties expressed as *predicates*. Consider for example, *the set of nationalities of football players who scored at least once in a world cup competition*. The notation used

to define sets in terms of properties is called *set comprehension* and is used to construct sets without having to list all elements:

$$\{p : Person | p \in WorldCupFootballers \land score(p) \geq 1 \bullet nationalityOf(p)\}$$

The symbols | and • separate the three parts of the set comprehension. The first part, *declaration*, declares the variables used in the definition, the second part is a *predicate* and the third part, the *term*, gives an expression representing the objects inserted in the new set. If instead of the nationalities of the players, age of each player which satisfies the predicate needs to be constructed, then the term can be changed to *ageOf(p)*. Both predicate and term can sometimes be omitted. For example, the term in the previous comprehension can be dropped to return persons. Moreover, if there are no constraints in the predicate part, i.e. this always evaluates to *true*, the predicate part can be omitted.

More complex properties on sets can be described using predicate quantification. These include universal quantification ($\forall$) and existential quantification ($\exists$) which are used to state that all or at least one objects in a set satisfy a particular property. Set operators such as subset ($S \subseteq T$), equality ($S == T$), union ($S \cup T$), intersection ($S \cap T$), and difference ($S \setminus T$) provide a mechanism for comparing sets and for creating new ones using sets which are already defined. Union and intersection operators can be generalised in order for them to be applied on a number of sets rather than just two.

2.1.2 Set Partitions

Generalised union enables the introduction of the notion of a partition of a set. For instance, given the set of all players participating in the world cup, one possible set partition is the one which creates 32 sets, with each set representing a specific team. Team membership is said to partition the set of players since (i) all players must be in at least one team, and (ii) players may not be in more than one of the teams. Each partition is a subset of the original set and is defined as follows:

Definition Given an index set I and sets P_i for every $i \in I$, we say that the indexed sets partition a set S if (i) they include all elements of S, i. e. $\bigcup_{i \in I} P_i = S$; and (ii) any two different partitions P_i and P_j have no common elements: $\forall i, j : I \cdot$ if $i \neq j$ then $P_i \cap P_j = \emptyset$.

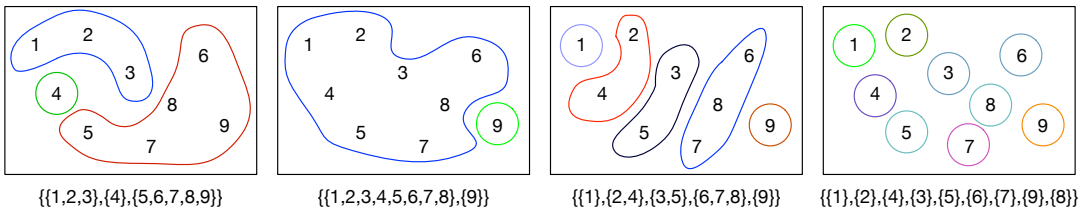


Figure 2.1: Four valid set partitions of the set {1,2,3,4,5,6,7,8,9}

Given the set {1,2,3,4,5,6,7,8,9}, Figure 2.1 shows a sample of valid set partitions. Set partitions provide a mechanism to cluster objects within a set, with the elements of the partition themselves sets, which can be modified using the set operators described above.

2.1.3 Power Set and Cartesian Product

The elements of a set partition of S are all subsets of S. The power set of S, written $\mathcal{P}(S)$, gives an enumeration of all these possible subsets. Using set comprehension, the power set is constructed as follows:

Definition Given a set S containing objects of type X, the power set of S, written as $\mathcal{P}(S)$, is defined to be the set of all subsets of S:

$$\mathcal{P}(S) \stackrel{\text{def}}{=} \{T | T \subseteq S\}$$

Given a set S containing n objects, $\mathcal{P}(S)$ contains 2^n objects. The Cartesian Product between sets provides a mechanism to combine objects from distinct sets. Using set comprehension, the Cartesian Product between two sets is defined as follows:

Definition Given a set S of type X, and another T of type Y, the Cartesian Product of S and T, written as $S \times T$, is defines to be the set of all pairs with the first object an element of S, and second object an element of T:

$$S \times T \stackrel{\text{def}}{=} \{x : X, y : Y | x \in S \land y \in T \bullet (x, y)\}$$

2.1.4 Relations

The notation and operations defined so far are used to construct and manipulate collections of related objects. *Relations* are then used to define correspondences

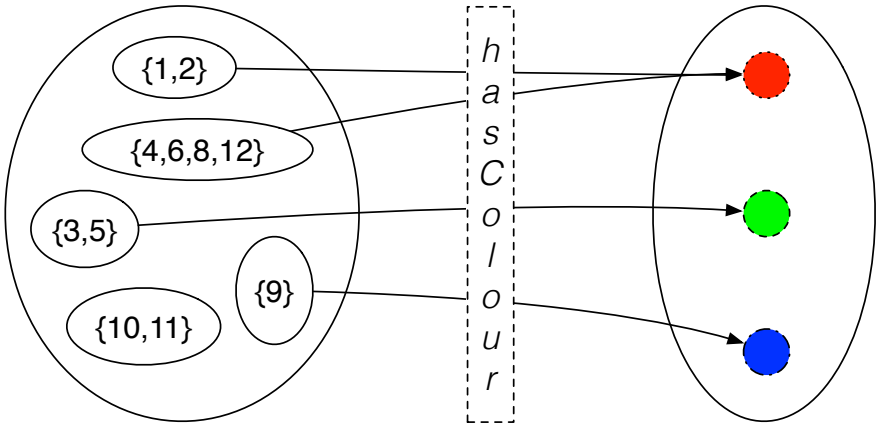


Figure 2.2: Arrows describe a relation between objects. In this case between objects in a set partition and colours in another set.

between these objects. Given sets S and T, a relation between these sets can be created to map in some specific way objects from one set to the other. Whereas the Cartesian Product denotes the upper bound on the number of mappings possible between objects in sets, a relation is used to define specific correspondences between these objects. Figure 2.2 illustrates two sets storing objects of different type. The set on the left shows a specific set partition, whereas the one on the right contains three objects, namely colours *red*, *green* and *blue*. A relation, for instance *hasColour* represented as arrows, maps objects from one set to the other. Note that not all objects need to form part of this mapping. For instance, in the case of object $\{10, 11\}$, the relation *hasColour* is $undefined$. Relations can naturally be expressed as sets of pairs. The use of sets to express relations enables the use of set comprehension to construct relations and the set operators previously defined can be used to combine relations of the same type.

2.2 Graphs

Figure 2.3 illustrates two graphs representing two different problems. In the first (left) nodes represent towns, whereas in the second (right) nodes represent process state. Despite representing different problems, they exhibit common features, in that both consist of a number of nodes (vertices or states) connected via arcs (edges or transitions) and both nodes and arcs carry some relevant information. The *transition relation* defines how nodes are connected in the graph. This relation, can either be directed as is the case with the CPU process life-cycle

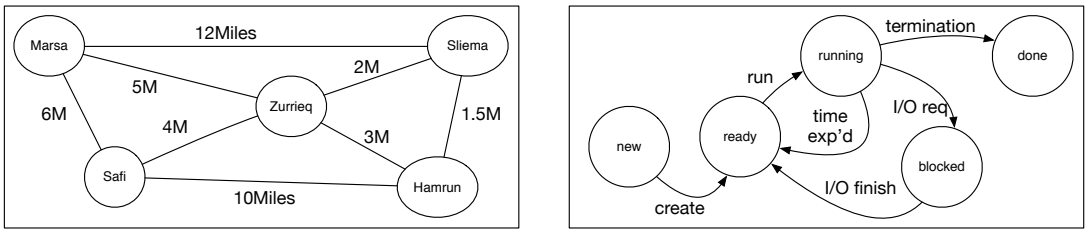


Figure 2.3: Graphs representing two different problems. In the first distances between towns is shown, whereas the second describes transitions between the different states of a running process.

which describes the different running process state transitions, or else undirected as in the distances between towns example. For the towns distance graph, this relation can be used to answer queries such as *give me three towns whose total distance between them is less than 6 miles*. The labelled arcs provide sufficient information such that the set $\{Zurrieq, Sliema, Hamrun\}$ can be computed. In the case of the running process graph, the relation can answer queries such as *is create, run and termination a valid sequence of events?* which is clearly valid and returns the set $\{new, ready, running, done\}$ enumerating the nodes visited whilst moving through the sequence. Arcs in the graph can be represented as triples (v, l, v'), where v and v' denote nodes in the graph connected via the labelled arc l. A graph is defined as follows:

Definition The graph $G = (V, L, E)$, consists of a set of nodes V, a set of labels L, and a set of labelled arcs between vertices $E \subseteq V \times L \times V$.

Labels can be used as predicates and thus enable a more generic method of attaching semantics to arcs and nodes. Consider for example augmenting the labels of the towns distances graph with elevation information in addition to distance. Arcs of the form (v, l, v') are extended to $(v, < l, m >, v')$, such that $(Marsa, 6M, Safi)$ becomes $(Marsa, < 6M, 200m >, Safi)$ by extending labels to vectors of information. A similar approach is taken with node labels, for instance instead of including elevation information in E, elevation values can be included within nodes in addition of town names. In general, graph labels for both nodes and arcs take the form of a sequence of key, value pairs. Keys are unique whereas values depend on what property is being modelled. In the case of town names, values can simply be a string of characters; in the case of elevation, a floating point number. Moreover, there are instances where the value is chosen

from a pre-established set of possibilities. For instance, if the exact population size is not important in terms of numbers, *size* values can be selected from the set $\{small, medium, large\}$. Rather than placing queries on a graph $G=\{V, L, E\}$, of the type *What is the distance between Zurrieq and Safi?*, it is sometimes useful to query the graph with *Are there two towns whose distance between them is less that 4 miles?*. The answer to this query is the set of pairs of nodes which satisfy the predicate, in this case $\{(Zurrieq, Sliema),(Zurrieq, Hamrun)\}$. Using set comprehension notation, and assuming a *distance* function is available, this query is expressed as follows:

$$\{(v, l, v') : E | distance(l) < 4 \bullet (nameOf(v), nameOf(v'))\}$$

2.2.1 Scene Graph

Graphs may be used to represent many different concepts. One which is closely related to computer graphics, is the *scene graph*, which in its basic form models the spatial relationships between objects in a virtual scene. Figure 2.4 illustrates a 2D scene of a room and a scene graph describing it. In its basic form, the scene graph is used to group together similar objects. For instance, furniture objects including tables and chairs are all located under the node $Furniture$ and these are further subdivided under $Chairs$ and $Tables$ nodes. The *on* relation is defined over the objects in the scene in order to describe objects which are directly placed on other objects. In this specific example, the relation is defined by the set $\{(PurpleChair, BlackTable), (TV, BlackTable)\}$. It clearly does not apply to all objects, however other relations (e.g. *proximity*, *contains*) can be added to the graph and set comprehensions can be used to construct these sets over objects in the scene.

If the objects in Figure 2.4 left are de-constructed, i.e. rather than viewing the scene as three chairs, one table, one TV, one lamp, two pictures, walls, floor and ceiling, these are viewed as a set of geometric line primitives making up the scene, then a very useful operation on this set is one which re-constructs the scene into its constituent objects. This operation takes a set of lines and produces a set partition whose elements represent individual objects. This specific problem is addressed by object recognition and scene understanding techniques, which make use of relations between these different parts to infer an interpretation of the scene, i.e. three chairs, one table, one TV, one lamp, two pictures, walls, floor and ceiling.

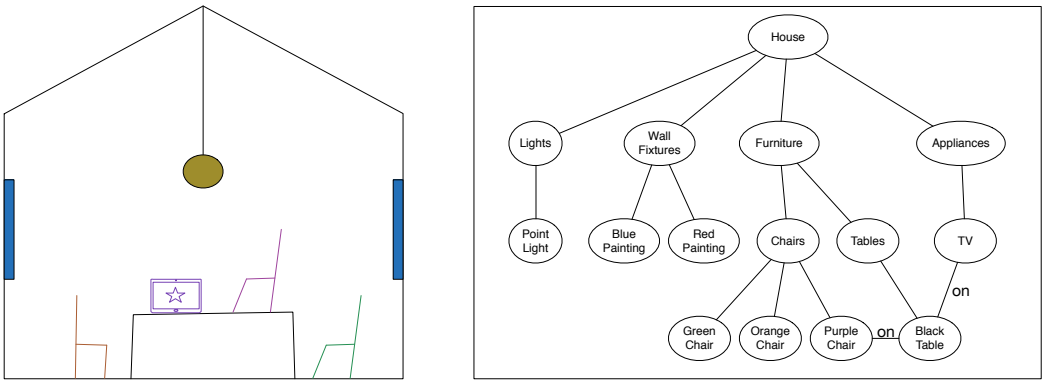


Figure 2.4: A simple 2D scene and a corresponding scene graph describing the elements in the scene. The *on* relation provides additional information on which objects are located on the table.

2.2.2 Transition Trees

A graph $G = (V, L, E)$ is said to be a *tree*, if it satisfies a number of constraints, namely, that every node may have no more than one predecessor (called the parent), except for one node which has zero parents and is labelled as the *root* node. Every node is reachable from the root node. A predecessor relation, explicitly defines an order over the nodes in the tree. The leftmost directed graph of Figure 2.5, illustrates a tree G_T with eight vertices and the predecessor relation:

$$E = \{(root, A),(A, B),(A, C),(B, D),(C, E),(D, F),(D, G)\}$$

A traversal of the tree, in either depth or breadth first order, produces a set of labels which describe the possible *connectivity paths* starting from the root node. For instance, a depth first traversal of the left-most tree in Figure 2.5 results in:

$$a \rightarrow b \rightarrow d \rightarrow f$$
$$a \rightarrow b \rightarrow d \rightarrow g$$
$$a \rightarrow c \rightarrow e$$

A *transition tree* can be created for each node in G_T, whereby each node is set as the root node. The middle and right most trees in Figure 2.5 illustrate the transition trees for nodes C and D respectively. The *depth* of a transition tree defines the maximum length of connectivity paths. For instance, if the depth for the C transition tree is set to 3, the resulting set of connectivity paths would be equal to $\{c \rightarrow a,\ c \rightarrow b \rightarrow d,\ e\}$. If the depth of the D transition tree is set to 2, the set of connectivity paths would be equal to $\{f,\ g,\ d \rightarrow b\}$.

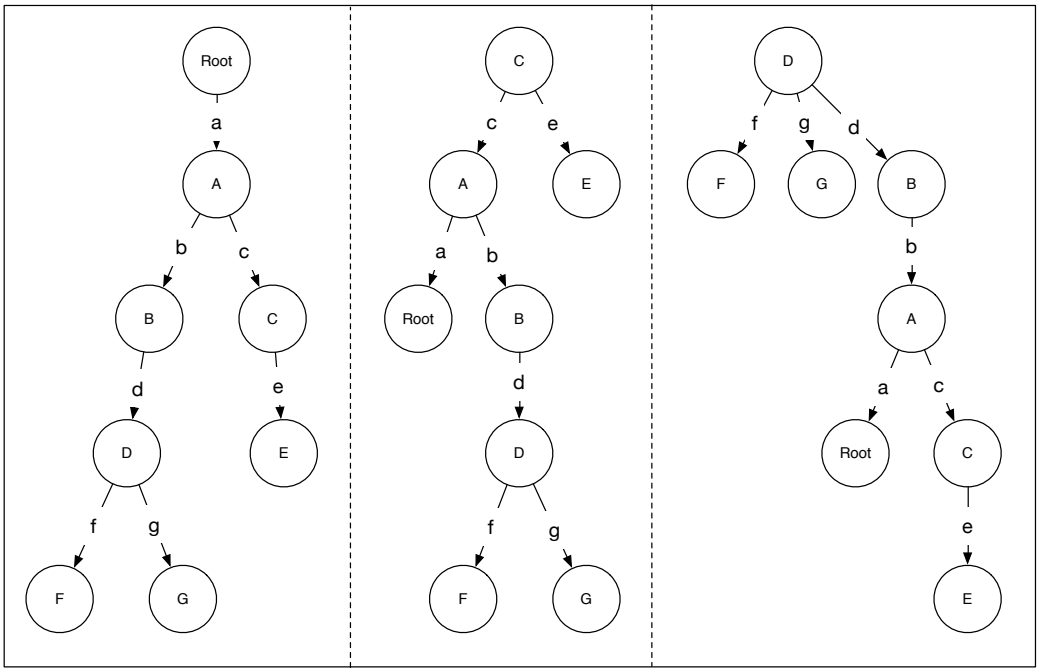


Figure 2.5: Three transition trees, with root nodes set from left to right to Root, C and D.

2.2.3 Graph Compatibility

Two graphs can be compared together in order to establish their *compatibility*. For instance, graphs encoding 3D objects can be directly compared to establish object similarity (Maple & Wang, 2004). Graphs G and H are equal only when their respective sets of vertices, labels and edges are equal as follows:

Definition A graph $G = (V_G, L_G, E_G)$ is equal to a graph $H = (V_H, L_H, E_H)$, written $G = H$, if an only if $V_G = V_H$, $L_G = L_H$ and $E_G = E_H$.

This definition of graph equality is usually relaxed in order to measure the distance between graphs using a variety of metrics. Figure 2.6 illustrates four 2D shapes with corresponding graphs modelling side connectivity. Nodes represent sides, whereas arcs define a relation specifying which of the sides are connected, with labels stating the smallest angle subtended between each pair of connected sides. In this case, a perfectly reasonable distance metric, compares the cardinality of the elements of the set partition of E when grouped by angles $x \in X$. The resulting set partitions are shown in Table 2.1. Using this distance metric, the first two shapes of Figure 2.6 would appear to be identical, whereas the square and octagon shapes would appear to be completely different. The house shape

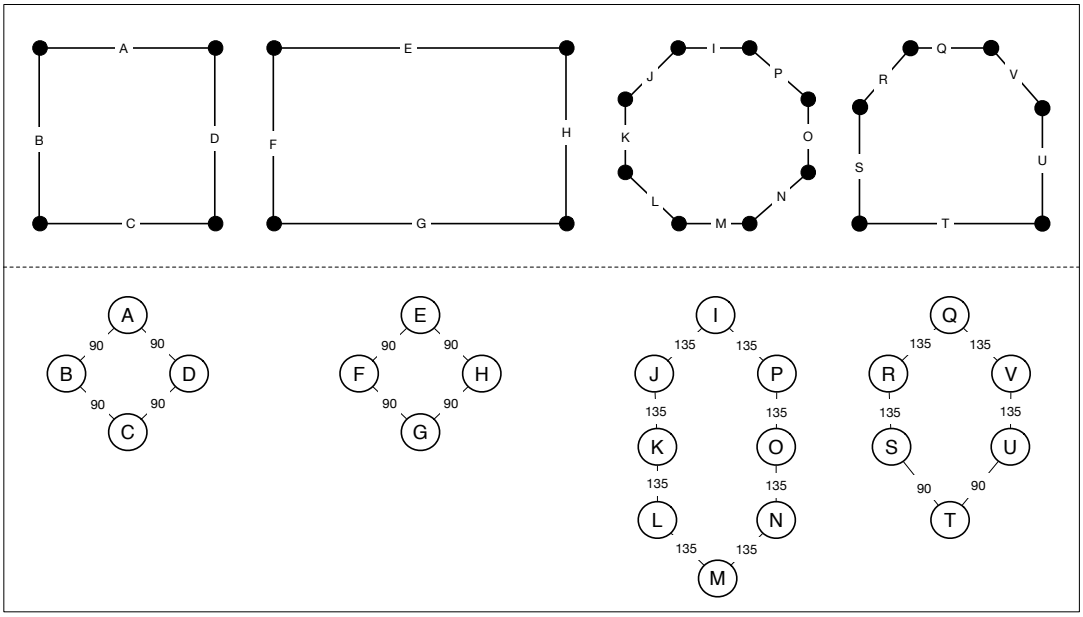


Figure 2.6: 2D shapes with their respective graphs, where nodes represent shape sides and edges are labeled with the smallest angle subtended between each pair of adjacent sides.

square	$\{\{(A, 90, B), (B, 90, C), (C, 90, D), (D, 90, A)\}\}$
rectangle	$\{\{(E, 90, F), (F, 90, G), (G, 90, H), (H, 90, E)\}\}$
octagon	$\{\{(I, 135, J), (J, 135, K), (K, 135, L), (L, 135, M), (M, 135, N), (N, 135, O), (O, 135, P), (P, 135, I)\}\}$
house	$\{\{(Q, 135, R), (R, 135, S), (U, 135, V), (V, 135, Q)\}, \{(S, 90, T), (T, 90, U)\}\}$

Table 2.1: Set partitions computed according to arc angle values for shapes in Figure 2.6

set partition contains a partition with four arcs at 135 degrees and another with two arcs at 90 degrees and therefore has elements (in this case angles between sides) in common with all the three other shapes. The shape distance metric used here is a heuristic, and therefore not guaranteed to give an optimal solution on all inputs (in this case 2D shapes) as illustrated in Figure 2.7. Using this same metric, the first and second shapes are identical, since the respective set partitions both have one partition (90 degrees) of cardinality 4. In order to further discriminate between these shapes, additional information has to be stored within the arcs. For instance, direction of rotation as either clockwise or anti-clockwise can be added and the set partitions originally computed to distinguish between relations of various angles can be further refined to now include orientation information.

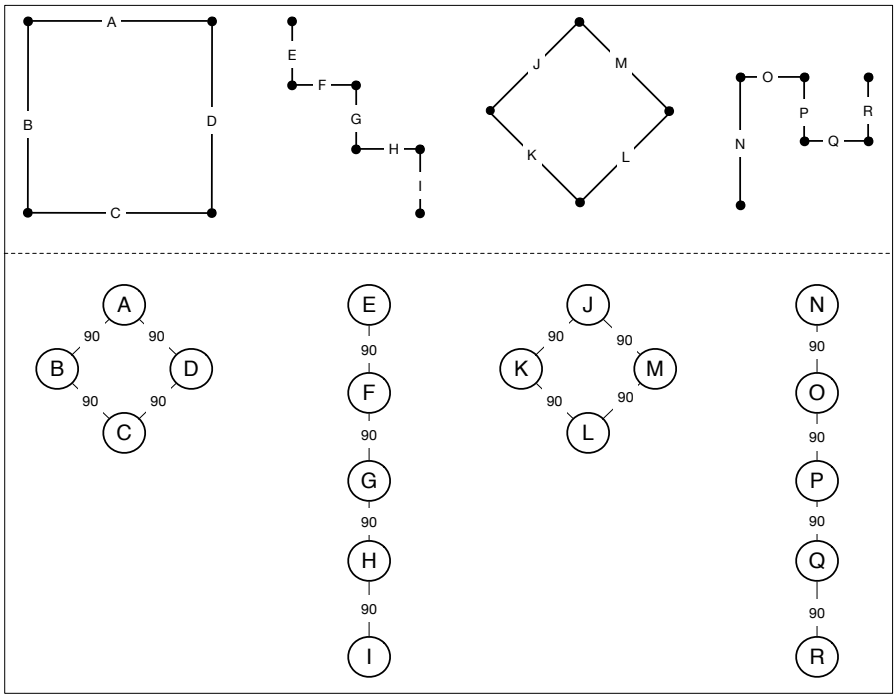


Figure 2.7: 2D shapes with their respective graphs, where nodes represent shape sides and edges are labelled with the smallest angle subtended between each pair of adjacent sides.

2.3 Point Clouds

A point cloud is a collection of geometric data points within a coordinate system. This collection can be viewed as a set, in that order is not important and no two elements are the same even if these points happen to have exactly the same properties. In the context of this work, a point cloud is used to describe a discrete point-based external surface representation. Minimally, it consists of a collection of geometric points storing per-point position information with no connectivity relation between points. A set embellished only of position information is referred to as a raw point cloud. If required, connectivity between points may be computed by applying some surface reconstruction algorithm to produce a continuous surface representation. Rapid advancements in acquisition methods and hardware have resulted in point clouds which can be very large and able to capture high frequency surface detail, with sizes ranging between 10^6 and 10^9 becoming commonplace.

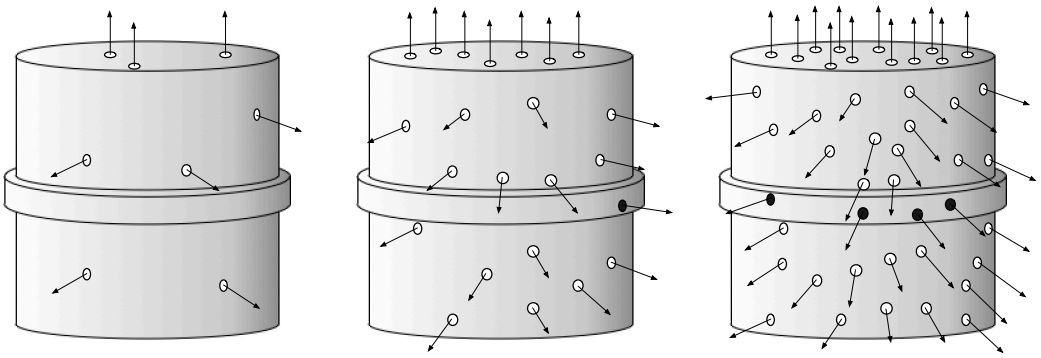


Figure 2.8: A cylinder with an etched band along its centre with (from left to right) increasing number of surface samples. Only the third set of samples provide some information about the etched band.

2.3.1 Point Sampling

At its very basic, a point sample consists of three real numbers defined within some 3D coordinate space which specify position. A direction vector representing the surface normal from where the point is sampled is also usually computed. Figure 2.8 illustrates a cylinder being represented with, from left to right, an increasing number of points with position and normal information. Additional properties are usually attached to the point depending on how the point is used. For instance, when used for visualisation purposes, the sample point tries to capture a very small area on the surface of the object and is therefore augmented with visualisation properties such as colour, alpha blending and pixel size properties.

Sampling refers to the process of acquiring a discrete set of points representative of a signal. In this context, the signal can either be a virtual or real scene. A virtual scene consists of a collection of geometric primitives (e.g. triangles, spheres), whereas a real scene can be anything physical around us. Whether virtual or real, a scene SC can be characterised as a set of parametrised 3D surface patches SP and defined as follows:

Definition A smooth parametrised surface patch SP in $\mathbb{R}^3$ is a function $x : U \subset \mathbb{R}^2 \Rightarrow \mathbb{R}^3$. As (u,v) varies over U, $x(u,v)$s traces out a surface patch in $\mathbb{R}^3$.

The function x, may represent any function that defines a surface. Figure 2.9 visually illustrates the definition of a surface patch above. For very simple surfaces, for instance a plane (continuous and without boundaries), three samples are enough in order to establish the function describing the surface patch whereas

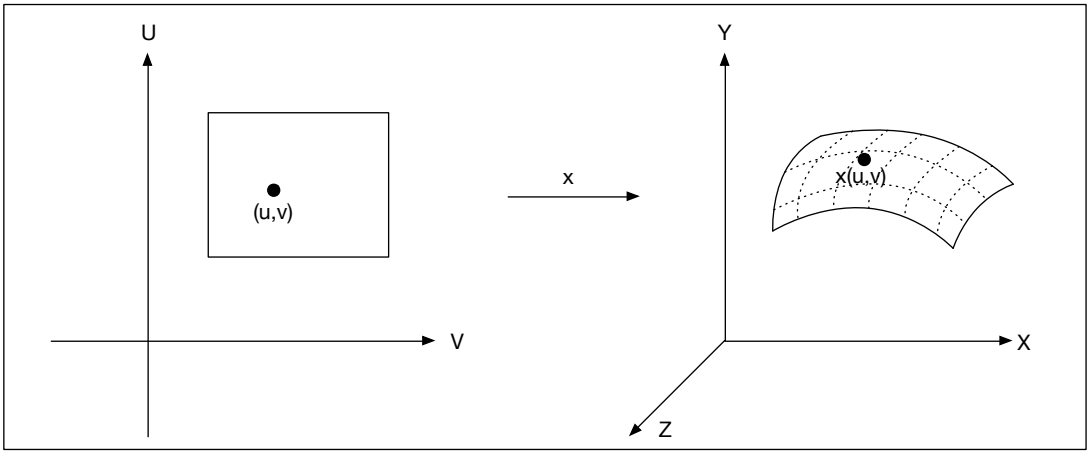


Figure 2.9: The function x traces a surface patch in $\mathbb{R}^3$.

for more complex free-form surfaces, many more samples are required to establish x. In the case of a generic scene, there may exist many sets of surface patches which can describe the scene, all of which might be valid. Given a particular set, any complex surface patch contained within may be further split into simpler surface patches. Establishing an optimal set of surface patches is in itself a very active research topic (Cohen-Steiner *et al.*, 2004) and is not discussed here.

When sampling, aliasing may occur depending on the sampling frequency used, where in order to correctly reconstruct an input signal, this needs to be sampled at least two times the original signal frequency (Oppenheim *et al.*, 1989). The higher the number of samples acquired from a scene, the closer the discrete point cloud representation is to the surface patch and in the general case the easier it is to reconstruct (e.g. for visualisation purposes). In the case of Figure 2.8, a geometric definition of a cylinder with an etched band around it is increasingly sampled from left to right. The additional samples on the right can be used to improve the reconstruction of the original signal. Whereas in all three cases there are enough points to parametrically fit the large cylinder and thus the same cylinder can be reconstructed, given no knowledge of the scene, the more samples acquired from a surface the higher the confidence that the object in the scene is a cylinder. Moreover, surface detail such as the band around the cylinder, is only captured in the third set of samples. In the second set of samples only one sample is acquired from the band and this can easily be disregarded as noise.

Scene sampling can be generalised as a ray casting process. Figure 2.10 illustrates how this process is carried out. An observer is positioned in the scene,

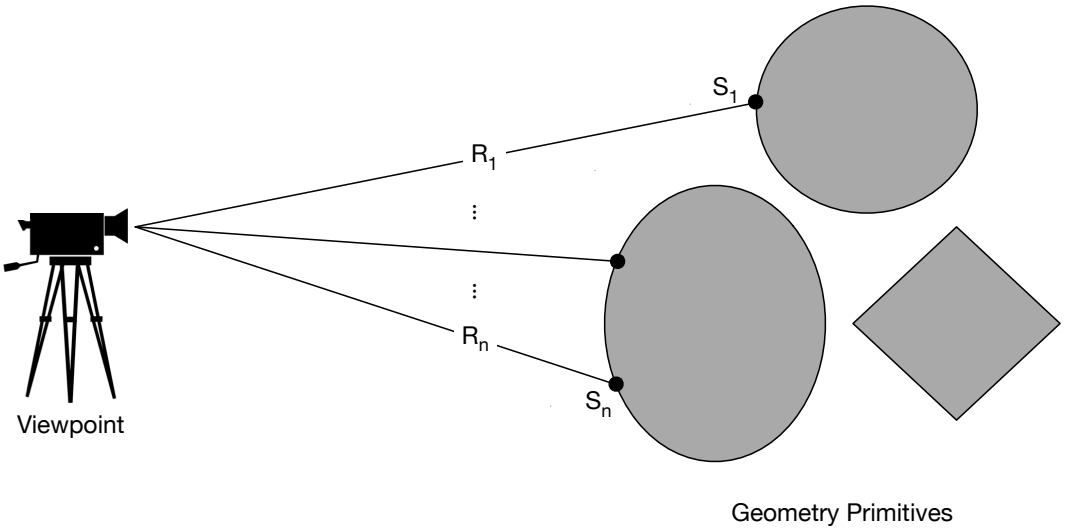


Figure 2.10: Sampling can be done using a ray casting process.

which casts rays towards the objects in the scene. Samples are taken by intersecting rays with visible object surfaces in the scene. In this example the diamond shape is not sampled since it is occluded by the oval shape. The set of samples produced from one observer viewpoint is referred to as a *scan* and is defined using set comprehension as follows:

$$\{r : Ray, sp : SP | Intersect(r, sp) \bullet (Sample(FirstIntersect(r, sp)))\}$$

This set comprehension describes the samples produced by one scan of a scene. The properties of a point are inserted in the set only if an intersection exists between one of the rays and the objects (represented as a set of surface patches, SP) in the scene. The set comprehension returns a set with the position of first ray surface intersections and depending on the scanner used, possibly other properties (e.g. colour and normal). The distribution of points depends on the set of rays r. By changing scanner viewpoint (position and/or direction) a new set of rays results in different intersection points within the scene producing additional samples. In general, in order to acquire samples from all surface patches in a scene, multiple scans are required, followed by a registration process which combines these scans into a single coherent point cloud. Given a set I of scans S whose points have been registered within the same coordinate space, a point cloud is defined as their union $\bigcup_{i \in I} S_i$.

2.3.2 Acquisition Methods

Many 3D acquisition methods generate point clouds as output. Alternatively, they may generate range images, analogous to a regular image, which store depth values along each of regularly spaced rays in space. Range images can easily be transformed to a point cloud, whereas the inverse is not always possible, especially in the case when the point cloud corresponds to more than one scan. Point clouds are used throughout this work, rather than range images, since these are appropriate for all types of scanners and can be used in all stages of the 3D acquisition pipeline. Bernardini & Rushmeier (2002) provide an in depth discussion of the 3D acquisition pipeline. All the algorithms presented in this work take point clouds as their input, hence this section briefly discusses the different classes of 3D scanners which produce them and a number of properties associated with them.

Point clouds acquired using a variety of 3D scanners have been used as case studies in this work. These scanners can be grouped into two main categories, namely triangulation-based scanners and time-of-flight scanners (Kolb *et al.*, 2009). In order to determine the 3D position of a point, triangulation-based scanners must observe a specific surface point from at least two separate viewpoints. Given this constraint is satisfied, the position is determined by computing corresponding pixels from the two (or more) calibrated viewpoints. This correspondence defines a pair of rays in space, with the intersection of these two rays determining the 3D position on the surface of the object as illustrated in Figure 2.11. Triangulation-based scanners are further subdivided by how the viewpoint is represented. In passive stereo systems (Gross & Pfister, 2011), the viewpoints contain cameras and no controlled light is introduced in the scene. In this case, determining correspondence between pixels on the two image planes equates to searching for pixels with similar features. In many cases this proves to be a difficult task, hence the introduction of active stereo systems which augment the setup with a spatially temporarily varying projected pattern designed to introduce features into the scene that make the correspondence problem easier to solve. Many variants have been developed using this setup including single-light-stripe systems (Petrov *et al.*, 1998) and structured-light systems (Ribo & Brandner, 2005). For indoor scene acquisition, different scanners have been used in this thesis, namely Asus Xtion (Asus, 2015), Microsoft Kinect v1 (Zhang, 2012) and Structure Sensor (Occipital, 2015), which are all based on structured

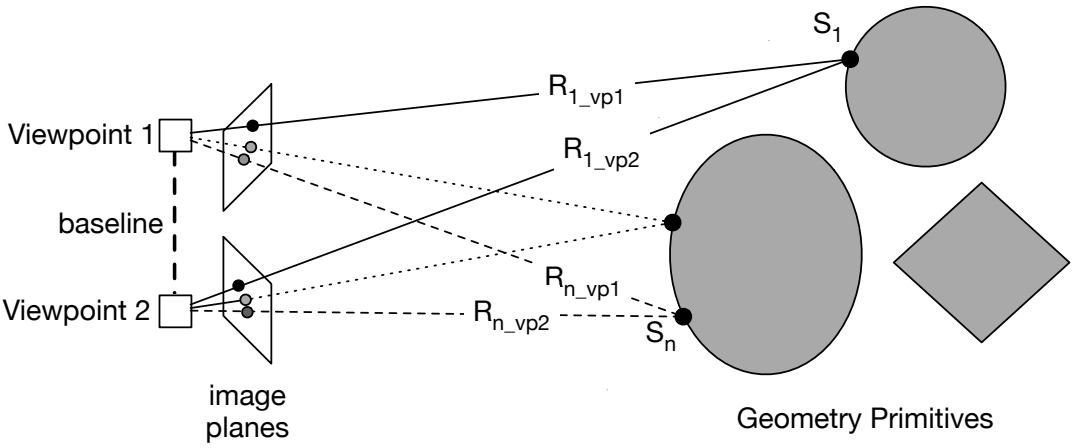


Figure 2.11: Triangulation scanners determine surface samples on a scene by computing corresponding pixels from two viewpoints, which in turn define a pair of rays in space. The intersection point between these two rays results in the surface sample position.

light active stereo. In all cases one of the viewpoints is equipped with a projector which projects a unique infrared pattern of dots, usually a grid, which an infrared camera (the second viewpoint) then uses to determine distance from objects.

Figure 2.12 illustrates the point cloud resulting from scanning a keyboard using the aforementioned scanners. In all three cases the keyboard is represented by around 20K samples. The top row shows the points sampled but doesn't clearly show the quality of the point cloud. In order to illustrate the difference between the point clouds, the second row shows a triangular mesh computed over these points. The samples acquired by the Microsoft Kinect (V1) sensor are clearly inferior to the other two sensors. The second and third point clouds are very similar, with the point cloud acquired via the Structure Sensor being slightly more accurate, for instance when sampling the large zero key on the keypad. Figure 2.13 illustrates the point cloud representing the same keyboard using less samples, resulting from an increase in the distance between the sensor and the keyboard. The surface details, in this case all the keys, which were visible before are now lost due to aliasing. Even though sample positions are computed correctly, not enough samples are available to reconstruct the high frequency surface details of the original signal. The number of samples representing the keyboard goes down from 20K to 1K when the distance between the scanner and keyboard is increased five-fold.

Time-of-flight (TOF) scanners using light detection and ranging (LiDaR)

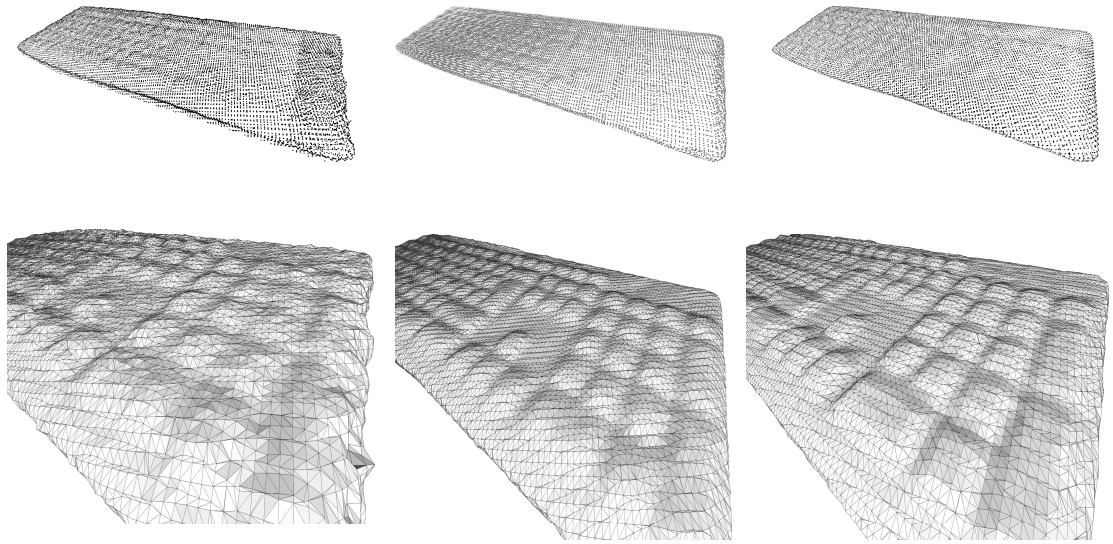


Figure 2.12: Point clouds acquired using the Microsoft Kinect, Asus Xtion and Structure Sensor triangulation-based scanners respectively. The Skanect software (Tisserand & Burrus, 2015) is used to extract depth information and carry out tessellation. MeshLab (Cignoni *et al.*, 2008) is used to render both top row point clouds and bottom row triangular meshes.

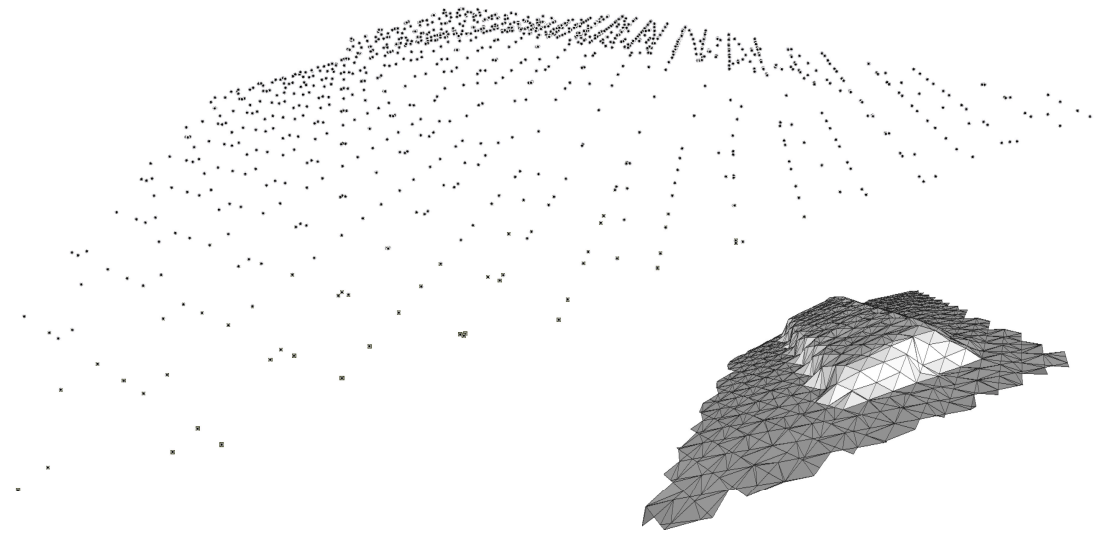


Figure 2.13: Point cloud of same keyboard scanned from a distance X5 larger than the point clouds in Figure 2.12 using the Structure Sensor. Part of the table is included in order to place the keyboard in context and make it easier to visualise. The point cloud consists of 1K samples.

principles are predominantly used to acquire very large scenes, although some can also be used for indoor scenes. As opposed to triangulation based scanners only one viewpoint is necessary to determine ray-object intersections. Fundamentally, TOF scanners measure the time it takes for a laser, either pulsed or modulated, to travel to the nearest object in a scene and back along the same path to a detector built into the scanner. The accuracy of TOF scanners depends on how accurately the round-trip time can be measured. Recent advances, for instance scanners using phase-shifting technology, have increased measurement accuracy (Zhang & Yau, 2006). A popular application of the LiDaR principle is that of making high-resolution terrestrial maps. Scanners are mounted on airborne systems in order to generate precise, 3D information of the Earth surface. Each point in the cloud has 3D spatial coordinates representing latitude, longitude and altitude. Figure 2.14 illustrates different scans produced by TOF scanners. The top row left-hand side point cloud illustrates a 360° scan of a green area at the University of Warwick. The right-hand side point cloud illustrates part of a scan representing a pre-historic temple in Malta. The bottom row illustrates a 20M samples point cloud of an urban area in Malta acquired via an airborne LiDaR scanner.

2.3.3 Quality

The quality of a point cloud P can be measured in a number of different ways. For instance, Pauly *et al.* (2004) present a framework for analysing shape uncertainty and variability in point-sampled scenes using a statistical representation that quantifies for each point, the likelihood that a surface fitting the data passes through that point. The resulting likelihood and confidence maps are then used to describe the quality of P. A less elaborate approach to measure the quality of a point cloud is given here, which takes in consideration object clutter and sample noise. Given a scene SC, decomposed into a set of surface patches SP, the quality of the point cloud representing it is a combination of:

1. the number of observed and acquired patches $s \in SP$
2. for each $s \in SP$, the signal to noise ratio in s
3. for each $s \in SP$, the contribution of each sample s towards reconstructing the input signal SC

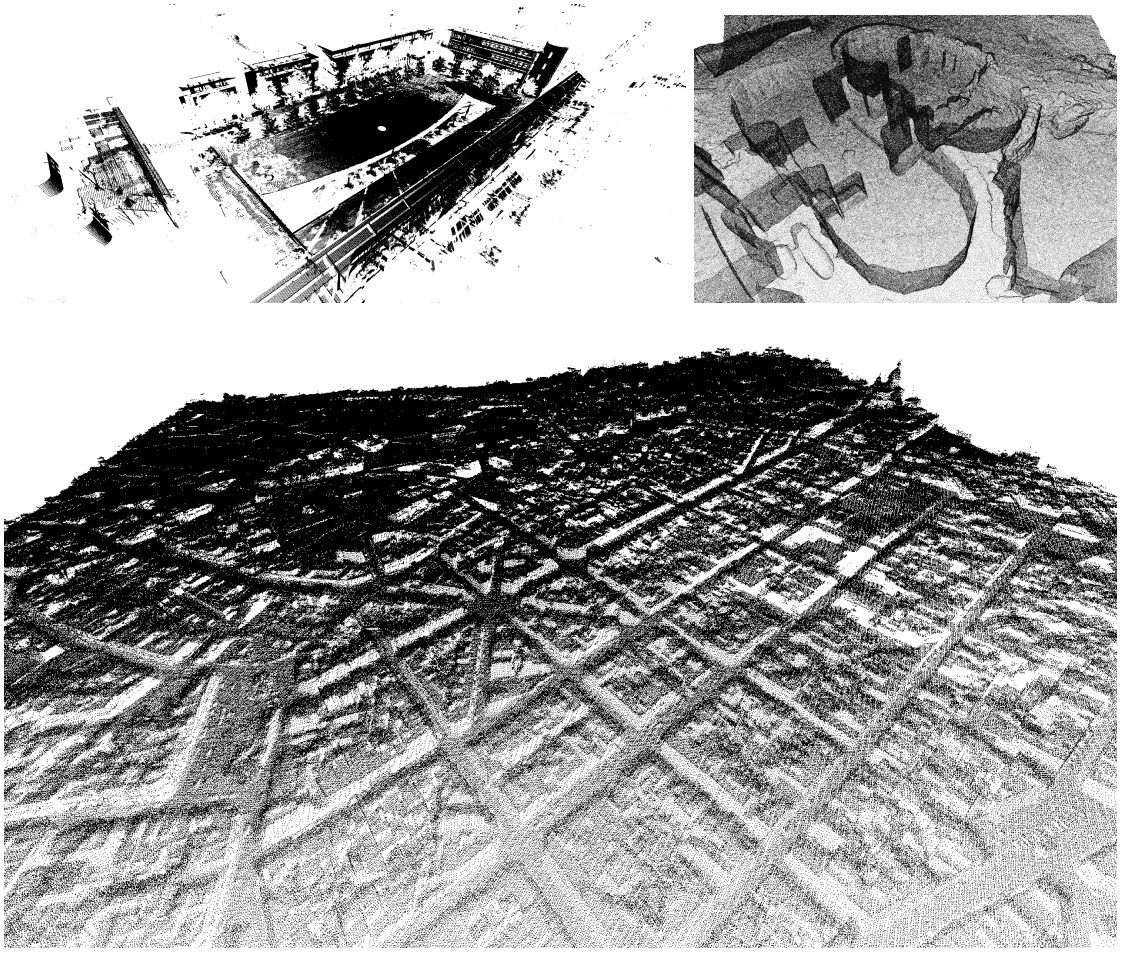


Figure 2.14: Point clouds acquired using time-of-flight scanners. Top left-hand side shows part a pre-historic temple, whereas top right-hand side shows an open space within the University of Warwick (scanned 2014, using FARO Focus3D scanner). Bottom image illustrates part of a LIDAR scan of the Maltese islands at an average density of 4 samples/m^2

These values are determined by the methodology used to acquire the point cloud, which usually takes into consideration properties of the scene (e.g. shiny or translucent materials, surface geometric detail), the hardware used for acquisition and the registration process which combines the different scans together. In general, the lower the quality of a point cloud with respect to the three points listed above, the harder reconstructing the scene is. Whereas a densely sampled point cloud might indicate high quality, this on its own does not guarantee that all patches in SP have been sampled but that at least, if high frequency surface changes are present in the signal represented by any of these surface patches, then there's a higher probability that the sampling rate used was adequate. Moreover, some patches $s \in SP$ might be more densely sampled than others notwithstanding the possibility that these particular patches are not geometrically complex (third criteria). For instance in Figure 2.13, there's no way to determine that samples are taken off from the surface of a keyboard rather than, for instance, the surface of a book. The point sampling rate is sufficient to represent the face of the table surface patch given that it's geometrically less complex. For every scan, sample density depends on the distance between the scanner and the surface hit by the ray. When multiple scans are registered together and combined, sample density increases in areas where scans overlap. The second criteria above caters for the fact that some noise is bound to occur in all scans and essentially results from the precision of acquired samples. With many scanners, sample precision is measured as a ratio of the sample spacing. Precision in computing ray-object intersections also depends on the material properties of the objects in the scene and whether sampling occurs at depth discontinuities between object surfaces.

2.4 Acceleration Structures for Storing Point Clouds

This section briefly describes data structures used to store point clouds. Given the size of point clouds produced by scanners, it is important to use data structures which scale efficiently with the size of the data. Naively, a set of points can be stored as a array, however this does not facilitate operations such as locating nearby points. In this work, both grid-based and tree-based data structures are used, which are briefly outlined next.

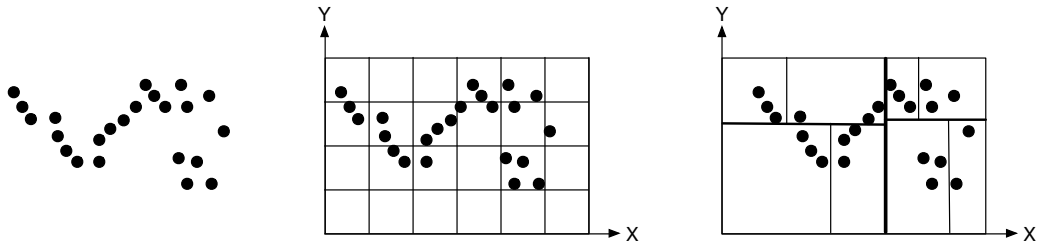


Figure 2.15: A set of points is partitioned using a uniform grid (12 cells) and a kd-tree with k=2 using median split to decide the parameters of the partitioning plane (8 cells).

2.4.1 Uniform Grid

Grid-based structures employ spatial hashing to organise points in a 3D grid. Figure 2.15 illustrates a set of points (left-hand side) partitioned using spatial information in a uniform grid (middle). Although very efficient to compute, a uniform grid does not adapt to the distribution of points. In this case, some cells have 1 point in them whereas others have four. If taken to the extreme, a uniform grid might degenerate to a simple list if all the points end up in a single cell. Cell size plays an important factor in the efficiency of the data structure.

2.4.2 Kd-tree

The kd-tree (Friedman *et al.*, 1977) is an acceleration structure that recursively subdivides space in two using an axis-aligned plane. There are various heuristics to determine both the splitting axis and the position of the partitioning plane. The initial axis can be chosen randomly or according to the most variance among the points. A median-split operation can be used to determine the position of the partitioning plane. The resultant spaces are then recursively partitioned. During each split, the points are added to the half-space in which they are contained. The recursion is terminated when the number of points in a leaf does not exceed a minimum amount specified or the depth of the tree has reached a predetermined value. The asymptotic time complexity for searching nearest points in a kd-tree is $O(n \log n)$ (see Algorithm 1). Figure 2.15 (right), gives an example of a 2D kd-tree, showing a set of points partitioned using the acceleration structure.

2.5 Local operations on points

This section details a number of operation of points which are generally used when processing point clouds.

2.5.1 Neighbourhood

The neighbourhood of a point p is defined using set comprehension as follows:

Definition The neighbourhood of point $p \in P$, within a distance r is the set $N_r(p) = \{q : P | distance(p, q) < r \bullet q\}$.

Given this subset of points from P, it is usually useful to limit the number of points and thus produce the k-neighbourhood (k-NN) of p. This is usually done by sorting points in $N_r(p)$ from closest to furtherest to p. If the size of $N_r(p)$ is less than k, this is increased by iteratively incrementing the distance r from p until $|N_r(p)| \geq k$. Acceleration structures, such as the kd-tree described above, are used to speed up the computation of the k-NN. Various implementations, based on Algorithm 1, exist which provide this functionality.

2.5.2 Principal Component Analysis

Principal component analysis (PCA) is used to measure data in terms of its principal components, representing the directions along which there is most variance, the directions where the data is most spread out. In the case of point clouds, PCA is normally applied to a subset of points, generally obtained using a k-NN process, to determine local properties at a specific point. The process computes three orthonormal eigenvectors and corresponding eigenvalues. The eigenvector with the highest eigenvalue represents the principal component, whereas the second and third successively represent lower variance.

2.5.3 Volumes

A number of convex polyhedra can be used to bound the space occupied by a set of points. These bounding volumes are then used to group together points, accelerate general operations by discarding the whole set of points when the operation does not interest the volume, and point decimation and interpolation, amongst others.

Algorithm 1 High-level description of k-NN computation using kd-tree

Input Point cloud $\{P\}$, $p \in P$, k, distance d.
Output Ordered List Q with maximum size k.
$o = \emptyset$
$r = d/2$
$nodeStack = \emptyset$
$nodeStack.push(root)$
while $nodeStack \neq \emptyset$ **do**
    $node = nodeStack.pop()$
    if $node.isLeaf$ **then**
        for each $q \in node.points$ **do**
            if $distance(p, q) \leq r$ **then**
                $o = o \cup q$
            end if
        end for
    else
        if $overlap(node.axis, p.Position[node.axis], -r, node.left)$ **then**
            $nodeStack.push(node.left)$
        end if
        if $overlap(node.axis, p.Position[node.axis], r, node.right)$ **then**
            $nodeStack.push(node.right)$
        end if
    end if
end while
$sort(o)$
$Q = \{o_1 ... o_k\}$

AABB

Axis-aligned bounding boxes (AABB) are quadrilaterally-faced hexahedra with the edges parallel to the major coordinate axes. AABBs have a very compact representation in that only two points are required to define the extents of the box. During construction, the extents are determined from the extrema for each coordinate axis, such that:

$$[min_x, min_y, min_z] = [\min(P_1^x, \ldots, P_n^x), \min(P_1^y, \ldots, P_n^y), \min(P_1^z, \ldots, P_n^z)]$$
$$[max_x, max_y, max_z] = [\max(P_1^x, \ldots, P_n^x), \max(P_1^y, \ldots, P_n^y), \max(P_1^z, \ldots, P_n^z)],$$

where n is the number of points in the set.

OBB

Oriented bounding boxes (OBB) are similar to AABBs but in general provide a tighter fit to the bounded data set due to relaxed constraints on axial-alignment. OBBs are constructed by finding the principal components of a point set using PCA. OBBs can still be represented using the extents, however, an orthonormal basis is also required to describe the orientation of the volume.

2.6 Decimation and Interpolation

One aspect of particular relevance to point clouds is sampling for decimation and interpolation purposes. Decimation strategies use sampling methods to selectively determine a subset of points from a larger set describing a surface. Sampling can either be random across the input, or guided for instance using importance sampling techniques in order to provide some guarantees on the distribution of samples chosen. Decimation can be seen as a down-sampling process, where the resulting samples are somehow representative of the set from where these points are sampled. Interpolation on the other hand, adds new points to the original data set. Random sampling strategies are also used to determine whether the points in a given set can be described as a collection of basic shape primitives (e.g. spheres and planes).

2.6.1 Point Set Decimation

Given a point cloud P, a sub-sampling process is used to select a number of representative points $Q \subseteq P$. This is usually done in order to minimise variability in the point density of a point cloud or to minimise the computational cost of applying an operation over a larger set of points. The cardinality of Q, unless specified a priori, is determined by spatial properties constraining points in P. Sub-sampling of P is carried out using Poisson disc sampling in conjunction with an accept-reject strategy. Algorithm 2 illustrates the sampling process. For an input set P, an initial point p is chosen at random and added to output set Q. The point is marked as invalid, and the next random point is chosen. p is tested against a minimum distance d_{min} from any point in Q. If no point is found in the neighbourhood then p is accepted; otherwise p is rejected. The process repeats until all points in P have been exhausted.

Algorithm 2 Point cloud decimation

Input Point cloud P, d_{min}.
Output Point cloud Q.
$Q = \emptyset$
$R = P$
$\chi = floor(\xi * |R|)$
while $R \neq \emptyset$ **do**
    $p = R[\chi]$
    $R[\chi] = R[\chi]/p$
    $reject = false$
    for each $q \in Q$ **do**
        if $distance(p, q) < d_{min}$ **then**
            $reject = true$
        end if
    end for
    if $reject = false$ **then**
        $Q = Q \cup p$
    end if
    $\chi = floor(\xi * |R|)$
end while

2.6.2 Point Set Up-Sampling

In order to increase the point density of a point cloud, up-sampling is often used. In this work, up-sampling is used in order to increase the size of a point cloud

for the evaluation of out-of-core methods. Whereas a variety of point cloud up-sampling algorithms exist (Weyrich *et al.*, 2004), since this work does not concern itself with the quality of the up-sampled point cloud a straight forward approach is adopted. Algorithm 3 describes the method used, which doubles the size of a point cloud per application. For each point $p \in P$, a k-NN query is used to return the two closest points p^m and p^n, and a new point is interpolated between the three. Note that the interpolation function used puts more weight on p in order to reduce the probability of generating new points with the same coordinates.

Algorithm 3 Point cloud up-sampling

1: **Input** Point cloud P, set Q.
2: **Output** Point cloud $P \cup Q$.
3: $Q = \emptyset$
4: **for** each p $\in$ P **do**
5: $(p^m, p^n) = \text{k-NN}(p, 2)$
6: $Q \cup \text{InterpolatePoint}(p, p^m, p^n)$
7: **end for**

2.7 Random Sample Consensus (RanSaC)

The RanSaC paradigm (Fischler & Bolles, 1981) has been proposed as a method for fitting geometric models to data, and for outlier rejection in noisy data. In the case of point clouds, it is used to determine whether points can be fitted to a number of parametric shapes. Shape parameters are extracted via random sampling of minimal sets from the input data, where a minimal set contains the smallest number of points required to uniquely define a given type of geometric shape. For instance, in the case of a plane primitive, the minimal set consists of three points. Figure 2.16 shows a set of points on a 2D surface fitted to two types of geometric shapes, namely line and circle. Points which do not fit within the set of extracted shapes are usually discarded as outliers on the assumption that these represent noise within the data set. An error threshold, represented by dotted lines on the right-hand side of Figure 2.16 represents the maximum distance from the specified shape surface.

Algorithm 4 outlines the basic RanSaC process. When applied to point clouds, the input consists of a point cloud P, a set of shape types to fit (e.g. S=$\{circle, line\}$ for Figure 2.16), an error threshold ε representing the maximum distance of a compatible point to the shape being checked, a trials count c

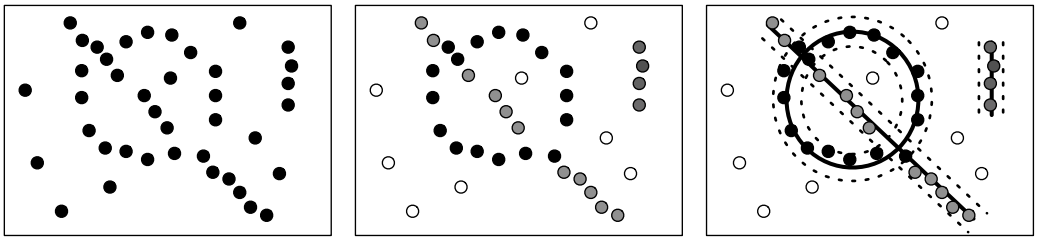


Figure 2.16: Unstructured points partitioned into two lines and one circle. Note that on the intersection between line and circle points can go to either of the shapes depending on the shape fitting order. Outliers (white dots in central figure) are removed.

Algorithm 4 High-level description of General RanSaC

Input Point cloud P, set of Shape Types S, Tolerance ε, Trials c, Acceptance r.
Output Set partition $\{Q\}$ of $\{P\}$.
while $fitmoreshapes$ **do**
    $bestscore = 0$
    $bestshape = < \emptyset, null >$
    for each s ∈ S **do**
        $trialscount = c$
        while $(trialscount > 0)$ **do**
            $crtshape_{minset} = \{p_{1..n} : P | p_1 \neq p_2 \neq \ldots \neq p_n\}$
            for each p ∈ P **do**
                if compatible$_s$(p, $crtshape_{minset}$, ε) **then**
                    inc($crtscore$)
                end if
            end for
            if $(crtscore > bestscore)$ **then**
                $bestscore = crtscore$
                $bestshape = < crtshape_{minset}, s >$
            end if
            $trialscount$-=1;
        end while
    end for
    $points_{bestshape} = \{p : P | \text{compatible}_s(p, bestshape, \varepsilon)\}$
    if $|points_{bestshape}| > (|P| * r)$ **then**
        $P = P \setminus points_{bestshape}$
        $Q = Q \bigcup \{points_{bestshape}\}$
    end if
    Determine whether $fitmoreshapes$ should be set to false
end while

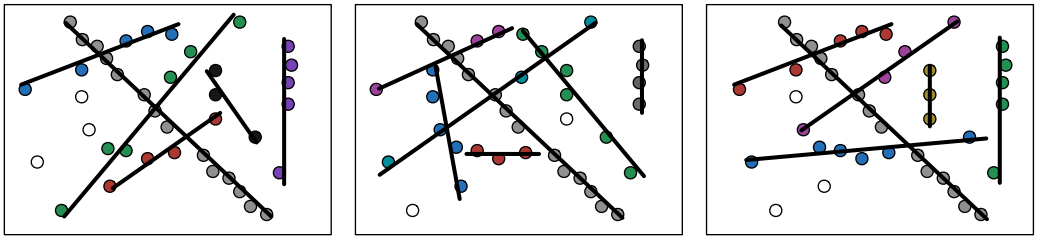


Figure 2.17: Randomness in the choice of support samples may lead to set partitions which are different from each other and still valid. In general, shape primitive fitting in the RanSaC paradigm is non-deterministic.

representing the number of a trials carried out per shape type and a minimum number of points ratio r with respect to the size of P, required to accept the shape. The algorithm iteratively extracts shape primitives by randomly choosing minimal sets from P (line 9) for the number of trials c specified. Within each iteration (lines 8-20) all points in P are checked for compatibility with the shape primitive described by $crtshape_{minset}$ (line 11). This process is carried out for all shape types (lines 6-21) and the best shape parameters which fit the data (points in P) are selected (line 17). The metric which is traditionally used, is the total number of points which fit within the shape parameters. The points compatible with the shape chosen are removed from P (line 23) and the set with the compatible points included in the set partition Q (line 24). The function which computes shape compatibility of points can vary, but usually takes in consideration the position and surface normal of points. The predicate $fitmoreshapes$ is used to establish whether to keep on fitting shapes. This predicate takes in consideration a number of parameters, such as the updated cardinality of P. If this is sufficiently low than $fitmoreshapes$ is set to false. Alternatively, this predicate can be set to false if during the previous iteration no shape is fitted to P. In all cases, the outcome of the algorithm is a set partition Q which describes a mapping between points in P and $|Q|$ shapes.

The set partition output by RanSaC is in general non-deterministic given than minimal sets are chosen randomly from the input data. Figure 2.17 illustrates a typical scenario where the same input is partitioned into three different and valid set partitions by fitting line primitives to the data. This might pose a limitation in terms of stability in that results obtained might not be replicable. Notwithstanding, RanSaC also has many desirable properties. It is conceptually easy to understand, is very general thus allowing its application in many domains

and most important can deal with data containing more than 50% outliers (Roth & Levine, 1993). Without optimisations, as outlined in Algorithm 4, its major drawback is the high computational demand. The complexity of RanSaC stems from two factors; the number of minimal sets that are randomly drawn (trials) and the cost of evaluating the score for every candidate shape. In both cases, execution time depends on the size of P. A number of improvements over the standard RanSaC algorithm exist which addresses efficiency. These are further discussed in Chapter 3, which discusses how RanSac has been used in the context of point cloud segmentation.

2.8 Summary

This chapter has provided a background to concepts related to point cloud processing. First, a brief introduction to sets as collections of typed objects is given, in addition to operators which generate and manipulate these sets. Graphs are then discussed, together with transition trees which are used to measure compatibility between graphs. Point clouds are then defined in terms of sets, in addition to a brief overview of different sample acquisition methods and point cloud qualities. Next, data structures used to accelerate point based computations are presented, followed by neighbourhood, component analysis and volume generation processes. Point cloud decimation and interpolation are then described. Finally, the Random Sample Consensus paradigm is presented, highlighting its advantages and limitations. This background is provided as a basis for the segmentation, object recognition and scene understanding techniques described in the next chapters.

CHAPTER 3

Segmentation of Point Clouds

A segmentation process computes a set partition (§2.1.2) over some input data. Various segmentation algorithms exist, tailored for a variety of input data including images, 3D meshes and point clouds. Image segmentation has largely been investigated within the computer vision and image processing community, with the purpose of identifying specific objects. For instance, a common application is the recognition of faces in photographs or the movement of people in a sequence of images. For a detailed survey of algorithms in this area, the reader is referred to Pal & Pal (1993) and Zhang *et al.* (2008). Segmentation of 3D meshes (Gumhold *et al.*, 2001; Lavoué *et al.*, 2012) has also received considerable attention, mainly due to its importance in 3D object recognition and 3D object retrieval methods (Daras & Axenopoulos, 2010; Li *et al.*, 2012), which use the computed partitions to improve matching results. The methods employed in these segmentation algorithms generally assume that the input consists of a continuous surface definition, for instance, a set of triangle primitives over vertices. The reader is referred to Shamir (2008) and Chen *et al.* (2009) for a comprehensive survey of techniques and applications. This chapter specifically focuses on the segmentation of raw point cloud data. In a number of cases, this problem may be more straightforward than when using images, as the input might circumvent some of the ambiguities induced by the 3D to 2D projection of images, but in other cases is harder due to the lack of colour cues. Additionally, since a point cloud consists of a discrete surface definition, multiple plausible continuous surface definitions may exist as in Figure 3.1. Depending on the acquisition methodology used (§2.3.2), the resultant point clouds are often noisy and irregular in terms of point densities across the scene. For a point cloud P, consisting of a set of geometric points p, segmentation computes a set partition

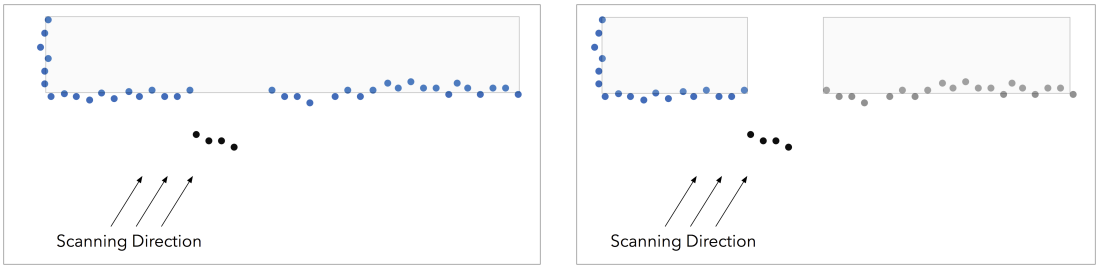


Figure 3.1: Ambiguity in point cloud (and image) segmentation algorithms as opposed to 3D object (mesh) segmentation. Two alternative solutions.

with each element containing a subset of P. Whereas set comprehensions as the one listed below, which partitions P into two sets may sometimes be sufficient to produce the required segments, a more elaborate general-purpose process is usually required.

$$above_{ground} = \{p : P | coord_y(p) \geq estimate_{ground}\}$$

The simple partition induced by the set comprehension above may be sufficient to discriminate between points that are on the ground and those that aren't, but this will only work in very specific situations. For instance, Zhou *et al.* (2014) produce an initial set partition consisting of ground and above-ground points given continuous ground level information acquired using a vehicle LiDaR scanner. In general, the segmentation of a raw point cloud is used to determine primitive patches which are later used as building blocks for the identification of objects and structures. Figure 3.2 illustrates a typical bottom-up object recognition hierarchy. Each layer can be considered as a different set partition, with the bottom layer consisting of primitives resulting from the segmentation process. Object partitions are constructed from the union of primitive elements, given these satisfy a number of discriminative conditions. These groups are then further composed into objects to finally produce a set partition of P representing a scene consisting of a hierarchy of groups. Clearly, the segment primitives produced at the base (leaves) of this hierarchy are critical to a successful application of the recognition process. This chapter, reviews a variety of methods which are used to produce these primitives.

In a number of cases, for instance Anguelov *et al.* (2005), segmentation processes have been presented which label subsets of points as specific objects, which is more akin to 3D object recognition methods. In this dissertation, these methods are categorised as either 3D object recognition or scene understanding solu-

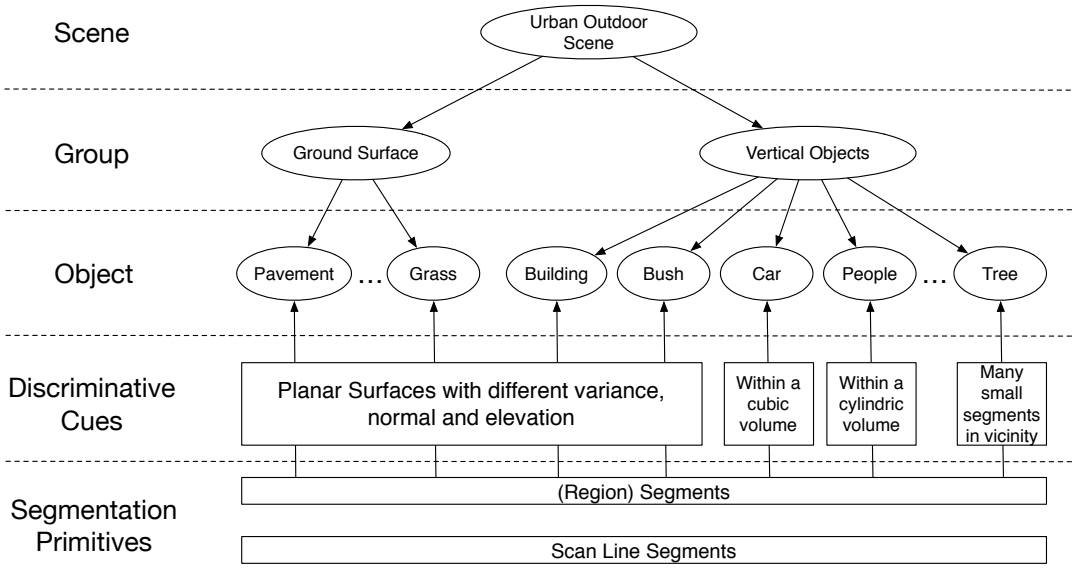


Figure 3.2: Segmentation may be used to determine the primitives which are later used to identify objects present in a point cloud. Segmentation is responsible for the lower layers of the hierarchy above. Upper layers are the responsibility of object recognition and scene understanding algorithms. Diagram based on Zhao *et al.* (2010)

tions, and are discussed in Chapter 4. Since a point cloud can be constructed from a set of range images, some segmentation algorithms (e.g. Gotardo *et al.* (2003) and Bab-Hadiashar & Gheissari (2006)) take range images as input. Hoover *et al.* (1996) propose an experimental setup for comparison of range image segmentation algorithms. The reader is referred to Zhang *et al.* (2008) and Sonka *et al.* (2014) for a survey of segmentation techniques specifically used for range images. In the case of raw point cloud data, segmentation methods are based on either a parametric shape fitting or a region-growing process. In both cases, the aim is to produce a set partition of primitive segments, where each element represents a component in the construction of some object or structure during a recognition/identification phase.

3.1 Segmentation using Region-Growing Algorithms

This section provides an overview of segmentation algorithms which produce a set partition whose elements represent surface patches adhering to some particular characteristics, for instance points with similar surface normals, or a spatially isolated cluster of points. The region growing approach to segmentation has been

studied for several decades in computer vision, where it is often formulated as a graph clustering problem. Instances of such approaches are graph cuts (Boykov & Funka-Lea, 2006), including min-cuts (Wu & Leahy, 1993) and normalised cuts (Shi & Malik, 2000). The graph-cuts algorithm has been extended to point clouds by Golovinskiy & Funkhouser (2009) using the k-NN (§2.5.1) function to build a graph and assign weights to edges according to an exponential decay in length between points. The method proposed requires prior knowledge of an object's location, and is used to distinguish between points making up the object and surrounding clutter.

Algorithm 5 describes the basic process carried out by a region growing segmentation algorithm. A similarity function (line 14) is used to compare two neighbouring points together and establish whether they should form part of the same region. Common similarity functions take in consideration local surface normals and curvature. A tolerance value is used to determine the maximum distance between point properties. Increasing this distance, generally results in larger regions (under-segmentation), whereas decreasing this distance generally results in smaller regions (over-segmentation). The similarity function may either check distances between the two neighbouring points, the one that is already in the region and the one that is being tested for membership, or the new point against the region seed point. For instance, when growing regions with similar surface normals, in order to avoid incremental variations in surface normals resulting from the region growing process, new points are checked against the original seed point. A number of variations to the standard region growing process have been described, mainly adopting different seeding and region-growing criteria depending on the information which is available with the data. Vosselman *et al.* (2004), Pu *et al.* (2006) and Mattausch *et al.* (2014) utilise a region-growing algorithm which results in a set partition of P containing elements which are locally planar. Points with the lowest local curvatures are used as seed points, and candidate points are only accepted if the orthogonal distance of the candidate point to the plane associated with the region is below some threshold. Clearly, this segmentation process favours points clouds which can easily fit within a set of plane primitives and results in a considerable number of planes if the point cloud contains many curved surfaces.

Pauling *et al.* (2009) propose the use of ellipsoidal region growing in order to segment the point cloud into clusters of points contained in ellipsoids. Their method merges initially computed ellipsoids into larger ellipsoidal segments using

Algorithm 5 Generic Region Growing Process

Input Point cloud P, distance d_{min}, k neighbours, tolerance δ, queue *seeds*, queue *region*.
Output Set Partition Q of P.
$Q = \emptyset$
$seeds \leftarrow^{enqueue}$ SelectSeed(P)
while $seeds \neq empty$ **do**
    $Q_{region} = \emptyset$
    $seed \leftarrow^{dequeue} seeds$
    $Q_{region} \leftarrow^{add} seed$
    $region \leftarrow^{enqueue} seed$
    while $region \neq empty$ **do**
        $n = \leftarrow^{dequeue} region$
        $neighbours =$ k-NN(n, k, d_{min}) ▷ see Algorithm 1
        for each $p \in neighbours$ **do**
            if NotVisited(p) $\wedge$ Similar(p, n, δ) **then**
                $Q_{region} \leftarrow^{add} n$
                $region \leftarrow^{enqueue} n$
            else
                $seeds \leftarrow^{enqueue} n$
            end if
        end for
    end while
    $Q \leftarrow^{add} Q_{region}$
end while

a minimum spanning tree algorithm (Graham & Hell, 1985). Merging of ellipsoids is based on two criteria, namely shape and density distances. The former metric considers orientation and position of the ellipsoids whereas the latter takes into account a minimum sampling density. The process produces a set partition on the input, with the resulting ellipsoids somehow representative of the underlying data. In general the method is very subjective and tends to group together patches across different objects which are not related.

Moosmann *et al.* (2009) describe an algorithm tailored for the segmentation of objects from the ground in non-flat environments. The method uses the physical setup of the scanner to turn the data into a 2D graph with edges connecting all scanned points to their respective four neighbours. A local convexity value is computed for each edge in the graph and a region-growing process then grows seeds nodes into segments. The method heavily depends on the availability of scanner position data in order to produce the graph and therefore cannot be applied to generic point clouds.

A fast and accurate plane segmentation algorithm is presented by Deschaud & Goulette (2010) with the purpose of reducing the number of points in a point cloud via a decimation process over the detected planar segments. The process is based on an improved point normal estimation method followed by robust voxel region growing. Points are sorted in ascending order of local planarity, using a score which takes in consideration noise in the normal estimation of points. The algorithm is evaluated on a small number of points clouds, and results in slightly improved plane parameters in the presence of noise.

Douillard *et al.* (2011) present a set of segmentation methods intended to cover various types of point clouds ranging from the very dense to sparse, which neither assume the ground to be flat nor require a priori knowledge of the location of the objects to be segmented. Different models based on grids, Gaussian process and meshes are considered for representing and segmenting the ground. All segmentation processes are a combination of ground modelling and extraction, followed by region-growing voxel-based clustering methods on the remaining data. The different segmentation methods are evaluated on a number of points clouds, all consisting of a sufficiently large ground segment, since all methods depend on its correct identification. Results provide empirical evidence of the benefit of ground extraction prior to object segmentation.

3.2 Segmentation using Primitive Shape Fitting

Primitive shape fitting within a point cloud has been applied to a variety of tasks, for instance in the work by Lafarge & Alliez (2013) which uses plane fitting to produce an optimal surface reconstruction from a point set. This section provides an overview of segmentation algorithms, which produce a set partition whose elements contain points consistent with primitive parametric shapes. An additional element, containing points which are not assigned to any primitive, is usually added to the set partition for completeness. The plane is the most popular geometric primitive used in a shape fitting process, as many scenes naturally consist of a considerable number of planar surfaces.

Chaperon *et al.* (2001) propose the use of RanSaC and the Guassian image of a cylinder to extract cylinder primitives from point clouds. The process of mapping a point on a surface, to the unit normal of the surface at this point, is called the Guass map (Do Carmo & Do Carmo, 1976). The Guassian image is obtained by applying this process to a set of points, which in the case of a cylinder is represented by a circle. The extraction process is split in two parts following the computation of the Guassian image for the whole scene. RanSaC is first used to extract constrained planes in the Guassian image, and then for each set of points, cylinders are extracted. Their method is shown to work on point clouds of pipes in an industrial setting where the cylinder primitive is sufficient to describe the scene. In a more general context, the algorithm would struggle to produce a meaningful set partition.

Nistér (2005) carries out segmentation on a point cloud generated by a calibrated perspective camera in the context of a structure from motion system. Their system is capable of performing robust live ego-motion estimation by using RanSaC to estimate a pre-determined fixed number of candidates given the time constraints. In the general case however, where the number of shape primitives is unknown, this method cannot be applied.

Schnabel *et al.* (2007) describe a novel sampling strategy for RanSaC resulting in an efficient shape detection process which outputs a set partition consisting of planes, spheres, cylinders, cones and tori in addition to a set of remaining points. Their approach addresses the ever increasing size and complexity of point cloud data by improving both shape detection quality and performance of the basic RanSaC process (§2.7). These improvements are achieved by adopting a hierarchical structured sampling strategy for candidate shape generation as well

as a novel, lazy cost function evaluation scheme which significantly reduces the computational costs associated with RanSaC. The sampling strategy is built on the observation that shapes are local phenomena, i.e. the a priori probability that two points belong to the same shape is higher the smaller the distance between the points. This fact is exploited in the sampling strategy used in order to increase the probability of drawing minimal sets belonging to the same shape. A kd-tree acceleration structure is used to organise the points and minimal sets are only drawn from adjacent cells in the tree. This approach favours the detection of small shapes in a point cloud, and may miss global shapes in the input. Schnabel *et al.* (2008) adopts this RanSaC method and creates a topology graph which connects adjacent extracted shapes. A user-specific value is used to determine the maximum distance between primitives. The same method is used by Schnabel *et al.* (2009) to address the problem of completing and reconstructing models using primitive segments.

For the purpose of reconstructing 3D building models of a city, Tarsha-Kurdi *et al.* (2007) present a method specifically designed for the automatic detection of building roofs from LiDaR data. Both Hough transform and RanSaC methods are proposed for the detection of planes in the point cloud, with considerable performance and sensitivity advantages obtained bt RanSaC. Their main contribution is twofold; an extension to the RanSaC process, which in addition to point count also consider standard deviation to the optimal plane parameters. Moreover, they adapt the mathematical aspect of the algorithm with the geometry of a roof. These enhancements result in improved detection rates of roofs, however the geometry of the detected roofs is similar throughout the data.

Oehler *et al.* (2011) describe a multi-resolution method to partition a point cloud into planar segments using a combination of the 3D Hough transform and RanSaC for robust fitting. In order to improve efficiency, a coarse-to-fine resolution strategy is utilised. Local surface normals are first extracted at multiple resolution, over which the 3D Hough transform is applied to determine co-planar points. At each iteration, those points which are not explained by current plane primitives, are further processed using the Hough transform. At each iteration, RanSaC with the extracted planar parameters is used in order to improve accuracy and robustness. At the finest resolution, co-planar plane segments are merged and the remaining points distributed. The method is evaluated on a number of Kinect range images and point clouds from mounted laser scanners. The approach is shown to find the major plane segments of a scene and produced

good results in cluttered regions where sampling density is sufficient. Borrmann *et al.* (2011) also use the 3D Hough transform for detecting planes in a point cloud. They propose a new accumulator design intended to improve detection accuracy by making use of a randomised selection of points. It is shown to compare favourably with a region-growing method (Poppinga *et al.*, 2008). In a similar fashion to Oehler *et al.* (2011), the Hough transform method favours the detection of the principal structures in an indoor environment.

Algorithms for sphere detection in point clouds are proposed by Abuzaina *et al.* (2013) and Camurri *et al.* (2014) using the 3D Hough transform. In the first case, a fast and accurate sphere detection process is designed and evaluated on Microsoft Kinect generated point clouds. Performance is gained by uniformly down-sampling the original point cloud therefore resulting in less points undergoing the expensive computation of voting for parameters C_x, C_y, C_z and r, representing the centre and radius of the sphere. A trade-off between down-sampling levels and robustness of the detection process is determined. Camurri *et al.* (2014) propose a hybrid approach, referred to as the combined multi-point Hough transform (CMHT), which first identifies a region of interest using a single point accumulator, followed by a multi-point algorithm which refines the final detection. In practice, this improves the sphere recognition rates significantly. Both sphere detection methods make a number of assumptions on the input point clouds, in particular, that the data contains at least one spherical object (e.g. ball, apple) and therefore cannot be applied on generic point clouds.

3.3 Discussion

Segmentation algorithms play a critical role in point cloud processing pipelines, with the automatic set partitions produced directly contributing towards facilitating a variety of tasks across different domains. In a robotics scenario, the resulting segments may be used to identify obstacles and safely navigate an environment. Airborne LiDaR acquired point clouds have been used for tasks such as urban and transport planning to flood modelling. All these cases require some form of partitioning of the input data. The methods presented in this chapter broadly fall under two approaches: region-growing or shape fitting. In the former case, several criteria have been used to establish the boundaries of regions, and in the latter, RanSaC and Hough transform processes are parametrised with

the shape primitives to search for. Region-growing traditionally is sensitive to noise in point clouds, whereas shape fitting is more robust to data outliers but is usually biased towards determining the global principal segments of a point cloud. This is particularly highlighted in methods which adopt the 3D Hough transform (Oehler *et al.*, 2011; Borrmann *et al.*, 2011) which favour large planar segments across the scene. In relatively common scenarios where parts of multiple objects share the same plane parameters, for instance, an indoor scene with multiple desk tops or chair seats at the same height from the ground, this approach does not produce ideal segment primitives from which to distinguish between the different objects. Conversely, the RanSaC approach is expensive when applied globally and does not guarantee consistent segmentation results. This problem is addressed in the work by Schnabel *et al.* (2007) which takes advantage of the spatial locality of small shape primitives in a point cloud, and only draws support points from neighbouring nodes of a kd-tree acceleration structure. This provides for a very precise fitting of small primitive shapes, but may lose on larger global shapes. Additionally, parts of the point cloud which do not fit within any of the primitives used are grouped together as one segment. In an ideal scenario, these remaining points are still partitioned into a number of regions. Some segmentation algorithms are intended and optimised for specific inputs, for instance Abuzaina *et al.* (2013) and Camurri *et al.* (2014) which focus on the use of the 3D Hough transform for the identification of spheres in a point cloud. Similarly, Chaperon *et al.* (2001) focuses on the identification of cylinder primitives. A number of segmentation techniques make the assumption that a ground segment exists in the point cloud and depend on its identification before clustering the rest of the points (Douillard *et al.*, 2011; Moosmann *et al.*, 2009).

Whereas region-growing approaches tend to assign all the points in an input point cloud to segments, methods based on RanSaC (or the 3D Hough transform) may miss patches which do not fit any specific primitive. In scenes where there are no obvious mappings between points and shape primitives, for instance in some CH sites, a segmentation process should consider producing segments of different types, namely ones which describe shape primitives (e.g. planes, spheres) and others made up of free-form continuous patches. Chapter 5 proposes a novel segmentation algorithm, PaRSe, which seeks to address these limitations and takes advantages of both region growing and RanSaC shape fitting. PaRSe, which only assumes position information in the data set, partitions an input point cloud into segment primitives embellished with a graph representation describing connectiv-

ity between different types of segments. The resulting structure graph can then be used to cluster segments into more complex user-defined structures. PaRSe is designed to be generic and has been applied for the segmentation of point clouds from different fields, acquired using a variety of acquisition methodologies.

3.4 Summary

This chapter has provided a literature review for point cloud segmentation methods. Table 3.1 lists these methods, which fall within two main categories, namely those based on a region-growing process and those which employ a shape fitting process. For each method, a short description including category, is given. The following chapter presents a literature review on 3D object recognition and scene understanding methods from point clouds, many of which depend on a segmentation process prior to the recognition of objects in the scene.

Reference	R/S	**Comments**
Vosselman *et al.* (2004)	R	Boundary criteria - consistency with plane parameters of seeds
Pu *et al.* (2006)	R	Boundary criteria - consistency with plane parameters of seeds
Golovinskiy & Funkhouser (2009)	R	Min-cut approach using prior knowledge of the object location
Pauling *et al.* (2009)	R	Ellipsoidal region growing
Moosmann *et al.* (2009)	R	Object clustering above ground, requires physical setup of the scanner
Deschaud & Goulette (2010)	R	Robust voxel region growing using improved point normal estimation
Douillard *et al.* (2011)	R	Ground identification followed by region-growing voxel-based clustering methods
Mattausch *et al.* (2014)	R	Boundary criteria - consistency with plane parameters of curvature sorted seeds
Chaperon *et al.* (2001)	S	Extract cylinder primitives using RanSaC and Gaussian image of cylinder
Nistér (2005)	S	Identification of a fixed number of primitive shapes
Schnabel *et al.* (2007)	S	RanSaC shape fitting using locally sampled supports
Tarsha-Kurdi *et al.* (2007)	S	RanSaC for the automatic detection of building roofs from LiDaR data
Oehler *et al.* (2011)	S	Plane fitting using a coarse-to-fine resolution strategy, interspersing Hough transform and RanSaC methods
Borrmann *et al.* (2011)	S	Improve Hough transform accumulator for detecting planes
Abuzaina *et al.* (2013)	S	Sphere detection using Hough transform on down-sampled input
Camurri *et al.* (2014)	S	Sphere detection using Hough transform on identified regions of interest

Table 3.1: Summary of segmentation techniques; Reference, *R*egion Growing/*S*hape Fitting, Comments

CHAPTER 4

Object Recognition and Indoor Scene Understanding

The exponential growth of the web has played a critical role in the advancement of traditional text search engines, which have nowadays become standard tools used by many for both work and entertainment. Shape retrieval methods address a problem similar to traditional text-based searching, where instead of retrieving web pages based on some syntactic and/or semantic similarity metric, shape retrieval methods are used to search for objects in 3D model repositories. These repositories are increasing in size and popularity as more 3D acquisition methods are created and made available to the general public in the form of smartphones and tablets (Google, 2014). Several techniques adopt a mechanism whereby objects similar to a query model are searched for. The similarity metric may be based on a number of factors, which measure the distance between a query model and object descriptions in the repository. Since a function based solely on the explicit representation of objects, e.g. their respective surface meshes, would be computationally impractical in most cases, a mechanism is usually employed to capture and efficiently synthesise a descriptor which encodes the salient characteristics of an object. These descriptors augment the explicit representation of an object with an implicit representation which indirectly encodes the shape of an object using an intermediate form, for instance, a histogram of surface normals. A variety of object descriptors have been proposed for this purpose, which are then used to measure the similarity between objects. These descriptors are key to the efficient indexing and searching of ever growing 3D repositories (Bustos *et al.*, 2007). For a comprehensive survey of shape retrieval algorithms, the reader is referred to Lew *et al.* (2006); Tangelder & Veltkamp (2008); Li *et al.* (2012).

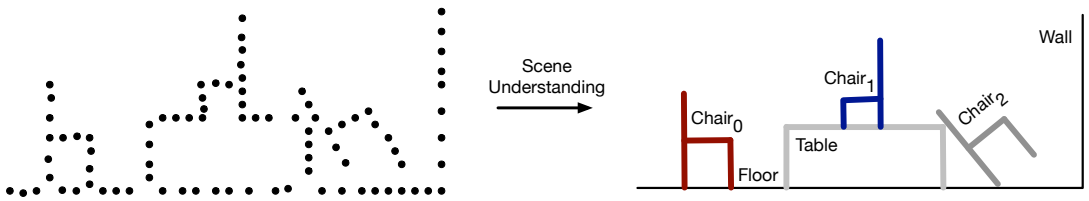


Figure 4.1: Scene understanding takes as input a point cloud P (left hand side) and a set of objects O, partitions P and creates a relation between $p \in P$ and $o \in O$. In this example, scene understanding should compute the set partition $\{chair_0, chair_1, chair_2, table, floor, wall\}$ of P.

Object descriptors also play a critical role in object recognition and scene understanding methods. These descriptors are usually based on either visual appearance or shape features of an object, or both. Computer vision algorithms based solely on an object's visual appearance, assuming clear distinguishing features are present, have been used to search for objects in an RGB image. A number of techniques augment appearance with shape features by using RGBD images, with D representing the distance between pixel and sensor. Lai *et al.* (2011) demonstrate that the inclusion of shape information is mostly beneficial in the case of class (or category) recognition. Since the focus of this thesis is in the application of these techniques to raw point clouds which do not include appearance information, these methods are not reviewed here and the reader is referred to Juan & Gwun (2009) and Rublee *et al.* (2011) for a comparison of appearance-based techniques. The object recognition techniques discussed in this chapter address the problem of determining which objects are present in a point cloud P, given a trained set of object O using only shape features. Similarly, scene understanding addresses the problem of identifying the elements of a scene represented as a point cloud P, given sets of trained objects O and/or scenes S. Both object recognition and scene understanding can be viewed as a correspondence problem between the elements of a set partition of a point cloud P and objects O. Whereas object recognition techniques mainly rely on descriptors based on local surface properties of an object in order to determine whether these are present in P, scene understanding techniques usually also take in consideration properties of the scene and synthesise context-sensitive scene descriptors. Object recognition techniques have mostly been used to identify free-form objects lying on some flat surface such as a tabletop whereas scene understanding techniques, spearheaded by the widespread availability of commodity 3D scanners,

have recently been applied extensively to the identification of objects (e.g. furniture in Nan *et al.* (2012)) and structures (e.g. walls in Adan & Huber (2011)) in indoor scenes. Methods addressing this specific problem fall within the indoor scene understanding category. Figure 4.1 illustrates an example of the typical problem tackled in indoor scene understanding.

Techniques for indoor scene understanding and shape recognition can be categorised in a number of different ways, for instance, those which require a training process versus those that don't. Figure 4.2 shows a flowchart illustrating the main components of these algorithms, with the two main blocks representing the training and the searching phases. All object recognition techniques require a training process to produce object descriptors which characterise and describe objects potentially present in P, prior to recognition. This is not always the case for indoor scene understanding. If a training process is carried out, specific scene information extracted from S (e.g. location of floor in scene, upward direction, object scale, spatial relations between objects) is usually embedded within the object descriptors during training. Alternatively, some scene understanding techniques rely on the presence of repetitions and symmetry in P in order to identify similar segment clusters which could represent objects. Within the training block, a distinction exists between those algorithms that embed specific scene parameters in the trained model representations and those that don't. Within the searching block, user input might be required to initiate the search process or may be optional in order to improve on the results obtained. In what follows, algorithms designed for object recognition from point clouds using trained object descriptors are reviewed first (§4.1), followed by techniques designed for indoor scene understanding (§4.2). A feature comparison of the methods used for indoor scene understanding is then carried out (§4.3), in order to highlight current research gaps and propose an alternative approach which addresses some of these limitations.

4.1 Object Recognition from Point Clouds

The recognition of objects from point clouds representing real scenes is generally more difficult, than for 3D models, due to increased clutter and occlusion. The quality of P (§2.3.3) may vary depending on the acquisition pipeline used, the material properties of the objects present in the scene and the scanner used for

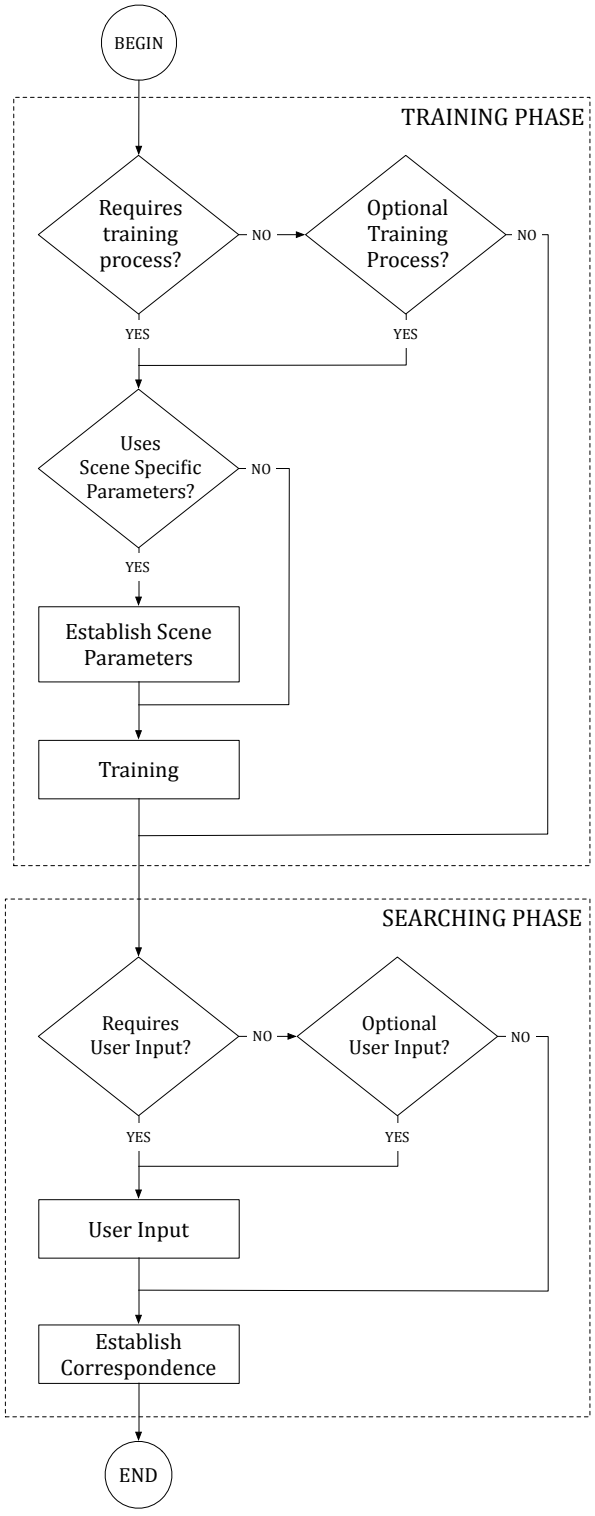


Figure 4.2: Two main blocks in shape recognition and scene understanding techniques include (optionally) a training phase followed by a searching phase.

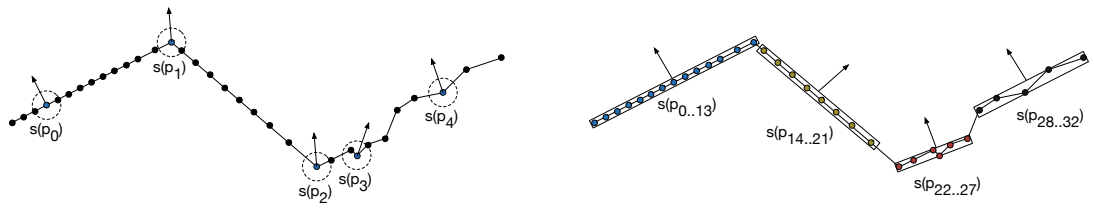


Figure 4.3: The surface of a shape o is described on the left hand side via a set of point-based local surface features $D_l(o)=\{s(p_0)\ \ldots\ s(p_4)\}$, whereas on the right-hand side the segments produced via a RanSaC plane fitting process are all used to compute $D_g(o)=\{s(p_0 \ldots p_{13})\ ,s(p_{14} \ldots p_{21}),s(p_{22} \ldots p_{27}),s(p_{28} \ldots p_{32})\}$. Both may be used to produce $D(o)=D_l(o) \oplus D_g(o)$

sampling. Some techniques replace P with a set of RGBD images. Clearly, these images can be transformed into P, but the inverse is usually not possible without knowing the camera parameters of these images.

All object recognition techniques described in this section try to establish a correspondence between trained objects in O and partitions of P. The training process results in the creation of object descriptors associated with each object. These descriptors may require a segmentation process on the object, a decision which will influence the design of the similarity function between objects. When segmentation is not carried out, an object is described as a set of local surface properties based on individual points and their surrounding geometry. For instance, given an object $o \in O$ consisting of n sampled surface points, a very basic object descriptor consists of the set containing n points each storing information about its surrounding (local) surface characteristics (e.g. curvature and surface normal inferred from k-NN queries). If a segmentation process is carried out on o, then the descriptor would consist of properties associated with the resulting elements of the set partition. In many of the techniques discussed below, which represent an object as a set of local surface properties, no prior segmentation process of P is required. Correspondence relies on identifying point descriptors in P which are similar to those of the trained objects in O followed by some verification procedure, e.g. iterative closest point (ICP) (Chetverikov *et al.*, 2002), to confirm the match. This is usually the case when P represents a collection of cluttered objects on a flat surface. In a number of cases, especially when P represents a more complex scene, segmentation is an important pre-processing step and object descriptors would be based on segment properties. Vosselman *et al.* (2004) highlight the importance of segmentation for recognising structures

and shapes in laser scanned point clouds and how this is applied to urban planning, industry and forestry documentation. In general, object descriptors may be defined as follows:

$$D(o) = \{s_l(p_0), s_l(p_1), \ldots, s_l(p_n)\} \oplus \{s_g(p_0 \ldots p_d), s_g(p_e \ldots p_k), \ldots, s_g(p_v \ldots p_z)\},$$

where $D(o)$ is a function that computes a descriptor for object o as a set of features each representing either point-based $s_l(p_i)$ or patch-based $s_g(p_{i..k})$ surface properties. Note that usually either one of the two sets is empty depending on what information is required to synthesise a particular object descriptor. Figure 4.3 illustrates this difference with a simple example. Given these sets of features, different data structures have been proposed to store them including bin-based histogram approaches and others based on graphs. Depending on the technique, the $\oplus$ operator takes these two sets and creates a descriptor which encapsulates and merges relevant surface information. Object recognition techniques using descriptors which do not rely on a segmentation process are described first (§4.1.1), followed by those which are built on top of segmentation (§4.1.2).

4.1.1 Point-Based Object Recognition

Besl & Jain (1985) and Faugeras & Hebert (1986) are amongst the first to discuss object recognition from range images and make use of local surface descriptors based on the geometric properties measured in a neighbourhood of a point. In the former, points are characterised according to the signs of their mean and Guassian curvatures and then classified as either peaks, pits, ridges and valleys. In the latter, local curvature is used for detecting primitive features such as lines, planes and quadratic patches in range data scenes. These properties are then used to detect similarities between trained objects and range images.

Point signatures used to describe 3D free-form surfaces were proposed by Chua & Jarvis (1997). The representation describes the structural neighbourhood of a point and is invariant to both rotation and translation. Recognition is achieved by matching the signatures of data points representing the scene to the signatures of data points representing trained models and has been used in scenes containing some partially overlapping models. Point signatures are created for all points in a model, where for each point p, a sphere of radius r is first positioned centred at p. The intersection of the sphere with the object surface produces a 3D space curve, whose orientation can be defined by an orthonormal basis formed

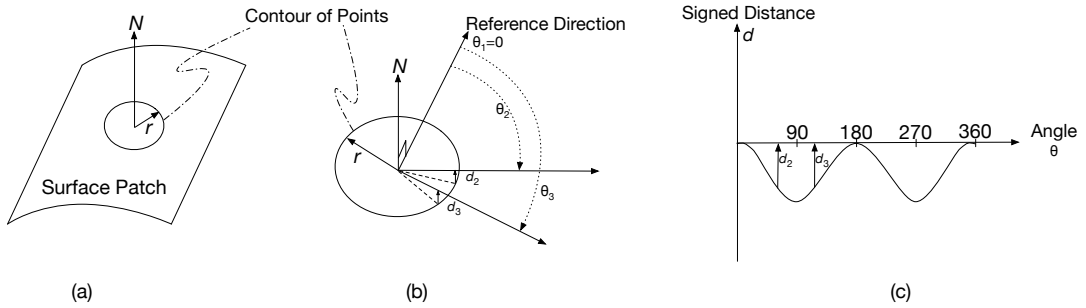


Figure 4.4: Point signatures: (a) contour of points at a fixed radius, (b) reference direction and two angles from this direction, (c) point signature as distance profile from translated fitted plane. Diagram based on example given in Chua & Jarvis (1997).

by the surface normal N at p, a reference vector, and their cross-product as shown in Figure 4.4(b). A new plane E' is defined by translating the fitted plane E at p in a direction parallel to the surface normal N, which is then used to define a planar curve using the projection distance of points from E' to E. Figure 4.4(c) illustrates the point signature of p, resulting from the signed distance between these two planes at different angles. During recognition, for a given scene, point signatures are computed at arbitrarily spaced seed points and each of these signatures is used to vote for models that contain points having similar signatures. Models are ordered according to the votes they receive and the most voted model is then verified. Due to the simple representation of the one-dimensional point signature, matching of point signatures is efficient and fast. In order to cater for noisy range data, a tolerance band is introduced along the signed distance direction. The system is evaluated both for single object matching and retrieval in a cluttered scene. In the latter case, the scene consisted of four different face masks piled on a terrain. The system is shown to perform well in this case-study.

Whereas both Stein & Medioni (1992) and Chua & Jarvis (1997) adopt a 1D representation that accumulates surface information along a 3D curve, a more descriptive 2D representation that accumulates information about a surface patch is proposed by Johnson & Hebert (1999). Specifically, they introduce *spin images* (SI) as object descriptors for 3D object recognition in point clouds. The spin image (Johnson, 1997) is a shape representation which describes local surface properties of an object and is constructed around oriented points on the object. When not available, each point's surface normal is computed by fitting a plane

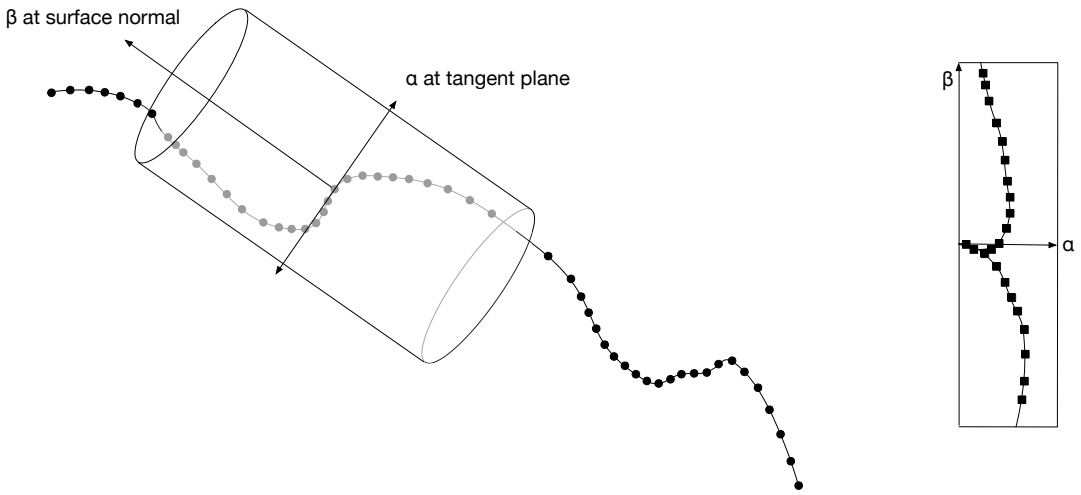


Figure 4.5: Spin-images are created for points on the surface of an object. Left hand side show a cross-section of a surface (for clarity) together with a cylinder oriented along the β and α directions. Right-hand side shows the spin image resulting from this cross-section of the surface. Additional pixels are added to the spin-image depending on the remaining points within the cylinder. More detailed examples showing models used for object recognition are illustrated in Johnson & Hebert (1999)

to a neighbourhood of points. For each oriented point, a SI consisting of a 2D accumulator buffer indexed by parameters α and β is created. The coordinates (α,β) are computed for each point within the support distance (user defined) and the bin indexed by (α,β) is incremented. Dark areas in the SI correspond to many projected points, effectively resulting in an image describing the point density distribution around each point. Figure 4.5 illustrates a simple example. Bin size is used to vary the geometric width of the bins and thus the resolution of the SI, with the ideal value providing a good balance between encoding global shape and averaging of point positions. Johnson & Hebert (1999) suggest that the best bin size is the one which exactly matches the mesh resolution computed over the point samples, under the assumption that the model is uniformly sampled. Whereas clutter and noise is not an issue when creating SI for training models, this becomes relevant in the context of object recognition from scenes containing clutter and occlusion. Since SI are created for all the points in the scene for comparison with those created for single objects, a scene SI may contain points from several objects. In the worst case, all the points in a scene may contribute to the SI of the scene resulting in poor recognition performance. In order to limit the effect of self occlusion and clutter during SI matching, another parameter is used namely the support angle which specifies the maximum angle between the

direction of the oriented point basis of a SI and the surface normal of points that are allowed to contribute to the SI. Moreover the support distance is limited in order to cover only a small distance from the oriented point basis and increase the probability that the points contributing to the SI are from the same object. Given a point cloud P, with parameters D_s and A_s representing the support distance and angle respectively, the subset P_s resulting from the following set comprehension contributes towards the SI of each oriented point p_s,

$$P_s = \{p : P | dst(p, p_s) < D_s \wedge acos(p_s, p) < A_s \bullet pos(p)\}.$$

A surface matching engine is used to establish correspondences between SI from trained models and SI from the scene using a loss function of the linear correlation coefficient as a measure of similarity. A modified ICP algorithm is then used to determine geometric consistency of the proposed mappings. The input point cloud is processed to remove isolated points and small patches. Moreover, a mesh computed over the point cloud is smoothed and re-sampled to change the scene data resolution to that of the trained models. Given these assumptions, recognition rates are measured and shown to produce good results in the presence of both occlusion and clutter.

Ruiz-Correa *et al.* (2001) extend SI to spherical spin images (SSI). Like SI, SSI are signatures associated with the vertices of a polygonal mesh of a given resolution that approximates the surface of an object and are represented as points onto a unit sphere. The set of SSI for an object is constructed using the linear correlation coefficient to define an equivalence relation on the set of SI. A comparative study between SI and SSI is carried out on 138 scenes using a library of 5 models. Each scene consisted of either 4 or 5 of these models piled up on a surface and in a similar fashion to Johnson & Hebert (1999), a uniform distribution at a resolution of 1mm is enforced on the acquired scenes. Results presented show that the SSI descriptors improve on SI both in terms of performance and accuracy.

Hetzel *et al.* (2001) explores the use of view-based histograms for 3D object recognition from range images. The descriptor makes use of three shape specific local features namely, pixel depth, surface normal and curvature. Curvature is computed as a shape index value (Koenderink & van Doorn, 1992). The three histograms are combined into a multi-dimensional histogram to model the probability distribution of different feature combinations and thus of certain shape

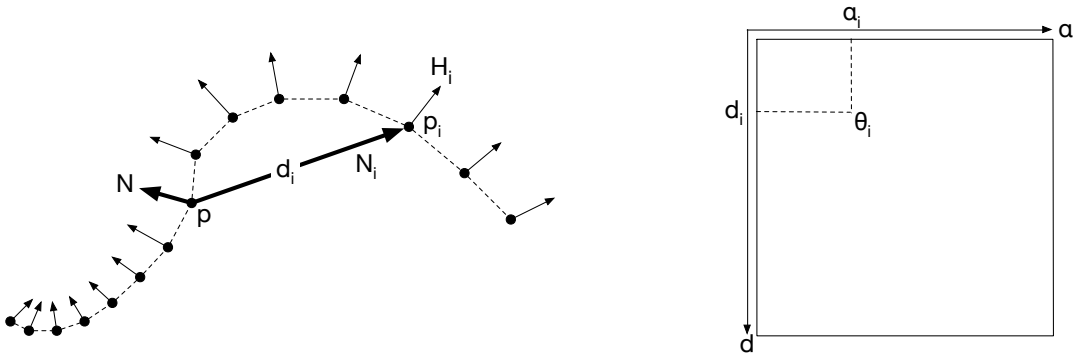


Figure 4.6: Left hand side shows samples from a 1D portion of a surface, whereas the right hand side illustrates a partial surface signature for the point p, which is only taking in consideration point p_i. θ_i represents the tuple consisting of the distance d_i between p and p_i and the angle α_i between the surface normals at p and p_i.

patches. Object recognition is performed using either a histogram matching χ^2-divergence test or a probabilistic recognition process which calculates the posterior probability of an object given the data using the Bayesian theorem (Shafer *et al.*, 1976). The system is evaluated using 30 synthetic free-form objects as training models from which 66 depth buffer images are produced by moving the virtual camera around the object at intervals of between 23° and 26°. The training set thus consisted of 1980 images. Testing is then carried on the same set of objects, this time scanned from 192 different viewpoints each for a total of 5760 images. Occlusion is simulated by randomly blocking some regions of the images (20% to 80%) and only collect feature vectors from the remaining regions. Different combinations of features are tested for recognition rates, showing that using normals and curvatures independently perform much better (around 80% recognition rate for both) than just using depth data (around 40% recognition rate). When all three features are combined a recognition rate of 93% is reported. At 20% occlusion, recognition rates fall to 89% and 87% when matching using probabilistic and χ^2 test respectively. The authors show that recognition rates only start deteriorating to less than 60% recognition rates when occlusion increases beyond 60%.

Another surface representation scheme, the 3D point's fingerprint, is proposed by Sun & Abidi (2001). The descriptor encodes the normal angle variations and the contour radius variations along different geodesic circles projected on the tangent plane of a point. A point selection process is carried out in order to retain only those fingerprint descriptors which have high radius contour variations. A

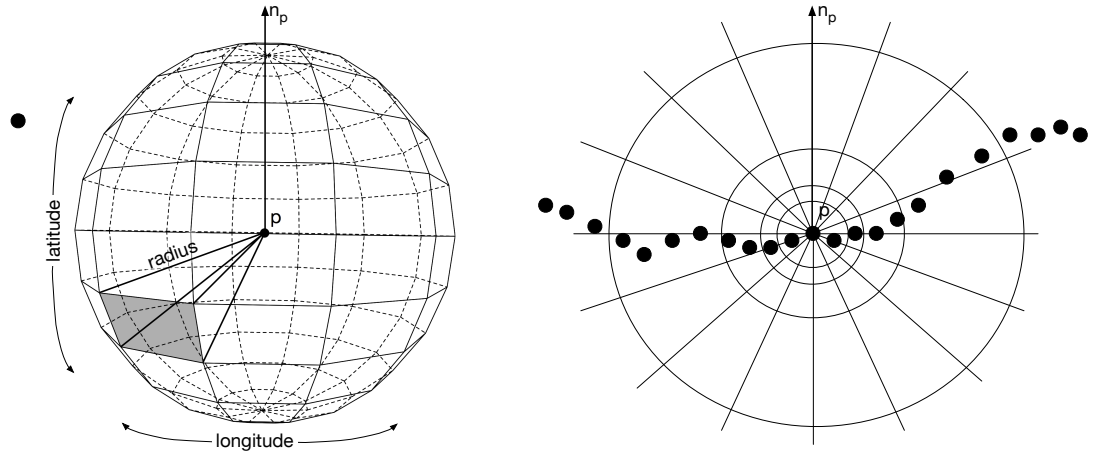


Figure 4.7: 3D shape descriptor histogram bins subdivided along the latitude, longitude and radial directions. To avoid clutter the bin is not shown subdivided logarithmically along the radial direction. $p \in P$ is positioned at the centre of the sphere and n_p indicates the surface normal of p.

cross correlation method is used to measure the similarity of two points and is shown to work for point cloud registration tasks.

Yamany & Farag (2002) describe another surface descriptor used initially for object registration and later for recognition. As opposed to SI (Johnson, 1997), which are based on point density around a point, the proposed representation scheme makes use of surface curvature information at certain points to produce images referred to as surface signatures. The signature computed at selected points encodes the surface curvature seen from each of these points using all the other points. The simplex angle (Delingette, 1999) is used to estimate the curvature value at points on a free-form surface. This curvature value is computed on all points and used to create the surface signature at positions (d,α) as illustrated in Figure 4.6. Object descriptors consist of a set of surface signatures computed at landmarks where curvature is above a user-specified threshold. Signature matching is carried out using a template matching scheme in which a measure defines how well a portion of an image matches a template. The technique is shown to perform well on a number of scenes consisting of a small number of models on a flat surface with some clutter and occlusion. When compared with SI, the authors show how more feature points from the scene are required to match SI rather than signature images, making the latter a more viable representation for object recognition in complex scenes.

Another technique for the recognition of objects in noisy and cluttered point

clouds (case study focuses on vehicles) is proposed by Frome *et al.* (2004). Image based regional point descriptors (Lowe, 1999; Belongie *et al.*, 2002; Mikolajczyk & Schmid, 2005) are extended by another dimension and used for 3D recognition and surface matching. The new descriptors, the 3D shape context and the harmonic shape context are used to capture the regional shape of the scene at a point p using the distribution of points in a support region surrounding p. The support region is discretised into bins, equally spaced along the latitude and longitude dimensions, as seen in Figure 4.7. The radial dimension is divided logarithmically such that bins closer to p are smaller thus making the descriptor more robust to distortions in shape as the distance from p increases. Bin $b(j, k, l)$ accumulates a weighted count $w(p_i)$ for each $p_i \in P$ whose spherical coordinates fall within the sphere region of p. The north pole of the sphere is set to the surface normal n_s, thus leaving one degree of freedom in the longitude dimension. For this reason the descriptor is synthesised as a set of histograms each rotated about the north pole whilst computing bin values. A second descriptor, the harmonic shape context, is synthesised from a 3D shape context descriptor. Bin values are used as samples to calculate a spherical harmonic transformation (Kazhdan *et al.*, 2003) for the shells at each interval along the radial dimension, resulting in a vector of coefficients which are rotationally invariant in the longitude direction and thus removing the remaining degree of freedom. Both descriptors, in addition to another based on SI (Johnson & Hebert, 1999) are evaluated using a training set consisting of several point clouds S representing vehicles from which m reference object descriptors (3 variants) per point cloud are computed. Object recognition is carried out by determining which of the reference objects is closest to the k representative descriptors computed on the input point cloud. Experiments show that all three methods perform roughly the same on point clouds with artificially added Guassian noise with a standard deviation of 5cm along the viewing direction. When the noise standard deviation is increased to 10cm 3D shape context descriptors perform better.

Anguelov *et al.* (2005) utilises a Markov random field (MRF) (Kindermann *et al.*, 1980) over points to label each point from a set of class labels, including the background. The MRF uses a set of pre-specified features of scan points, for instance SI (Johnson & Hebert, 1999) and height of the point, to provide evidence on their likely labels. The method assumes connectivity (links) exist between adjacent points in the point cloud which are either provided by the scanner or introduced by connecting neighbouring points in the scan. These links are used

to relate the labels of nearby points, thereby imposing a preference for spatial contiguity of the labels, with the strength of these links depending on distance. A graph-cut algorithm then uses the weights provided by the training phase, to label points of unseen scenes. Their method is validated on three case studies; terrain classification, segmentation of an articulated object, and point labelling of synthesised scenes consisting of combinations of vehicles, trees, houses and background. In the case of terrain classification, points are labelled as either ground, building, tree and shrubbery. Specific distances from the ground are used as features during the training phase, for instance ground points are easily identified with a z-coordinate value close to 0. Similarly, shrubbery, includes points at around 2m from the ground. Their approach correctly labels 93% of the points. In the case of articulated objects, three different wooden puppets are used. SI are included as local point descriptors to augment the feature set used at the training stage and achieve good results in labelling the different components (head, limbs, torso and background) of the puppet. Good results are also obtained with synthesised scenes. Whereas, achieving reliable labelling in the case studies presented, their method relies heavily on the existence of distinguishing local surface features on objects.

Chen & Bhanu (2007) propose the local surface patch (LSP), a descriptor based on the computation of shape indices from maximum and minimum principal curvatures at a number of feature points. The effectiveness of this representation is measured against SI (Johnson & Hebert, 1999) and SSI (Ruiz-Correa *et al.*, 2003). The proposed approach first carries out a feature point extraction process from range images in order to determine points situated in areas of large shape variation. Surface normals at each point on the surface are established by fitting a quadratic surface over a local window centred at a point. Given all surface normals, Gaussian and mean curvatures are determined in addition to the minimum and maximum principal curvatures. From these values a shape index quantitative measure, initially proposed by Koenderink & van Doorn (1992), of the shape at a point p is defined. The system is evaluated on range data containing single object scenes as well as four two-object scenes. Recognition performance is similar to both SI and SSI representations and show how using LSP improves on the time required to establish and verify correspondences by a factor of 3.79 over SSI and 4.31 over SI.

Novatnack & Nishino (2008) investigate the effect of size variations in captured range images on local geometric structures. This scale variability is used

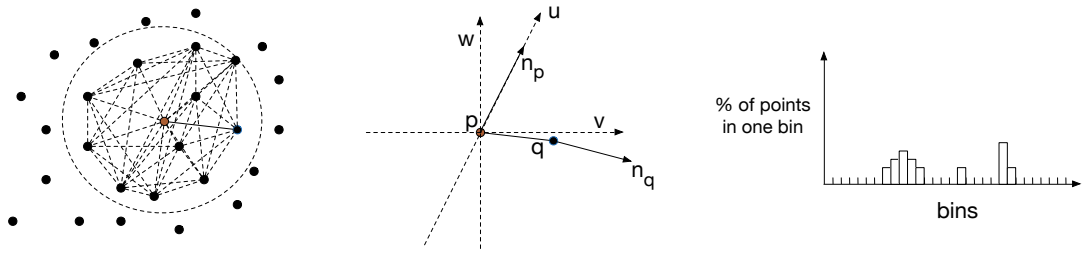


Figure 4.8: PFH descriptor histogram bins are computed for each point by computing an *idx* value from the summation of features f_0 to f_4.

as a source of discriminative information for surface matching. The *exponential map* descriptor is presented which encodes the components of the normals within a sphere centred at the surface point by deploying a 2D parametrisation of the local surface. It is used for aligning range images with the same global scale and also to fully automatically register multiple sets of range images with varying global scales corresponding to multiple objects.

Tombari *et al.* (2010) look into the problems arising when inaccurately choosing a local reference frame (LRF) when computing the signature or histogram at a surface point and show how this impacts the performance of 3D surface descriptors. Specifically they compare a novel descriptor, *SHOT*, with SI (Johnson, 1997), point signatures (Chua & Jarvis, 1997) and exponential maps (Novatnack & Nishino, 2008). The *SHOT* surface descriptor is intended to improve the generation of the LRF in terms of uniqueness and non-ambiguity. To this effect, instead of using standard Eigenvalue decomposition of the covariance matrix M resulting from the k nearest neighbours of a point p, a weighted linear combination is used in order to give distant points smaller weights. Moreover, in a similar fashion to the work of Bro *et al.* (2008), the signs of resulting eigenvectors of the LRF are oriented so that they are coherent with the majority of the vectors represented. SHOT encodes histograms with first-order differentials of the normals of the points within the neighbourhood. The descriptor is evaluated both on synthetic data and scenes captured using a triangulation-based scanner. In the latter case, eight models and 15 scenes are used with each scene consisting of two models. The authors show that SHOT outperforms all other descriptors in recognising the objects in the scene.

Rusu *et al.* (2010) present the viewport feature histogram (VFH) descriptor for recognition in point clouds that encodes both geometry and scanner viewpoint information. The technique presented is an extension of the fast point

feature histograms (FPFH) by Rusu *et al.* (2009), with the addition of viewpoint information. FPFH is an optimisation on the Point Feature Histogram (PFH) surface descriptor (Rusu, Blodow, Marton & Beetz, 2008; Rusu, Marton, Blodow & Beetz, 2008) which was initially applied for the alignment and registration of point clouds. The PFH describes the local geometry around a point p and is based on the combination of geometrical relations between the k-NN of p. PFH describes a feature space which associates points to specific surfaces types, for instance cylinder or plane. For each point p with surface normal n_p a set of k-NN within a sphere of radius r is first extracted. For each pair of points p, q in this subset, a point is first chosen as the source p_s (the other is set as the target p_t) such that the source is the point having the smaller angle between its associated normal and the line connecting the points. p_s and p_t with their respective surface normals n_s and n_t are then used to define an orthonormal basis where $u = n_s$, $v = \frac{(p_t - p_s) \times u}{\|p_t - p_s\|}$ and $w = u \times v$. Finally, a bin index value is computed as the summation of four feature calculations. Figure 4.8 illustrates a simple histogram example. A learning mechanism is used to associated points with surface classes including plane, sphere, cylinder cone, torus, edge and corner. The trained PFH descriptors are later used to differentiate between points lying on different surfaces and are able to recognise instances of primitive shapes in a scene consisting of mugs, glasses, bottles and books on a tabletop. Object recognition, in a tabletop object manipulation task carried out by a PR2 mobile robot, uses the FPFH descriptor and augments it to describe, using one VFH descriptor per viewpoint, the object. The viewpoint component is computed by collecting a histogram of the angles that the viewpoint direction makes with each normal on the object and factored into the FPFH components. The descriptor is validated on a number of scenes consisting of various kitchenware items including wine glasses, tumblers, drinking glasses, mugs, bowls and boxes. Prior to recognition, a segmentation process is carried out on the scene in order to remove the flat tabletop and cluster points together, effectively producing a set partition with object candidates. Objects are only slightly cluttered, with each separated by a minimum distance from each other. VFH is compared to SI (Johnson, 1997) with VFH achieving a better recognition rate of 98.1% compared to the 73.2% of SI. VFH also outperforms SI in a pose identification task.

4.1.2 Segment-Based Object Recognition

Fan *et al.* (1989) present a system which takes as input dense range data and automatically produces a symbolic description, an attributed graph, of the objects in the scene in terms of their visible surface patches. The segmentation and description of the surface is based on measured curvature and depth discontinuity properties. Object descriptors are synthesised as a set of such attributed graphs, typically between 4 and 6, each computed automatically from a range image of the object at a different viewpoint. A graph matcher is used to decompose the graph of the scene into sub-graphs corresponding to different objects matching the trained descriptors. The system was evaluated on three simple scenes each containing between 2 and 4 models.

A structural indexing technique, based on line segments, has been proposed by Stein & Medioni (1992) for matching points with surfaces using a 3D curve representation. The system is shown to be able to represent, match and recognise general objects. The object descriptor is based on a 3D curve which is extracted from objects using depth and orientation discontinuities and approximated by a set of consecutive line segments using curvature (between consecutive segments) and torsion (between consecutive bi-normals) information. The system is evaluated on 4 scenes, one borrowed from Fan *et al.* (1989), two consisting of three busts and a LiDAR terrain scene. For the latter, a tile is first copied from the data and then, during the recognition phase, is correctly aligned back with the original data.

Unnikrishnan & Hebert (2003) propose a method to robustly distinguish between planar structures and clutter in non-uniformly sampled point clouds of urban scenes. Points are first inserted into a uniform grid and each voxel given a score proportional to the local point density. Samples are then drawn by probabilistic region growing to cover the space, and M-estimation (Van De Geer & Van De Geer, 2000) is performed on each region to obtain a robust estimate of plane parameters best fitting the points in the region. Region plane parameters are then fused together using a generalisation of mean-shift based clustering. The detected planes are assumed to be walls, whereas the remaining points are associated with scene clutter such as trees, bushes and cars. As opposed to other techniques (e.g. Stamos & Allen (2000)), this approach specifically targets point clouds having low point density regions.

Lalonde *et al.* (2006) present a technique, aimed at improving safety in out-

door autonomous navigation systems, which classifies point clouds of terrain containing vegetation into either *scatter*, *linear* or $surface$. The *scatter* class is used to represent porous volumes such as grass and tree canopies, the *linear* class captures thin objects like wires and tree branches and the $surface$ class represents solid objects like ground surfaces and rocks. Similarly, Wang *et al.* (2008) outline a procedure for the recognition and structure analysis of tree canopies from LiDaR generated point clouds. First, the area of interest is chosen and the heights of each point are normalised with respect to a fixed planar ground. This is then segmented into small study cells using a 2D uniform grid over the ground plane. The main tree canopy layers and the height ranges of the layers are established according to a statistical analysis of the height distribution of the normalised raw points. In order to recognise individual trees, the 2D grid is extended for individual cells to sub canopy layers to include the tree crown region. Tree crowns are detected by projecting the normalised points into the sides of the local voxel space. Individual trees are then extracted by analysing the resulting 2D projections and performing a tree traversal process which groups the vertical neighbouring crown contours from layers at different height levels. Other techniques have focused on the recognition (and reconstruction) of tree features in point clouds. Specifically, Xu *et al.* (2007) device a more detailed approach, based on allometric theory (Niklas & Spatz, 2006), focusing on the recognition of the various elements of a tree in a dense point cloud representing the tree. Their technique is used to identify the crown, branches and leaves of trees in order to reconstruct quasi-identical meshes.

Road surface modelling from point clouds acquired using airborne LiDaR and laser-based mobile mapping system has also received considerable attention. The automatic recognition of traffic signs, curbstones and pavements, contributes to accurate and up-to-date road side information which can then be used for road planning and various location-based services. Jaakkola *et al.* (2008) describe a system intended to recognise a number of components of a road from point clouds acquired using vehicle-based laser scanning. The system is shown to identify parking lines, zebra crossings and curbstones. In order to detect parking lines and zebra crossings (painted markings), the acquisition process attaches intensity level values to points in the point cloud. A segmentation process first partitions the point cloud into two partitions, points belonging to painted markings according to their intensity levels and the rest. Points falling in regions with less than a specific density value are automatically removed on the assumption that

the acquisition process has sampled the road surface regularly and thus points falling in lower density areas consist mainly of buildings which are not used here. Several image processing filters are then applied on intensity and height raster images resulting from projecting the points on a plane parallel to the ground in order to extract the required structures. Beyond the accurate detection of road surface details, realistic 3D city modelling also necessitates the acquisition of building features. A number of purposely designed recognition algorithms address this problem by automatically extracting building features from point clouds acquired using ground-based laser scanners. Pu *et al.* (2006) present a pipeline for automatically extracting building facade features such as walls, windows and doors. The point cloud is first segmented (Vosselman *et al.*, 2004) and then segment properties are used to extract potential building features based on prior building facade knowledge. This knowledge includes constraints such as roofs are on top of walls, windows and doors are always on the walls, walls are vertical, etc. These constraints are encoded during recognition and each segment is checked to determine which kind of feature it is. Recognition of facade features is carried out in a specific order namely ground, wall, roof, window, door depending on the encoding of the constraints. For instance, extrusions and intrusions on the walls are labelled as windows or doors. The convex hulls of each segment is computed to determine the area of the segment, and used as another constraint. Recently, Nguatem *et al.* (2014) have described a system which specifically focuses on the identification of different types of windows (e.g. Gothic) and doors in a facade.

Schnabel *et al.* (2008) propose a system intended to detect architectural features, e.g. windows and columns, in an unstructured point cloud. They first decompose point data into primitive shapes from which a topology graph is created capturing neighbourhood relations between primitives. In a second stage, this topology graph is searched for characteristic sub-graphs corresponding to the sought user-defined elements. Segmentation is carried out using the algorithm presented by Schnabel *et al.* (2007) to recognise planes, spheres, cylinders, cones and tori. The output of the segmentation algorithm is a set partition, where each partition is associated with a shape primitive ϕ_i except for one, R, which groups together those points which do not fit any primitive. Following segmentation, all points are inserted in a regular grid with cell width t to accelerate the computation of the distance function. Modifying the value of t, results in the creation of different neighbourhood relations and in general increasing it will increase the

number of edges in the graph. Shape recognition is achieved via constrained sub-graph matching between a user-defined graph augmented with additional constraints representing the characteristics of the shape being searched, itself a topology graph, and the topology graph representing the point cloud. If searching for saddle-back roofs three additional constraints are added, namely i) the planes should exceed a certain size, ii) are similar, and iii) their intersection line is parallel to the ground. The method is applied on a number of case-studies including the detection of Gothic windows in a medieval chapel and columns in a choir scene sculpture.

Golovinskiy *et al.* (2009) investigates the design of a system for recognising objects such as cars, lamp posts and traffic lights, in point clouds of urban environments. The algorithm proposed takes a point cloud P representing a city and a set of training objects with their location labelled on the 2D plane of the city, and creates as output a set partition of P and labelling with every $p \in P$ associated to a partition and every partition mapped to a label representing the recognised object (possibly background). Their process first generates a list of locations for potential objects of interest in P, then predicts for each of these locations which of the nearby points are part of the object and which are background clutter. For each of these potential objects, a set of features describing the shape and spatial context of the object are determined and used to classify the object according to the previously labelled examples in the training set. The descriptor consists of a classifier built over the feature vectors extracted from the training set, which during recognition labels potential objects. Several classifiers are evaluated including k-NN, random forests (Liaw & Wiener, 2002), and support vector machines (Hearst *et al.*, 1998) available within the Weka toolkit (Witten *et al.*, 1999). An accurate segmentation of the point cloud is critical to the success of the method in that potential objects need to be void of clutter and background noise and ideally only consist of points sampled from one object. For this purpose, a min-cut algorithm (Stoer & Wagner, 1997) is used to extract the objects from the background. For each of the objects, several shape features are computed including number of points, estimated volume, average height, standard deviation in height and the standard deviation in the two principal horizontal directions. Moreover, a SI object descriptor centred at the predicted object location with a radius of 2m and central axis perpendicular to the ground is computed. Contextual spatial features are extracted such as distance to nearest street by incorporating digital imagery of the city used. The

system is able to recognise 65% of the 6698 objects in the city.

Zhao *et al.* (2010) utilise a robot vehicle system to generate point clouds from range images of urban environments with the intent of producing a semantic map of the objects present in an area. Their method first computes segmentation primitives from range images, using scan-line segments and edge points, then merges these segments considering both modelling costs and classification probabilities. The process tries to identify buildings, roads, trees, car, humans and bushes by attaching likelihood values to segments, for instance a person can be restricted within a cylinder, a car has a maximum width, length and depth. The method is generally able to recognise these objects with the assumption that all the objects are acquired from the same height along the path of a moving vehicle.

Mura *et al.* (2013) present a pipeline for the automatic recognition of rooms in an interior building under clutter and occlusions given a set of range images. For each image, an occlusion aware process is first used to extract vertical planar patches. These patches are then projected on the horizontal plane to get line segments from which cells are built at the intersections of the representative lines. Clustering of the cells is used to extract the separate rooms. The system is evaluated on real and synthetic data sets and is able to accurately recognise the rooms present in the scenes.

Gao & Yang (2013) propose a segmentation and identification pipeline intended to recognise buildings from ground-based LiDaR data in urban scenes. In addition to a point cloud with rich street-level details, the method requires the scanning/driving trajectory. Depth maps are extracted from a virtual camera which is placed looking in the direction of the scanner and perpendicular to the ground following the trajectory in the point cloud. From these depth maps a histogram is created with the horizontal axis corresponding to positions sampled at every 0.5m along the driving route and the vertical axis representing corresponding number of visible foreground pixels in the depth map along that scan line. On the assumption that there are significant gaps between buildings, the system is able to recognise individual buildings in the point cloud by segmenting the histogram along sustained peaks on the horizontal axis. The system is shown to work on point clouds representing both mass produced single-family houses and a typical down-town area of buildings varying in size and style. The overall recognition rate is stated at 86% of the buildings present in the datasets. Another building recognition algorithm was presented by Frueh *et al.* (2005) which how-

ever focuses more on the alignment of camera images with the extracted building facades.

Karpathy *et al.* (2013) describe an object discovery method in 3D scenes acquired using a triangulation-based sensor based on segment shape analysis. Rather than synthesising a descriptor for a point, this method looks into the creation of a descriptor for segments. A segmentation process, which uses the mesh produced during the acquisition step (Newcombe *et al.*, 2011), partitions the data using a region-growing approach using a local curvature-aware metric. The resulting segments are post-processed in order to accept for further processing those having at least 500 points and rejecting those that are more than one metre in size or less than 2cm thin. These value are chosen to fit the dataset acquired which consists to several counters with objects on them. The properties associated with the resulting segments include compactness, symmetry, smoothness, local convexity and recurrence across different scenes. Several options are considered to combine these properties into one score, with the RBF kernel support vector machine (Scholkopf *et al.*, 1997) capable of reliably distinguishing objects. Given a uniformly sampled scene, with specific parameters, the technique is able to correctly identify small objects on a counter. The main limitation is the requirement for a consistent segmentation process, which is generally difficult when the size of the scene acquired goes beyond a small area, for instance when scanning a room.

4.1.3 Summary

Both point-based and segment-based object descriptors have been used to address the recognition task from point cloud data. Some of the techniques presented require a range image of the scene, whereas others work directly on point clouds. Methods using point-based descriptors, tackle the problem of identification of objects within a collection located on a common surface, and take into account possible object occlusions and to a certain extent noise in the acquired samples. These methods search for similarities between points in the trained objects and points in the target point cloud, and any matches need to be verified using an ICP algorithm in order to corroborate the match. In general, these methods are more sensitive to sample noise, since descriptors are based on the neighbourhood of points. Uniform point density is in many cases necessary in order to tally the point descriptors computed on the training set to those computed on the point cloud. Table 4.1 lists the point-based methods presented. A number of seg-

mentation based 3D object identification methods have also been described. In particular, the topology graph matching technique presented by Schnabel *et al.* (2008), which builds a graph of segment primitives using a spatial neighbourhood function, can be used for general-purpose identification tasks. Sub-graph matching is used to identify regular structures in the graph, such as saddle-back roofs and stairs. A variety of methods have been designed to suit a specific identification task and preclude them from being used to address more general identification tasks. These are listed in Table 4.3.

Reference	**Comments**
Curvature Signs (Besl & Jain, 1985)	Point-based Classifier
Local Primitive (Faugeras & Hebert, 1986)	Point-based Classifier
Attributed Graph (Fan *et al.*, 1989)	Graph of segments
Structural Indexing (Stein & Medioni, 1992)	3D Curve from segments
Point Signatures (Chua & Jarvis, 1997)	3D Curve from points
Spin Images (Johnson, 1997)	Point-based 2D Histogram
Surface Signatures (Yamany & Farag, 2002)	Point-based 2D Histogram
Shape Context (Frome *et al.*, 2004)	Point-based 3D Histogram
Harmonic Shape Context (Frome *et al.*, 2004)	Point-based 3D Histogram
Local Surface Patch (Chen & Bhanu, 2007)	Point-based 2D Histogram
Exponential Maps (Novatnack & Nishino, 2008)	Point-based 2D Histogram
SHOT (Tombari *et al.*, 2010)	Point-based 2D Histogram
Point Feature Histograms (Rusu *et al.*, 2009)	Point-based 2D Histogram
Viewpoint Feature Histogram (Rusu *et al.*, 2010)	Point-based 2D Histogram

Table 4.1: Summary of object/structure recognition techniques using point-based descriptors.

Reference	**Comments**
Topology Graph (Schnabel *et al.*, 2008)	Graph-based Classifier
Segment Shape Analysis (Karpathy *et al.*, 2013)	Shape-based Classifier

Table 4.2: Summary of object/structure recognition techniques based on segmentation.

4.2 Indoor Scene Understanding

The virtual reconstruction of indoor environments has recently witnessed a surge in popularity within the computer graphics community, mainly triggered by substantial improvements in relatively cheap portable 3D scanners. These scanners

Reference	Comments
(Unnikrishnan & Hebert, 2003)	Clutter/Planes classifier
(Anguelov *et al.*, 2005)	Ground/Building/Tree/Shrubbery classifier
(Lalonde *et al.*, 2006)	Grass/Tree/Rock/Wires classifier
(Wang *et al.*, 2008)	Structure Analysis of tree canopies
(Xu *et al.*, 2007)	Detailed tree reconstruction
(Jaakkola *et al.*, 2008)	Road surface modelling
(Pu *et al.*, 2006)	Facade wall, windows and doors

Table 4.3: Summary of techniques used for specific object recognition tasks.

enable easy and quick acquisition of small rooms with a typical volume of $5m^3$. Both triangulation based scanners (e.g. in Nan *et al.* (2012)) and time-of-flight laser scanners (e.g. in Mura *et al.* (2013)) have been used to acquire indoor environments. Triangulation based scanners are cheaper and are therefore more widely available. Scene understanding of indoor scenes usually tends to be more problematic due to increased clutter, resulting in low quality point clouds (§2.3.3). Indoor scenes acquired using triangular based depth scanners usually suffer from increased sample noise.

Scene understanding techniques are usually designed to work within a specific context represented by a training set of scenes, for instance point clouds representing similar office rooms or auditoriums. Given this scene-specific information, correspondence can be viewed as evolving from a purely geometric similarity function between objects to one which may include some form of semantic or knowledge-driven function. For many indoor scene understanding methods (Nan *et al.*, 2012; Kim *et al.*, 2012), the utilisation of prior knowledge is used (e.g. geometric - this is an example of a chair or, spatial - a monitor is found on a desk and a chair is found on the floor, scene upward direction), where the main difficulty is in the modelling of this knowledge as a scene descriptor and in making use of it efficiently. The use of this information during the training process, however, greatly limits the scope of these techniques to scenes which are very similar to the ones used for training and precludes them from correctly identifying any of the trained objects if these are positioned, scaled or oriented differently.

Methods for indoor scene understanding are categorised in two, namely supervised and unsupervised. Supervised methods entail a training process resulting in a scene descriptor, which encodes objects in O and a variety of properties from the scene. This scene descriptor is then used to establish correspondence between

the elements of the set partition of point cloud P and objects in O. Unsupervised methods, do not utilise a training phase and instead rely on the presence of patterns, such as repetition and symmetry between segments, in order to clusters together similar segments. Due to considerable sensor noise, resulting in low quality point clouds, point-based object descriptors are not generally used. For all methods, a segmentation process is required to first produce a set partition of P. In the next sections, supervised methods are presented first (§4.2.1), followed by unsupervised methods (§4.2.2).

4.2.1 Supervised Methods

Rušu *et al.* (2008) describe an object identification method for household kitchen environments, where cupboards and drawers are represented as a cuboid with doors and handles, whereas tables and shelves are represented as horizontal planes. The input consists of a set of range images with camera parameters of a scene. A geometrical mapping module first uniformly re-samples the input point cloud, removing noise, and embellishes points with surface curvature, normals (oriented using camera parameters from range images) and a geometric description of the local point neighbourhood using feature histograms. A functional mapping model then extracts semantic information based on 3D geometry and a set of assumptions about the world, in this case a kitchen environment. The assumptions include: tables are planar horizontal surfaces located at hip height, cupboards and drawers are vertical surfaces with a certain area and having a handle, kitchen appliance with knobs. Other elements such as chairs are not identified. Region growing using smoothness constraints is used to generate the segments. In order to identify the interesting segments further assumptions are carried out, namely that there exists only one floor and one ceiling planes and walls have specific properties. Knobs and handles are identified by looking at small clusters of points at specified distance from the vertical planar segments. The cuboid structures are then classified into different classes based on a set of high-level features, for example $hasHandle$ and $hasKnobs$.

Several methods have been proposed for recognising interior walls in a point cloud (Hähnel *et al.*, 2003; Thrun *et al.*, 2004; Budroni & Böhm, 2009). More recently, Adan & Huber (2011) present a method to recognise and reconstruct interior wall surfaces in indoor scenes under occlusion and clutter. Their approach is different from previous methods in that they explicitly reason about occlusions

and are thus able to handle scenes with high levels of occlusion. A point cloud P, with the *up* direction defined, is first down-sampled (0.04 of the original size) using a voxel-based scheme, where each $p \in P$ is quantised into a voxel. A number of assumptions are taken, namely that walls are aligned with an axis of the voxel space, and that the surfaces to be modelled are planar. Given these assumptions and a new set of points representing the occupied voxels, the approximate planes of the walls, ceiling and floor are then detected using projections into 2D followed by the application of the Hough transform to produce a set of surface candidates. Occlusion labelling is then carried out on each voxel of each surface candidate and a 2D image is then computed. A scene descriptor is trained using a support vector machine (SVM) classifier (Hearst *et al.*, 1998) to distinguish between proper openings in a wall and those resulting from occlusion using a 14-component feature vector based mainly on the area, width and heights of an opening. The descriptor, is trained on a set of 370 examples containing both valid and invalid openings, and is evaluated on a point cloud representing a two storey building consisting of forty similar rooms with consistently good results.

Koppula *et al.* (2011) and Anand *et al.* (2012) present a supervised method that exploits relational information derived from the full-scene 3D point cloud for object labelling. Point clouds, acquired using triangulation based scanners, are over-segmented using a region-growing process with takes in consideration local surface normals and distance between points. Each of these segments is then labelled with a specific category following a training process which builds a model encoding properties of these segments including visual appearance (colour and intensity), depth and contextual information. The model also assumes that if nearby segments are similar in visual appearance, then they are more likely to belong to the same object. In addition to appearance, the model also encodes local shape, for example, a table is horizontal and a sofa is usually smoothly curved. The model also caters for geometrical context, whereby it exploits the repeated occurrence of specific geometric configurations, for instance, a monitor is always on top of a table, and chairs are near tables. The model, a MRF, is trained from a set of labelled training examples (2495 labels) and used to classify a set of office and home scenes. Labels include wall, floor, tabletop, table leg, laptop and book. In many cases individual objects are divided into multiple segments because of over-segmentation. The majority of the classes are correctly identified, with problems in cases such as the tabletop being confused with a shelf-rack. All point clouds used for training and evaluation, are acquired from

the same height which favours a consistent segmentation process for objects, and is of critical importance to the method. Scanning from a slightly more elevated position may considerably change the outcome of the segmentation algorithm which would reduce the effectiveness of the method.

A search-classify approach for scene understanding of cluttered indoor scenes is presented by Nan *et al.* (2012). A randomised decision forest (RDF) classifier (Breiman, 2001) is trained with various indoor objects (e.g. cabinets, chairs, tables) using a set of discriminative features. The features are tightly coupled with the upward orientation of the objects, with each object segmented into three horizontal slabs by analysing point distribution along the scene upward direction. Features include aspect ratios of the top, middle and bottom slabs, OBB height-size ratio, bottom-top and mid-top size ratios, and changes in centre of masses between the three slabs. The trained RDF is then used during a segment growing classification process to identify subsets of segments in the scene making up a trained object. A segmentation process is first carried out on the points to produce a set of segments via a region growing process based on normal smoothness (> 0.8) and distance threshold ($< 1cm$). An adjacency graph is computed over these segments, connecting those which are less that $15cm$ from each other. The search-classify procedure starts by selecting m random segment triplets and test them for classification likelihood using the RDF. Those with a high value are further processed by traversing the adjacency graph and iteratively adding nearby segments and recalculating classification likelihood. The region growing process stops if accumulation of any neighbouring patch results in a decrease of the current likelihood. Since it is possible to include a segment in two separate objects, a template fitting via deformation process is carried out at the end to remove these ambiguities. The method has been applied on a data set consisting of scanned indoor scenes, with very good identification results. The main problem with the approach is the limitation to upward orientation embedded in the object descriptors of the trained RDF, with classification and fitting assuming a global scene upward orientation. Objects which do not obey this assumption yield incorrect results. Moreover, objects such as shelving-racks, which do not fit within the three horizontal slabs classification scheme cannot be reliably identified.

Shao *et al.* (2012) present an interactive approach to semantic modelling of an indoor scene from a set of RGBD images. Segmentation of these images is first carried out automatically using a Conditional Random Field (CRF) classifier (Lafferty *et al.*, 2001), and if not satisfactory, the user interactively draws

strokes on the images to achieve better segmentation results. Segments are classified as either sofa, table, monitor, wall, chair, floor, bed, cabinet, ceiling or background. After segmentation, the depth data of each segment is used to retrieve a matching 3D model from a database. For this purpose a Random Regression Forest (RRF) classifier (Liaw & Wiener, 2002) is used, which is trained using rendered depth images of 3D models in the database annotated with model class labels, orientation angle and distance from the virtual camera. Results show that this is a good approach towards scene modelling, which however requires user interaction, and is a viable method for semi-automated indoor scene understanding.

Kim *et al.* (2012) describe a method which exploits object repetitions and variability in a typical indoor scene. The learning phase uses frequently occurring 3D models to capture different configurations per model (e.g. hinge angles) from a number of scans. The descriptor used represents these models as a graph of box, cylinder and radial structure primitives. For each object, the common primitives are used as the proxy representation with the rest representing variable parts. In the recognition phase, the method first extracts the dominant plane in the scene which is assumed to represent the ground. Planes parallel to the ground are tagged as tabletops if they are at specific heights from the ground, and assume that working surfaces have similar heights across rooms. Scene prior information, for instance chairs are located on the ground, monitors on the desk and desks repeat horizontally, is used to match scene segments with the trained models. A 10cm distance threshold is used during the region-growing segmentation process. The method is evaluated on a number of synthetic and real-world scenes, obtaining good results in both cases. The main strength of this method is that it captures typical objects variability modes, however is dependent on specific information about the scene.

A sliding windows approach to object detection from RGBD images is presented by Song & Xiao (2014). The method first trains an ensemble of linear Exemplar-SVM (Malisiewicz *et al.*, 2011) using feature vectors extracted from depth images of CAD models. An axis-aligned 3D sliding window is then shifted across the scene to determine which of the objects trained is present. The feature vector for each depth image is composed of information relating to point density, local 3D shape, normals and truncated signed distance function (TSDF) (Newcombe *et al.*, 2011) which encodes self-occlusion within the depth image. In order to reduce sample space, an assumption is taken on the upward direction of

the models and only rotations along the upward direction are considered when training the SVM. During testing, the method exhaustively classifies each possible bounding box in the 3D space of the scene using all Exemplar-SVMs, and outputs a detection score which is then followed by a non-maximum suppression process on all boxes. The method is evaluated on a dataset of RGBD images containing five common indoor objects: chair, toilet, bed, sofa, and table. The experiments carried out show that the method is able to correctly identify these objects in many cases. Some false positives occur when objects have similar shapes, since the sliding window approach may consider only parts of an object in an input image. Moreover, since the 3D sliding window is axis aligned, the method fails to detect objects which are inclined, or are not directly placed on the floor.

4.2.2 Unsupervised Methods

Mattausch *et al.* (2014) present a method to automatically segment indoor scenes by detecting repeating objects. An input point cloud is first partitioned into a collection of nearly-planar segments, which are then grouped together using a segment similarity measure based on shape descriptors and spatial configurations of neighbouring segments. Region growing starts by first ordering points in ascending measure of curvature $c{=}e1/(e1+e2+e3)$, where $e1$, $e2$ and $e3$ are the three Eigenvalues obtained from PCA over a set of nearest neighbour points. Seeds for region growing are selected from this ordered list, and continue expanding until either the normal of the neighbouring point varies or the neighbouring point is outside the definition of the current segment plane. This set of segments is then partitioned into two categories, namely horizontal and vertical, by taking an assumption that the floor of the rooms is always the XY-plane. A feature descriptor is used to discriminate between the segments, and encodes segment area, ratio of segment width to length, ratios of areas (convex hull/segment area), height from the ground of centroid of segment, segment normal and a non-planarity value $d/(w+l+d)$. In addition to segment similarity, a spatial consistency model is also used which measures the similarity of spatial configurations (e.g. chair back and seat segments). In order to avoid some ambiguities, segments with areas greater than $1.2m^2$ (typically a tabletop), do not consider relations to nearby segments. Moreover, only those segments which are within a $20cm$ limit are considered as segments that could potentially be part of the same object. The

method is evaluated on data acquired using a laser range scanner which provided high quality point clouds of office scenes and is not tested on lower quality triangulation-based hand-held scanners. Results show that the method is able to consistently detect repetitive patterns such as shelves, tables and chairs. The method assumes that all objects of the same class have a consistent up-direction. Moreover, the segment representation used is not expressive enough to represent small objects with many planar regions like desk lamps.

4.3 Indoor Scene Understanding Feature Comparison

Table 4.4 shows a feature comparison for the indoor scene understanding algorithms described in this chapter. The comparison is based on a number of common properties associated with scene understanding methods.

Approach This property denotes whether the method is based on a supervised (which requires a training phase) or unsupervised (which does not use a training phase) process. Additionally, a method might require user input, for instance to guide the segmentation process.

Labelling The ability of the method to label the objects present in the scene, as opposed to just grouping together related segments.

Input Quality The quality (§2.3.3) of the point cloud required by the proposed method. Methods evaluated on point clouds acquired using commodity hand-held scanners are listed as *low*, whereas others evaluated on point clouds acquired using TOF laser scanners which produce less noise are listed as *high*.

Input Format The required input format for the scene understanding algorithm: *range images* or *point cloud*. Some methods rely on the availability of a range map to establish scene-specific parameters from camera information.

Context The reliance of the method on *scene specific parameters*, for instance user specified and consistent up direction amongst objects and specific distances or configurations between objects. If this scene information is essential, the method is referred to as *sensitive*, and *free* if not.

Object Pose Inv. Can an object be identified in a scene, if this is not consistent with the pose used during the training phase. Set to *true* if tilted or non-uniform scaled objects are identified, otherwise *false*.

Size Range What range of object sizes can be identified with respect to the area scanned: *coarse* (e.g. walls), *large* (e.g. chairs), *medium* (e.g. desk lamps) and *small* objects (e.g. computer mouse).

4.4 Discussion

Indoor scene understanding from point clouds has seen a surge in techniques, mostly using a segment-based supervised approach, which address the problem of identifying the main components of a scene. In order to do without the generally expensive training process, Mattausch *et al.* (2014) propose an unsupervised approach which searches for segment patterns in scenes using a variety of segment properties without the need of a training phase. However, whereas certain objects can be represented via a regular pattern, and therefore amenable to an unsupervised approach, others are more complex to describe. Amongst the many research gaps in the field, two are highlighted in the limitations of the methods described above. These are the absence of a method which takes advantage and combines supervised and unsupervised approaches and the inability of techniques, when using a supervised approach, to identify objects which are not necessarily in an upright pose or a specific distance from a user-specified floor. For instance, a scene consisting of a flight of stairs with objects placed on some of them cannot be interpreted correctly. Similarly, shelving units in a room, which usually vary in number of shelves and size, cannot be robustly identified using a supervised method. In these cases, unsupervised methods have been used to search for these regular patterns which cluster together segments with similar properties. An alternative approach which searches for specific segment patterns in the scene, and also for previously trained object descriptors in a context-free setting is missing.

Local point-based object descriptors(§4.1.1), for example SI, in an indoor scene understanding context have been shown to lead to poor performance with the size of the descriptor playing a critical role. In general, a relatively small sized descriptor makes the algorithm more robust under clutter and occlusion, but at the same time makes it harder to discriminate between locally similar shapes. Segment-based approaches have shown promise in many of the methods

Method	**Approach**	**Labelling**	**Inp. Quality**	**Inp. Format**	**Context**	**Obj. Pose Inv.**	**Size Range**
Rušu *et al.* (2008)	Supervised	✓	High	Range Maps	Sensitive	×	Coarse
Koppula *et al.* (2011)	Supervised	✓	Low	Range Maps	Sensitive	×	Medium
Adan & Huber (2011)	Supervised	✓	Low	Range Maps	Sensitive	×	Medium
Nan *et al.* (2012)	Supervised	✓	Low	Point Cloud	Sensitive	×	Medium
Shao *et al.* (2012)	Supervised/Interactive	✓	Low	Range Maps	Free	✓	Medium
Anand *et al.* (2012)	Supervised	✓	Low	Range Maps	Sensitive	×	Medium
Karpathy *et al.* (2013)	Supervised	✓	High	Point Cloud	Sensitive	×	High
Song & Xiao (2014)	Supervised	✓	High	Range Maps	Sensitive	×	Large
Kim *et al.* (2012)	Supervised	✓	Low	Point cloud	Sensitive	×	Medium
Mattausch *et al.* (2014)	Unsupervised	×	High	Point cloud	Sensitive	✓	Large

Table 4.4: Feature comparison of indoor scene understanding methods.

proposed, since these tend to mitigate the problems associated with noise and clutter prevalent in indoor scenes acquired using low quality hand-held scanners by grouping together spatially close points with similar properties. Nan *et al.* (2012) and Kim *et al.* (2012) both propose supervised methods which learn segmentation based object descriptors. During the training phase, both methods use prior information about object locations in the scene and upright direction and are therefore sensitive to object post changes in the target scenes. Similar restrictions are found in the method presented by Song & Xiao (2014), which trains Exemplar-SVMs using objects of a specific size and orientation. Shao *et al.* (2012) proposes a context-free setting, however, their method requires the user to provide hints to the segmentation process in order to extract object segments which are later matched against a repository of models.

Some of the techniques presented, only work when the input is in the form of a set of range images (Zhao *et al.*, 2010; Shao *et al.*, 2012) since these rely on camera parameters during the identification process. For point clouds acquired using simultaneous localisation and mapping (SLAM) (Newcombe *et al.*, 2011) techniques, which are common for indoor environments, range images are usually not available. Appearance (colour) information may be available and exploited (Koppula *et al.*, 2011; Anand *et al.*, 2012) in range images to improve identification results, but this information is not available in raw point cloud data.

A novel scene understanding framework is presented in Chapter 7, CoFFrS, which seeks to address these limitations, namely, dependence of scene specific context during training, restrictions on the pose of trained objects in a target scene, and the inability to interpret a scene by searching both for regular patterns and previously trained objects. Additionally, the framework should allow for easy integration of future extensions, for instance, an extension which exploits scene specific parameters when these are available during the interpretation of a scene, but not in the training phase. Arguably, the integration of prior information about a scene into a trained descriptor can lead to improved classification results from low quality point clouds. For instance, if there's a priori knowledge that chair seats are always at a fixed distance from the floor and at a particular orientation, then if a method can establish that a segment exists satisfying these properties, it can immediately infer the presence of a chair. However, in scenes where this is not the case, dependence on prior information can lead to inaccurate results. CoFFrS, which is based on PaRSe, the segmentation process presented in Chapter 5, is an attempt at designing a scene understanding framework which

does not depend on specific scene parameters, but only on the set of object classes that can be found in the scene.

4.5 Summary

This chapter has provided a literature review for 3D object recognition and indoor scene understanding methods from point clouds. Tables 4.1 and 4.2 list 3D object recognition methods using point-based and segment-based descriptors respectively. A variety of segment-based indoor scene understanding methods are also described in this chapter, with Table 4.4 providing a feature comparison of these methods. In all cases, scene specific parameters are utilised in the training and recognition phases, which diminish their effectiveness on scenes which include objects and structures in different poses to the trained descriptors. The following chapter introduces a novel raw point cloud segmentation method, PaRSe, which produces segment primitives used by CoFFrS, the scene understanding framework presented in Chapter 7.

CHAPTER 5

Point Cloud Structure Graphs

Segmentation plays a critical role in point cloud processing pipelines by contributing various levels of abstractions over raw data. These levels of abstraction, for instance a set partition of point cloud P mapping sets of points to geometric primitives, provide structural and shape information about the sampled scene. The segmentation techniques discussed in Chapter 3 have either employed a shape fitting or a region-growing approach. In the former, either the 3D Hough transform (Borrmann *et al.*, 2011) or RanSaC (§2.7) are used, whereas region-growing algorithms expand seed points over neighbours that comply with specified properties. This chapter introduces a novel general-purpose segmentation method for raw point clouds which combines a region-growing process with shape fitting using RanSaC. In order to increase the applicability of the segmentation process, and given the lack of context in which it is applied, only the plane primitive is used for fitting the data. When required, more complex shapes, e.g. cylinders and boxes are composed from the extracted planar segment primitives. Whereas it is possible to partition a point cloud into a collection of planar segments by directly applying shape fitting or by expanding seed points into regions of points with similar surface normals, the set partitions produced still do not exploit the benefits of both approaches. Moreover, in both cases, segmentation randomness

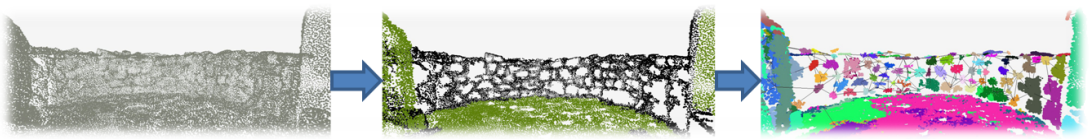


Figure 5.1: Automatic point cloud segmentation pipeline - Raw data is first segmented into smaller patches using a region growing process, then geometric planes (coloured patches) are mapped onto these segments using the RanSaC paradigm.

resulting from both selection of seeds and shape supports is still considerably high. Plane fitting using RanSaC is preferred over the 3D Hough transform method in this work as it has been shown to provide results in shorter time and of higher quality (Tarsha-Kurdi *et al.*, 2007). Figure 5.1 illustrates an example where a region growing process is first used to visualise the contour of stones making up an apse of a pre-historic temple and a RanSaC plane fitting process is then used to identify individual stones. The direct application of a RanSaC plane fitting process over all points in P, even if this is constrained using locality information as in Schnabel *et al.* (2007), does not produce a set partition enumerating the individual stones. Similarly, traditional region-growing using surface curvature properties cannot produce a segment representing the contour of the stones in the wall. The novel segmentation method presented in this chapter, PaRSe, addresses the following design goals:

- Can be applied to *generic* point clouds acquired from a variety of environments.
- functions with *minimal* information, namely position. Range images and camera parameters are not required.
- *Efficient* both in terms of memory and time complexities with data access patterns favouring parallelization.
- When applied to the same point cloud, using similar parameters, the segmentation algorithm should produce highly *repeatable* set partitions, both in terms of the number of segments and the assignment of points to segments.
- The elements of the set partitions are mapped to an *abstract* data type which can be used to easily carry out post-processing tasks.
- Segmentation should be *robust* to noise and occlusion.

Many point cloud processing tasks, for instance the removal of outliers, work on the assumption that a segmentation process is able to provide a meaningful level of abstraction. Similarly, many object recognition and scene understanding algorithms build upon the computed set partitions (§4.1.2 and §4.2). Segmentation processes have generally been tailored to suit specific scenarios and have therefore only been evaluated within one context, for instance, segmentation

of point clouds representing trees (Ning *et al.*, 2009), buildings (Dorninger & Nothegger, 2007) and industrial objects (Robbani & Vosselman, 2006). All these make a number of assumptions on the input point cloud.

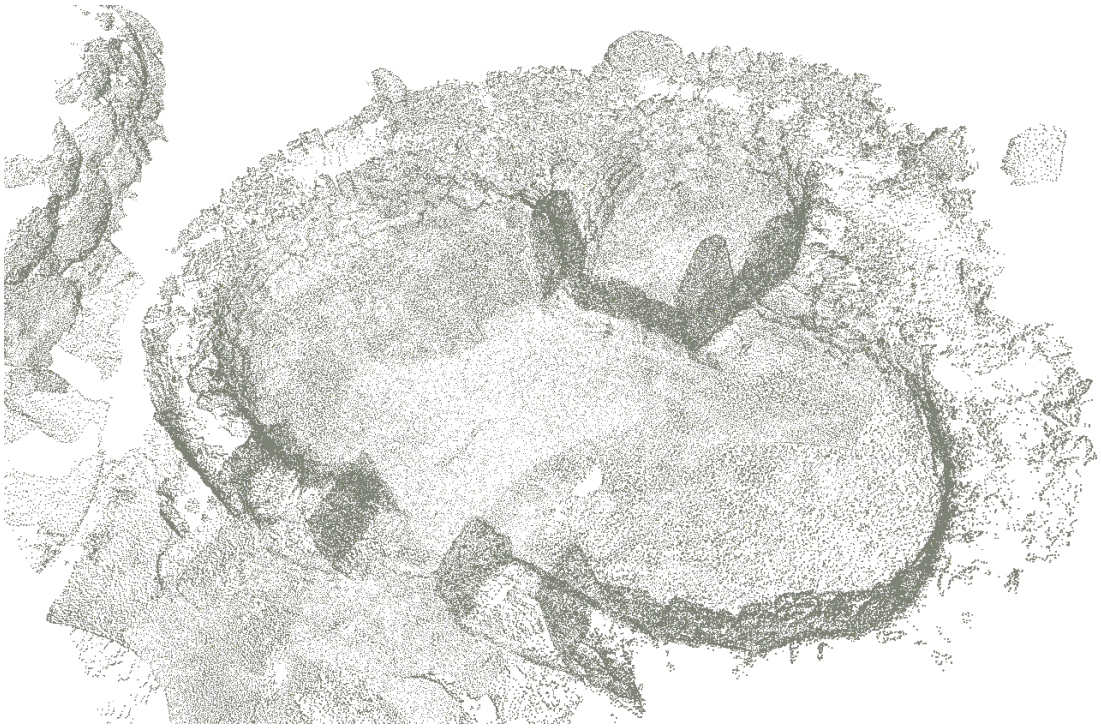


Figure 5.2: Point cloud of a section of the Mnajdra pre-historic temple (600K points).

As shall be shown, PaRSe, has been used to tackle tasks in a variety of contexts, ranging from point clouds representing simple geometric shapes to large airborne LiDaR data sets. While applicable to any point cloud, this work uses as a primary example a cultural heritage (CH) scene. Segmentation is particularly challenging in the CH context due to the generally more complex geometrical and surface properties (e.g. weathered and eroded stone) for certain CH sites. Segmentation of point clouds acquired from CH sites has not been given much attention in previous literature, notwithstanding the fact that in recent years many CH institutions have been engaged in the exercise of creating 3D virtual reproductions of sites for which they are responsible. Large architectural heritage sites are continuously being scanned and documented (for example in the work described by Ruther (2010a)) for the purpose of academic study, hypothesis evaluation, better preservation and CH dissemination to the general public. This scenario has contributed to an increase in importance for algorithms which are capable of analysing and processing point clouds efficiently. As pointed out

by Cignoni & Scopigno (2008), a major challenge is now how to manage the complexity of scanned data. In many of these cases, a segmentation process is required in order to partition the point cloud into a number of meaningful parts, which can be more easily managed. This partitioning of the point cloud effectively provides for a level of abstraction over raw position data which allows for easier and more efficient point cloud manipulation. Figure 5.2 illustrates the rendering of a point cloud acquired from the smallest of three temples in the Mnajdra pre-historic site. As pointed out by Cignoni & Scopigno (2008), the acquisition process is followed by substantial data processing, usually requiring user intervention, long processing times and above all tedious work. Ruther (2010a) describes how post-processing tasks usually take much more time than the actual acquisition process on site. This time can be decreased if the point cloud generated from the scanning process is partitioned into smaller meaningful subsets of points representing distinct geometries (e.g. Figure 5.1). This ability to automatically distinguish between different elements in the scene would benefit the CH professional working with the acquired point cloud. For example, tessellation problems common with complex sites such as pre-historic temples consisting mostly of weathered and eroded stone which usually require decimation, can be approached compositionally by tessellating segments individually according to requirements. In order to facilitate the dissemination of a virtual reconstruction of a CH site over the internet, a CH institution might want to down-sample the floor of the site but not the walls. The selection of parts of a scanned site, for example a specific wall or the floor, would usually require users to learn how a specific 3D modelling software is used. This work proposes an efficient and semantically meaningful point cloud segmentation pipeline which facilitates these tasks, and which only assumes the availability of position information within the data. The segmentation pipeline presented, enables the use of simple point cloud queries, where a processed point cloud can be used to efficiently query for and extract specific parts.

5.1 PaRSe - Method Overview

Algorithm 6 illustrates the high-level steps making up PaRSe. Key to the segmentation method presented here is the observation that objects, or collections of objects in a scene, typically consist of a number of surfaces connected via zero

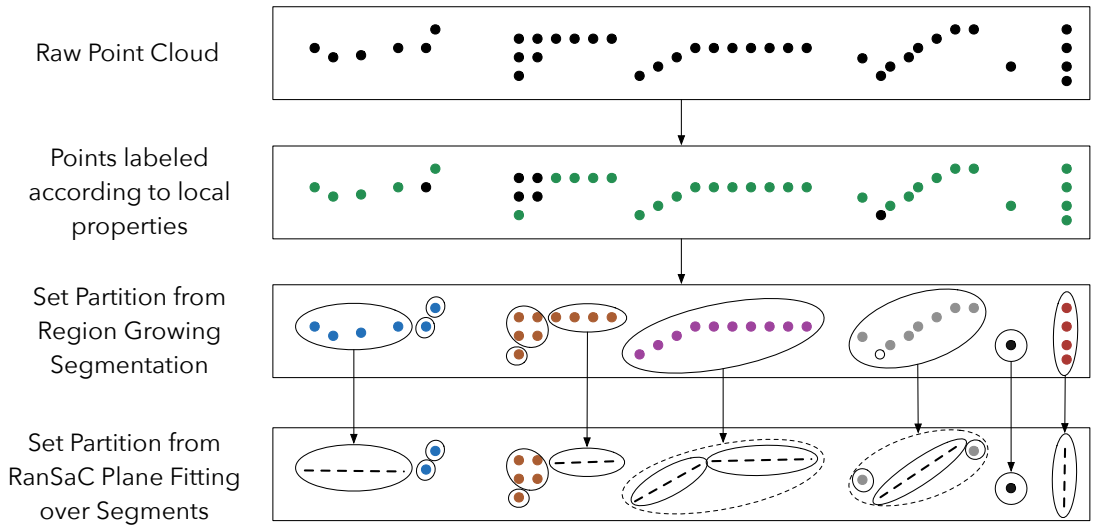


Figure 5.3: Levels of abstraction over a point cloud P. At the lowest level (top row) is the raw unstructured point cloud. All points are first labelled according to their local point neighbourhood properties. A region growing algorithm then produces a set partition as the second layer of abstraction. The elements of this set are then segmented again using a RanSaC based plane fitting process, with each resulting segment further subdivided if it consists of spatially disjoint point clusters (e.g. fourth column).

or more edges. For instance, when sampling a box object, each sample point on the box surface is a member of either one of two sets, namely one containing points that are sampled from an edge or a corner of the box, and the other containing those points which are not. This binary categorisation (Algorithm 6, line 2) is carried out via a local surface curvature computation for each point as described in §5.1.1. Certain objects, for instance a smooth spherical object, on the assumption that enough samples are acquired and noise is minimal, can result in an empty set of edge and corner points since every sample would belong to one set. A region growing process (Algorithm 6, line 3) then uses this labelling of points in order to partition the input point cloud into regions of the same type as described in §5.2. An additional level of abstraction is computed over the resulting regions, by computing for each region segment, another set partition which maps points in these segments to zero or more planar geometric primitives (Algorithm 6, line 4). Finally, these planar segments are further partitioned into disjoint point islands. Figure 5.3 illustrates the bottom-up pipeline described above on a simple example. Points in P are first labelled as either *surface* (green points) or *edge* (black points). Using this information, a region growing algorithm partitions P into a set of segments which are then further partitioned into

planar segments. Rather than trying to fit different geometric primitives to the segments, as in the work by Schnabel *et al.* (2008), a more generic strategy is adopted by only fitting plane primitives and subsequently constructing higher-order primitives from compositions of planes. Clearly, if a scene consists of a number of spheres, then first trying to fit planes and then searching for spheres is more expensive than directly using RanSaC to fit spheres. On the other hand, if a scene does not have any points which could fit a sphere, then trying to fit a sphere (and possibly other primitives) within the data is more expensive than just fitting plane primitives. To facilitate further processing which may be carried out on P, the elements of the final set partition are organised as a graph, referred to as the *structure graph*, which encodes connectivity information between the segments making up P. Figure 5.4 illustrates an example where the shape of a stairs and a chair are encoded as graphs. Given a structure graph for the scenes (chairs on stairs) shown, this is used to first identify the stairs and then the chairs by using transition trees (see §2.2.2) and measuring the compatibility between the graphs (see §2.2.3).

Algorithm 6 PaRSe three phase segmentation pipeline

1: **Input:** Point cloud P, segmentation parameters α.
2: $PointLabelling(P,\alpha)$ ▷ Input is split between *edge* and *surface* types
3: $Regions = RegionGrowing(P,\alpha)$ ▷ Grow regions using point type
4: $Segments = PlaneFitting(Regions,\alpha)$ ▷ Apply plane fitting to regions
5: $Segments_D = RegionGrowing(Segments,\alpha)$ ▷ Cluster disjoint point groups

5.1.1 Point Types

Both normal and curvature of a point on a surface can be estimated by considering a local neighbourhood of points. For this purpose, a k-NN query (§2.5.1) is used to determine the k_{max} nearest points for each $p \in P$. A maximum distance value, r, is used to bound the query to the local surface neighbourhood, just in case the point sample density within a particular area is very low. In addition to k_{max}, a value k_{min} is also set, which determines the minimum number of neighbours required to proceed with the computation. For a given point p, if the number of neighbours at a distance less than r is between k_{min} and k_{max}, PCA (§2.5.2) is used to determine whether p is most likely to be located on a surface or an edge by computing the maximum curvature of p. In a similar fashion to Hoppe *et al.* (1992), an OBB (§2.5.3) of the k_{max} neighbouring points of p is computed.

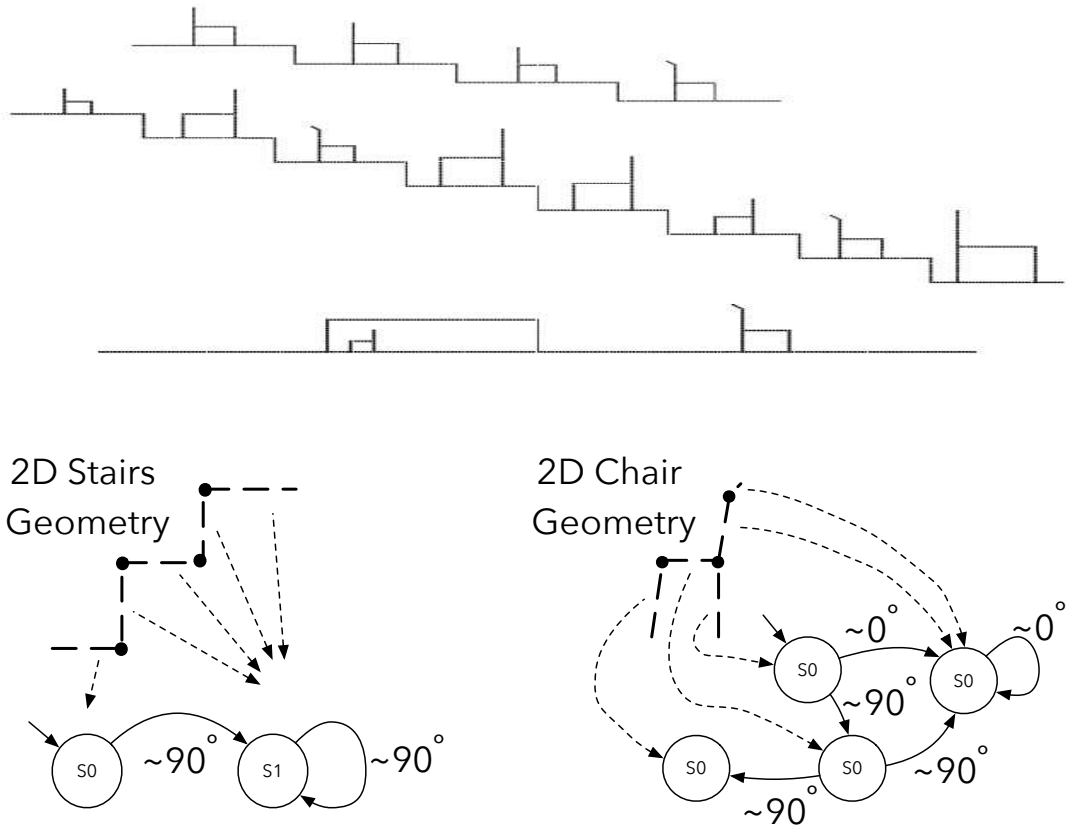


Figure 5.4: The point cloud above, representing 2D scenes of stairs and chairs, can be transformed into a graph describing connectivity between line segments. The two graphs, describing connectivity patterns for stairs and chairs can be used to map the line segments to object instances. The states of the graphs represent line segments, whereas the relation describes the approximate angle between line segments.

The ratio of the eigenvalues of this orthogonal basis is used to determine the type of each point. The number of eigenvalues returned by PCA depends on the dimensionality of the input data, which in this case is three and each represents the variance along the three eigenvectors describing the orthonormal basis of the computed OBB. The first eigenvalue $e1$ represents the largest variance, whereas the second $e2$, and third $e3$ represent the second and third smallest. Eigenvalues are further discussed in §2.5.3 and in more depth by Pauly *et al.* (2002). If the third eigenvalue is much smaller than the second eigenvalue, i.e. there is minimal variance along the third eigenvector, then the point is labelled as *surface*. The point is otherwise labelled as *edge*. A parameter, α, is used to determine the extent of the difference between these two eigenvalues. For example, if set to 12, then if the smallest eigenvalue multiplied by 12 is still smaller than the second eigenvalue, the point is tagged as *surface*. The neighbourhood function $\phi(p, r, k_{max})$ takes as parameters the point, radius and the number of neighbours required and returns a set of points closest to p and within distance r. For the case-studies in this chapter k_{max} was set to values ranging from 18 to 48 (Table 5.1). The following set comprehensions are used to partition P in three sets:

$$
\begin{aligned}
P_u &= \{p : P| \ |N_p| < k_{min} \bullet unspecified(p)\}, \\
P_s &= \{p : P| \ |N_p| \geq k_{min} \wedge (e3 * \alpha) < e2 \bullet surface(p)\}, \\
P_e &= \{p : P| \ |N_p| \geq k_{min} \wedge (e3 * \alpha) \geq e2 \bullet edge(p)\},
\end{aligned}
$$

where N_p is the set of neighbour points returned by $\phi(p, r, k_{max})$. Following this labelling, the input point cloud is thus partitioned in three sub-sets $P = P_u \cup P_s \cup P_e$. The resultant set partition depends on the different parameters used. Decreasing the value of r, would typically increase the size of P_u as this would generally decrease the size of N_p. The same effect can be had by increasing k_{min} which is set to three in all examples. Clearly, r should be set to a value which takes in consideration the coordinate space in which point clouds are defined. Alternatively, point clouds can be scaled such that a fixed r value can be used. These values are currently set manually by the user. Variations in α effect the distribution of points between P_s and P_e. At the extremes, this could lead to either P_s or P_e being empty on the assumption that $e3$ is not 0, i.e. all neighbours are perfectly sampled from a perfect plane. In general, decreasing α results in more points satisfying the set comprehension for P_s, whereas increasing it results in an increase in the number of points satisfying the set comprehension

for P_e. The following section describes how this parameter is used to straddle between over and under-segmentation. Figure 5.5 illustrates an example showing the process of how eigenvalues are used to determine whether a point lies on an edge or not. In order to improve clarity, the computation is carried out on a 2D surface and PCA is carried out on an oriented rectangle bounding p and its two neighbours, rather than an OBB volume as in the case of 3D. The second and third columns illustrate the effect of reducing sampling density on the computation of the set partition. Figure 5.6 illustrates the same example surface, with different columns representing increasing values of k_{max}. Whereas with k_{max} set to 3, small variations in the shape of the surface are captured, as k_{max} increases these small variations are lost, which might be a good thing in the context of a much larger point cloud.

The distinction between points in P_e and P_s is important. In general, points in P_s are surrounded along the surface of an object by points in P_e. For instance, Figure 5.7 shows edge points extracted from a simple scene where the different box faces are surrounded by points in P_e. Figure 5.8 illustrates the edge points computed on a more complex example. Figure 5.9 illustrates a number of point clouds, after points are labelled as either *edge* (black), *surface* (green) or *unspecified* (grey). In the first row, edge point fall mostly on plant leaves and boundaries of chairs. The second row shows how this type assignment of points contributes towards outlining the contour of the individual stones making up the apse wall of a pre-historic temple. The third row shows how edge points delineate the buildings and agricultural field boundaries in the LiDaR acquired point cloud.

Acceleration structures are used to speed up k-NN queries. Both a sparse grid and a kd-tree data structures have been used for this purpose, with the kd-tree approach generally providing better overall performance both for computing nearest neighbours and rendering purposes. This work uses the FLANN (Fast Library for Approximate Nearest Neighbour) libraries developed by Muja & Lowe (2009b) to carry out k-NN computations.

5.1.2 Segment Types

The obelisk point cloud shown in Figure 5.10, consists of a number of *surface segments* (green) connected together through one continuous *edge segment* (black). An edge segment is made up of a collection of points $p \in P_e$, whereas a surface segment consists of a collection of points $p \in P_s$. At this stage the set P_u of

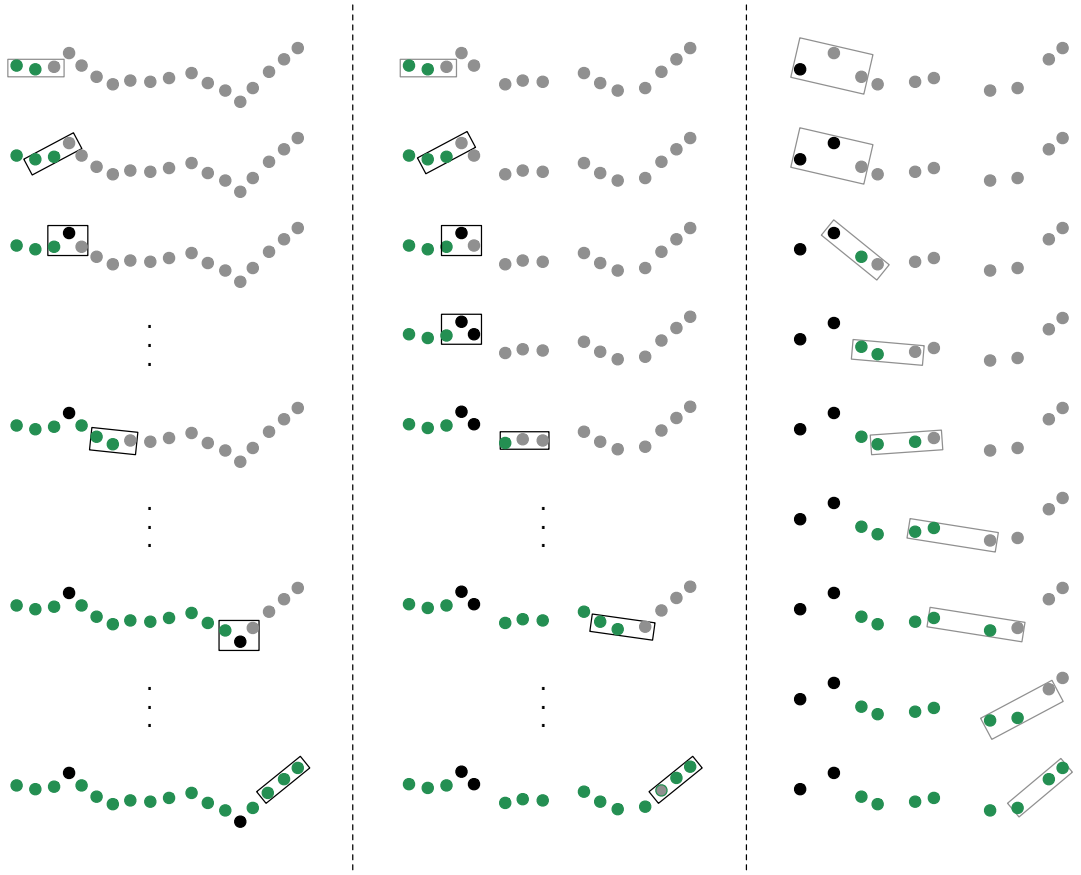


Figure 5.5: A 2D illustration of how points are tagged given k_{max}=3 and α=2 and assuming a large value for r. In practice, in order to extract a perfectly smooth surface, α is set to a very high value since the third eigenvalue will be very close to zero. This will also result in a higher number of points with type *edge*. The second and third rows, illustrate the same surface sampled using fewer points, whilst retaining the position of the samples from the first row. This change in density results in different labels for some of the points. Grey, green and black indicate unspecified, surface and edge point types respectively.

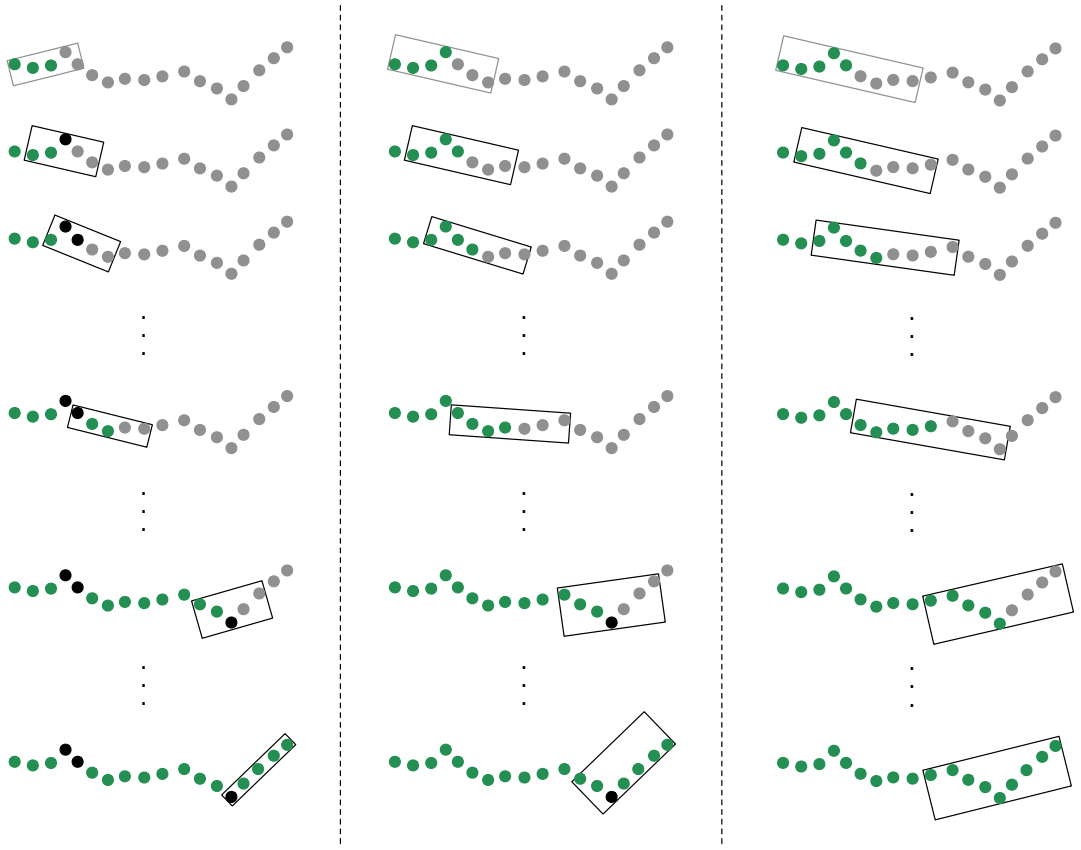


Figure 5.6: A 2D illustration of how points are tagged given k_{max}=5(left), 7(middle), 9(right) and α=2 and assuming a large value for r. The first, second and third row, illustrate the computation of the oriented rectangle over increasing values of k_{max}. For this particular case, as this value increases more points are labelled as *surface*. With k_{max} set to 9, the set of edge points P_e is empty. Grey, green and black indicate unspecified, surface and edge point types respectively.

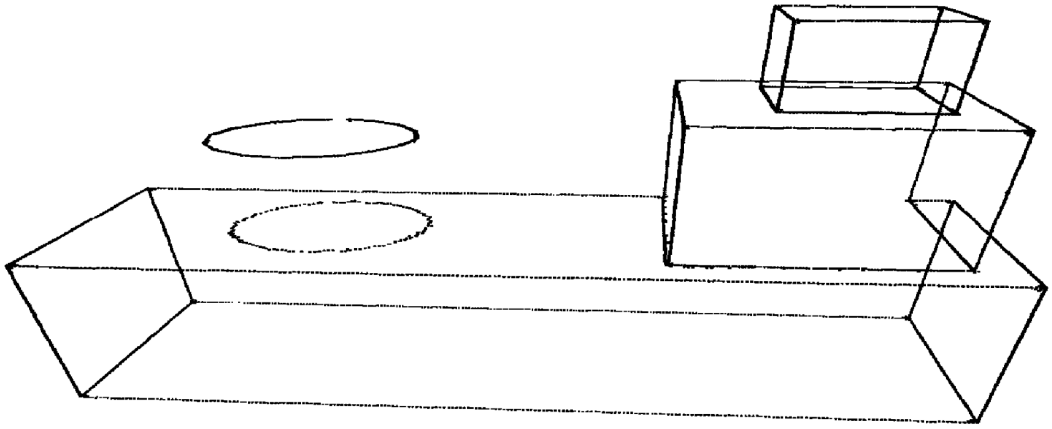


Figure 5.7: Edge points extracted from a point cloud of a synthesised scene with 4 primitive objects (three boxes and a cylinder). A number of important cues, for instance that the second box is aligned with the bottom box at the back is evident.

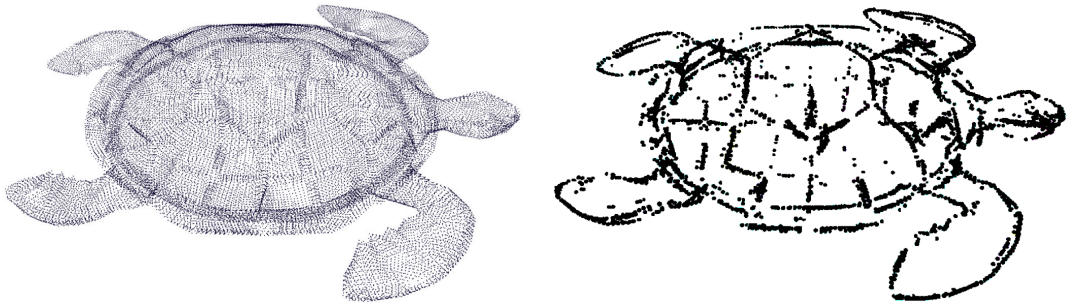


Figure 5.8: Points sampled from the surface of a turtle mesh are located either on a locally *smooth* or a *rough* area. Our segmentation pipeline builds on the distinction between these two categories of points. The ones located on rough areas are said to be of type *edge*. As shown on the right, for the tortoise example these points make up the outline of the 3D surface. All the other points are of type *surface*.

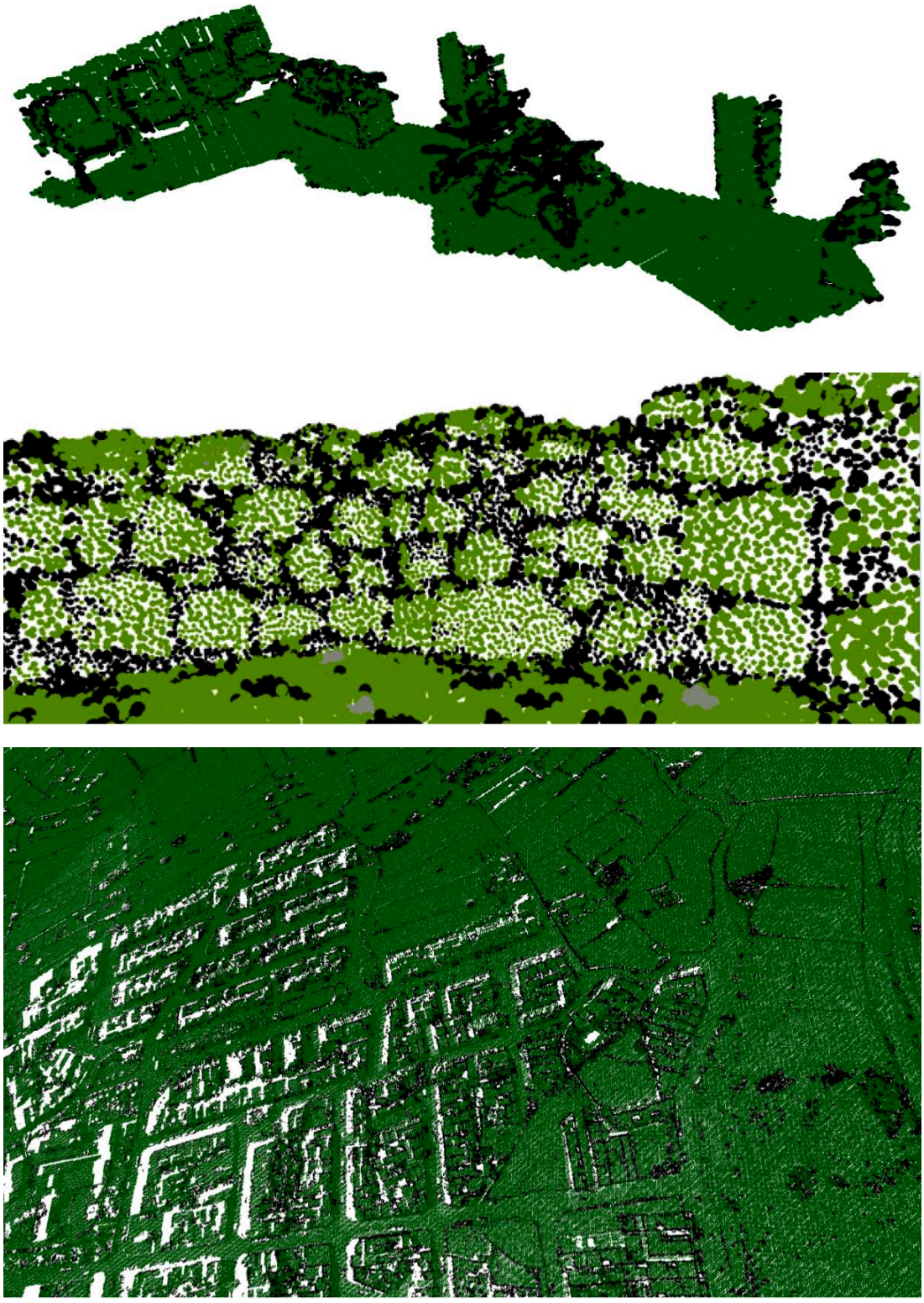


Figure 5.9: Edge points (coloured black) on a point cloud representing an indoor environment. Edge points on the temple apse point cloud are less obvious but clearly contribute to outline the contour of the individual stones making up the wall. Similarly, edge points delineate the buildings and field boundaries in the LiDaR point cloud.

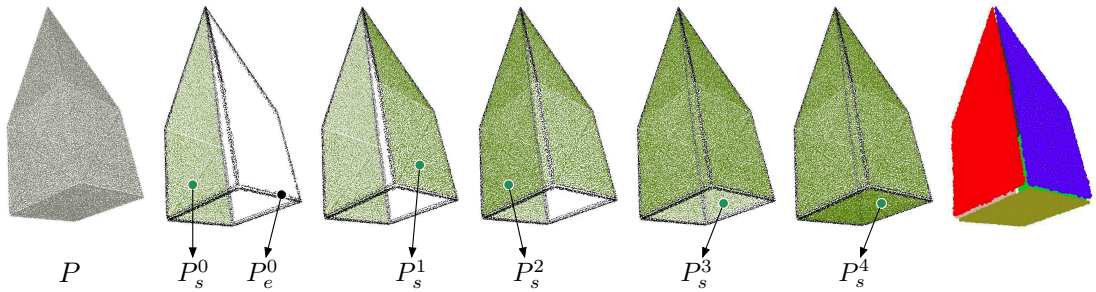


Figure 5.10: Segmentation of obelisk into edge and surface segments - starting from position only information all points are progressively assigned to segments, with all surface segments finally rendered using different colours.

unspecified points is discarded, since in practice, this set would usually be very small (with respect to P_e and P_s) and contain mostly outlier points. Point type information is used to determine the individual surface segments making up an object using a region growing process. Figure 5.10 shows the five region growing iterations required in order to partition P_s and P_e into $\{P_s^0, P_s^1, P_s^2, P_s^3, P_s^4\}$ and $\{P_e^0\}$ respectively. Each segment $P_s^n \subseteq P_s$ is surrounded by points from edge segment P_e^0. Note how P_e^0 is not divided into a number of different edge segments, for instance the four sides at the base, since no corner detection is carried out. This results from the design of the region growing algorithm, which only takes in consideration the type of the point and not other properties like surface normal. The generation of surface segments, i.e. the partitioning of P_e and P_s is described in the following section.

$$
\begin{aligned}
P &= \{p_0, p_1, \ldots, p_n\} \\
&= P_e \cup P_s \cup P_u \\
&= \{P_e^0, P_e^1, \ldots, P_e^n\} \cup \{P_s^0, P_s^1, \ldots, P_s^n\} \setminus P_u \\
&= \{\{P_e^{0.0}, P_e^{0.1} \ldots, P_e^{0.m}\}, \{P_e^{1.0}, P_e^{1.1} \ldots, P_e^{1.m}\}, \ldots, \{P_e^{n.0}, P_e^{n.1} \ldots, P_e^{n.m}\}\} \\
&\cup \{\{P_s^{0.0}, P_s^{0.1} \ldots, P_s^{0.m}\}, \{P_s^{1.0}, P_s^{1.1} \ldots, P_s^{1.m}\}, \ldots, \{P_s^{n.0}, P_s^{n.1} \ldots, P_s^{n.m}\}\} \setminus P_u
\end{aligned}
$$

5.2 Generation of Edge and Surface Segments

In addition to a set partition of segments, PaRSe generates a structure graph G, representing relations between the segments extracted. In order to produce G, a region growing algorithm is used to partition P_s and P_e and determine spatial connectivity across elements of these two set partitions. Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be an

undirected graph with nodes $n_i \in \mathcal{N}$ represent the set of segments $(S_s \cup S_e)$ where S_s and S_e represent the set partitions induced by region-growing over P_s and P_e respectively, and transitions $(n_i, n_j) \in \mathcal{E}$ corresponds to pairs of spatially adjacent segments. The union of points from all segment nodes $n_i \in N$ is equal to the set of points $P \setminus P_u$. A pairwise predicate *adjacent* is defined on segments n_i and n_j implementing the following set comprehension.

$$\{n_i, n_j : N | \exists p_i : n_i, \exists p_j : n_j \cdot (p_i \in \phi(p_j, r, k_{max}) \vee p_j \in \phi(p_i, r, k_{max})) \\ \wedge (\omega(p_i) \neq \omega(p_j)) \bullet adjacent(n_i, n_j)\},$$

where $\omega(p_i)$ returns the type of a point, and $\phi(p_i, r, k_{max})$ returns the set of k_{max} neighbour points of p_i within a distance r. A transition $e \in \mathcal{E}$, is created between all n_i and n_j nodes satisfying this set comprehension. Note that transitions are only created between segments of different type and therefore relation $\mathcal{E} \subseteq S_e \times S_s$. Given the possibility of an uneven distribution of points on a surface, both $\phi(p_i, r, k_{max})$ and $\phi(p_j, r, k_{max})$ are checked and a transition is created between n_i and n_j if either is satisfied. Figure 5.11 illustrates this scenario.

The pseudo code of the region-growing segmentation process implementing the set comprehension above returning a set of adjacent segments is listed in Algorithm 7. In order to improve the readability of the algorithm, the input is taken to be a point cloud P with point type information already established and assigned to P_e and P_s. In the pseudo code, type *Segment* refers to both a surface or edge segment, and the boolean array V is used to store the *visited* status of each point in $P \setminus P_u$. A point becomes *visited* as soon as it is associated with a segment. Two queues are maintained throughout the segmentation process. The first, Q_A, stores the currently active points, i.e. those points, retrieved by the neighbourhood query, to be considered next. The second queue, Q_R, stores potential new transitions to segments other than the currently active. Q_R is initialised with the pair $(p_i, s : Segment)$, where p_i, is randomly chosen from P_s. s is therefore a new surface segment seeded with the point p_i. Q_A is initialised by popping this first element from Q_R and pushing p_i onto the queue. The *activeType* property is set to the type of p_i, i.e. surface in this case. Function ω, depending on the input, returns the type of either point p_i or segment s.

The status of the two queues Q_R and Q_A determines the control flow of the algorithm. The outer while loop (line 3) uses Q_R to determine whether there are

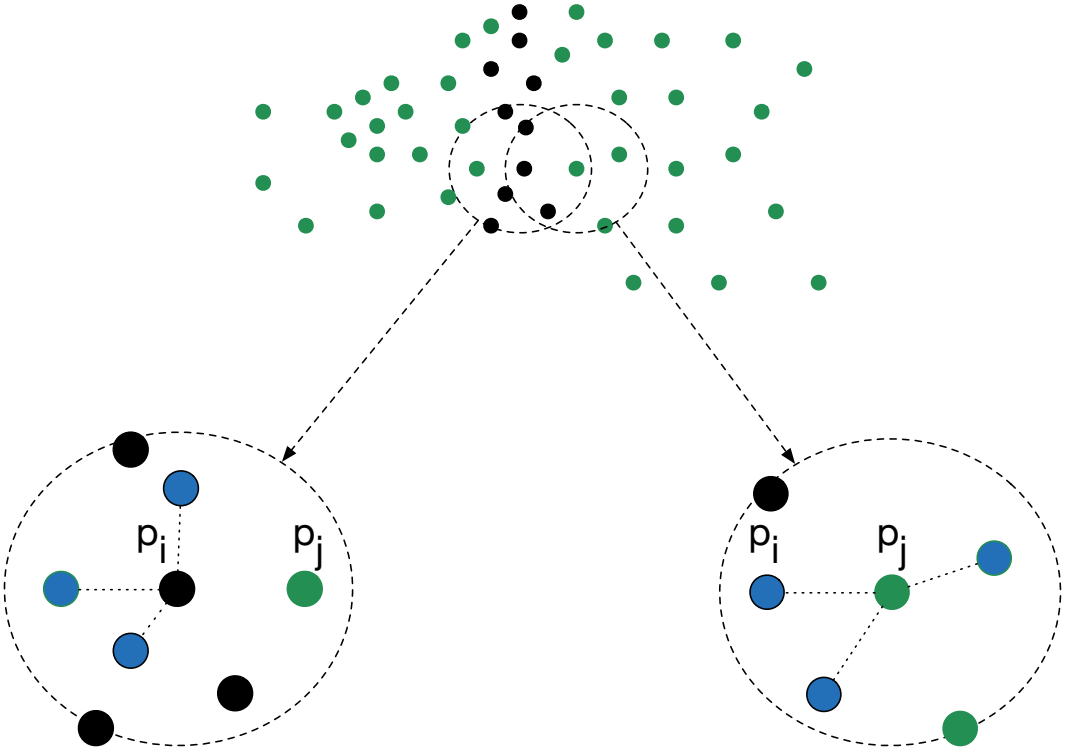


Figure 5.11: Neighbourhood membership is not a symmetric function, i.e. $p_j \in \phi(p_i, r, k_{max})$ does not imply $p_i \in \phi(p_j, r, k_{max})$.

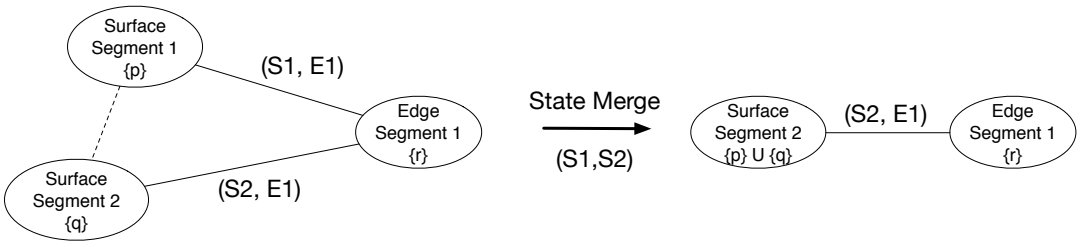


Figure 5.12: State merging takes two states n_i and n_j in $\mathcal{G}(\mathcal{N}, \mathcal{E})$ and joins them together by assigning points in n_i to n_j. Any transitions connected to n_i are transferred to n_j if not already present and n_i is then deleted.

any more potential segments that can be created, whereas the inner loop checks Q_A to determine whether there are any more points that can be added to the current segment. During the inner loop (line 6) the current point p_i is first added to the current segment s, then the points returned by the neighbourhood query $\phi(p_i, r, k_{max})$ are pushed onto Q_A if they are of the current active type. If not, the point together with a new instance of a segment s', as a pair, are pushed onto Q_R and the transition relation $\mathcal{E}$ is updated with the inclusion of the pair (s, s'). The algorithm implements the pairwise predicate *adjacent* between segments. In practice, when a point is assigned to a segment, rather than adding the point to the set of points making up s, point p is assigned a unique identifier for s. The inner loop (lines 6-25) has been implemented using a bag of tasks approach on multi-core hardware. This leads to the possibility of having two adjacent segments of the same type. In order to avoid this condition, a separate list M is maintained storing pairs of segments with these conditions (lines 20-22). At the end of the process, all pairs of segments in this list are merged together as shown in Figure 5.12.

5.2.1 Problems with Surface Generation

In a general context, it is very difficult to determine the performance of a segmentation algorithm as this usually depends on the specific task being performed. One method to assess the quality of segmentation is to have people manually segmenting a point cloud and then use a metric such as mean square error to quantify the difference between the manually crafted segments and those produced automatically. Chen *et al.* (2009) have done precisely this on 3D meshes of single objects. In our case, this is many times impractical given the size and variety of point clouds used. Nonetheless, a number of criteria can still be identified which

Algorithm 7 Region Growing Segmentation of Point Cloud P into $\mathcal{G} = (\mathcal{N}, \mathcal{E})$

Input: Sets P_e, P_s, $\mathcal{N}$, $\mathcal{E}$, k-NN function parameters for $\phi(p_i, r, k_{max})$, pairs of segments list M, boolean array V of size $|P \setminus P_u|$ and queues Q_A and Q_R.
Initialise: Assign all elements in V to $false$, create $new\ s : Segment$, enqueue Q_R with the pair ($p_i \in P_s$, s), $activeType$ set to $Surface$.
while Q_R is not empty **do**
    $(p_i, s) \Leftarrow^{deq} Q_R$
    $Q_A \Leftarrow^{enq} p_i$
    while Q_A is not empty **do**
        $p_i \Leftarrow^{deq} Q_A$
        $V[idx(p_i)] = \text{true}$
        $s \Leftarrow^{add} p_i$
        $nbr = \phi(p_i, r, k_{max})$
        for $k = 1$ to $|nbr|$ **do**
            if $V[idx(nbr_k)] == false$ **then**
                if $\omega(nbr_k) == activeType$ **then**
                    $Q_A \Leftarrow^{enq} nbr_k$
                else
                    $Q_R \Leftarrow^{enq} (nbr_k, new\ s' : Segment)$
                    $\mathcal{E} \Leftarrow^{add} (s, s')$
                end if
            else
                if $(\beta(nbr_k) \neq s) \wedge (\omega(nbr_k) == \omega(s))$ **then**
                    $M \Leftarrow^{add} (\beta(nbr_i), s)$
                end if
            end if
        end for
    end while
    $\mathcal{N} \Leftarrow^{add} s$
end while

result in either over or under-segmentation of an input point cloud P. Given an idealised set partition of P, over-segmentation will produce a set partition with a higher number of elements, whereas under-segmentation produces one with a lower number of elements. At the extremes, an over-segmented P results in a set partition where each element contains just one point, and under-segmentation produces a set partition with just one element containing all points.

An important factor determining the outcome of PaRSe is the density distribution of the points in the cloud as this directly affects the point set returned by the neighbourhood function. This function is first used to determine the type of each point then used when constructing the structure graph. For instance, due to the surface roughness present at the Mnajdra temple (Figures 5.2 and 5.14), sample density contributes when determining point type. Figure 5.13 shows a cross-section of a hypothetical surface with samples taken from it. When a higher sampling rate is used, many more points will be tagged as edge points as opposed to the lower sampled points. An important consideration, in order to minimise over-segmentation, is that the number of points k_{max} returned by the neighbourhood function to determine point type information is greater than or equal to the number of points returned by the neighbourhood function during region-growing. For the Mnajdra case study shown in Figure 5.14, k_{max} is set to 24 during point labelling, then to 6 during region-growing.

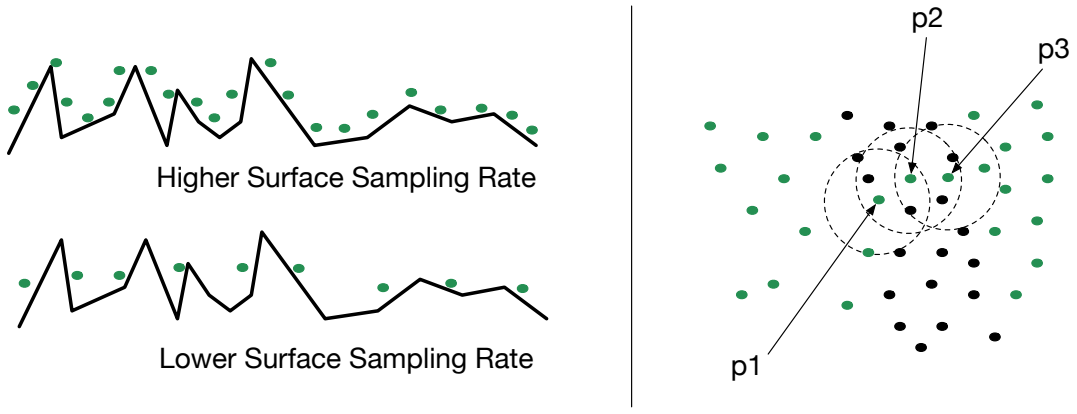


Figure 5.13: Variable sampling rates may result in different sets of surface segments over a specific region (left). Under-segmentation usually results from points labelled as surface, when ideally they should have been set to edge to contain the region growing process, for instance $p2$ above (right).

In the context of sites with complex surfaces, especially those where old and weathered stones are present, over-segmentation can easily occur. Figure 5.13

Figure 5.14: Panorama photograph of temple apse consisting of a number of stones. The bottom image illustrates the largest edge (black) and surface (green) segments members of P_e and P_s respectively, resulting from the region-growing phase of PaRSe. In many instances the density of the point cloud is good enough to delineate the individual stones.

(right) shows a simple boundary example where, if $p2$ had to be added to the current active queue, the current segment would end up with many more surface points, $p3$ and neighbours, which should really be a separate segment. Different heuristic measures can be adopted to minimise this occurrence, for instance using a minimum threshold on the number of neighbours with the same type before adding to the inner queue Q_A. This could minimise the possibility of under segmentation by measuring the evidence that a boundary between two regions exists. This measure can easily be incorporated in the region-growing process, which however cannot guarantee improvements. For instance, Figure 5.15 shows instances in the Mnajdra temple where over-segmentation occurs. Instances similar to the first case (top row) of over-segmentation are addressed in the next section, where a plane fitting process over the surface segments is carried out. In the second instance (middle and bottom rows) there isn't much that can be done, since the additional mortar is effectively filling the gap and thus joining the two stones together. For the case-studies presented in this chapter, this heuristic is not used.

Figure 5.16 illustrates another example of our segmentation process, this time applied on a synthesised point cloud (via the sampling of mesh triangles) of a conference room. The resulting set partition groups together points falling on individual objects (chairs, table, etc.) in the room. Another synthesised point cloud is shown in Figure 5.17 representing the Cornell box. Given the regular density of the point cloud, the region growing algorithm easily partitions the points into the different cube faces. The figure also shows the graph representing the different surface segments with edge states removed to improve clarity. The surface segment representing the floor is clearly visible in the graph, connecting the two inside boxes with the Cornell box. Note that in this case, the box face directly on the floor is separate to the floor and would not have been created if the scene was scanned and not synthesised by sampling the triangle primitives of the model.

5.3 RanSaC Plane Fitting

Following the region-growing process, plane primitives (parametrised with a tolerance value to include points close to the plane) are fitted to the segments from S_s and certain segments from S_e. The main intuition behind the use of planes

Figure 5.15: Over-segmentation during region-growing can easily occur on complex surfaces. In the top row, some surface points from the smaller stone are assigned to the large megalith. Over segmentation between the two circled megaliths in the photograph occurs because there is mortar placed between them.

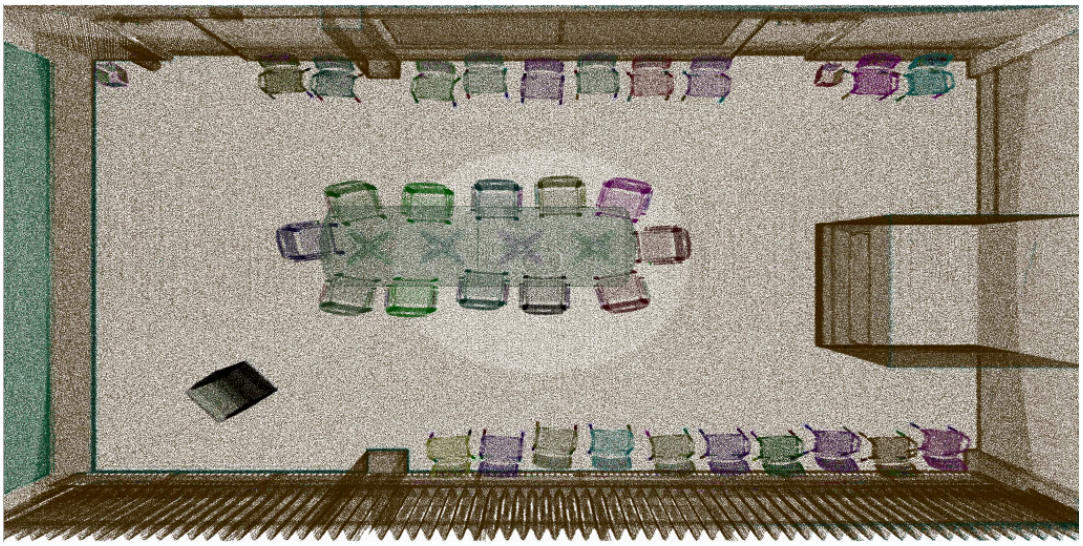


Figure 5.16: Segmentation of the conference room point cloud synthesised by sampling the triangle primitives making up the mesh. Different colours represent different nodes in the graph. In particular all the chairs in the room emerge as separate segments effectively resulting in a valid scene graph (§2.2.1) of the room from a raw point cloud.

is that these provide for an efficient representation of more diverse geometric objects as collections of planes connected together in a specific pattern. For example, the three apses of the Mnajdra temple might each fit a cylinder primitive, however, finer grain segmentation can be achieved when using a number of smaller planes each representing the individual stones composing the surface. Higher order shapes can in many cases be described using connectivity patterns of plane primitives, for instance roofs of houses or stairs in a house. In practice, this process seeks to transform a generic input point cloud P into a graph $\mathcal{G}$ describing connectivity information between segments which is amenable to a variety of domain-specific tasks.

In addition to the segments from S_s, a subset (possibly empty) of segments from S_e is also selected to undergo the RanSaC plane fitting process. This is mainly due to situations where either P is very noisy or where parts of the scene consist of very rough surfaces. In these cases, points may be labelled as edge points, resulting in an edge segment being created instead of a surface segment. A number of tests are carried out on all edge segments $s_e \in S_e$ in order to determine this subset S'_e, defined using the following set comprehension.

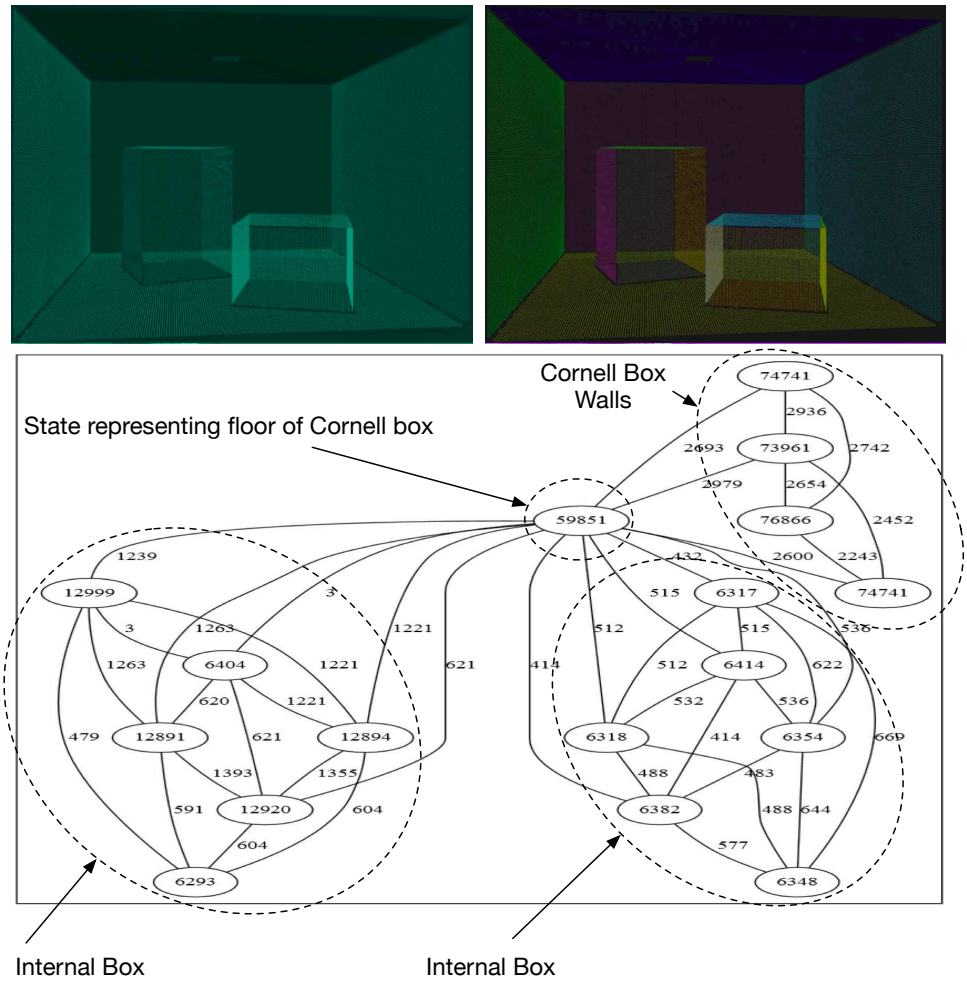


Figure 5.17: Segmentation of the conference room point cloud synthesised by sampling the triangle primitives making up the mesh. Different colours represent different nodes in the graph. In particular all the chairs in the room emerge as separate segments.

$$\{s_e : S_e |\ |s_e| > n \wedge connCount(s_e, G) < c \bullet s_e\},$$

where n represents the minimum number of points in the segment and c the maximum number of adjacent surface segments. Edge segments which satisfy these conditions are passed over for plane fitting. By default n is set to the average number of points in elements of S_s, and c is set to 3 in all examples. For instance, in the case of the Cornell box (Figure 5.17), none of the three edge segments is considered for plane fitting as they violate the size condition, whereas for the Mnajdra temple point cloud, the edge segment shown in Figure 5.15 violates the number of connected surface segments condition and is thus not considered.

Segment type is further refined during the plane fitting process and associated with the resulting elements of the set partitions produced for each processed segment. The following lists the segment types used.

- *surface·planar*: Segment is a collection of surface points which fit the parameters of a plane.
- *surface·complex*: Segment is a collection of surface points over which no plane could be fitted.
- *edge·planar*: Segment is a collection of edge points which fit the parameters of a plane.
- *edge·complex*: Segment is a collection of edge points over which no plane could be fitted.

The RanSaC plane fitting process (§ 2.7), takes surface segments in S_s and edge segments in S'_e, and determines whether there exists plane primitives which fit the data. Since the points (within a single segment) will nearly never perfectly fit a plane, a tolerance parameter, ϵ, is attached to each plane, i.e. points are allowed to fit the plane whenever the perpendicular distance d from a point to a plane is within the range $-\epsilon \leq d \leq \epsilon$. Figure 5.18 illustrates these parameters. For each segment, three points are randomly chosen to define the plane parameters. In order to decrease the time required to establish the best fitting plane, the three randomly chosen points are immediately discarded if any two of them have orthogonal surface normals or the three are collinear. If the three points are

Algorithm 8 RanSaC Plane Fitting and Creation of New Nodes in $\mathcal{G}$.

Input: $\mathcal{G}(\mathcal{N}, \mathcal{E})$, P_s, P'_e, Tolerance ϵ, Trials c, Threshold t, Stability s.
for all $P^i \in P_s \cup P'_e$ **do**
  $planesfitted = 0$
  $fitmoreplanes = true$
  while $fitmoreplanes$ **do**
    $fitmoreplaces = false$
    $bestscore = 0$
    $bestplane = null$
    $stablecount = 0$
    while $stablecount < s$ **do**
      while $trialscount > 0$ **do**
        $crtplane_{minset} = \{p_{1..3} : P^i | p_1 \neq p_2 \neq p_3\}$
        for each $p \in P^i$ **do**
          if $(compatible_{plane}(p, crtplane_{minset}, \epsilon))$ **then**
            $inc(crtscore)$
          end if
        end for
        $dec(trialscount)$
      end while
      if $(crtscore > bestscore)$ **then**
        $bestplane = crtplane_{minset}$
      else
        $inc(stablecount)$
      end if
    end while
    $points_{bestplane} = \{p : P^i | compatible_{plane}(p, bestplane, \epsilon)\}$
    if $(|points_{bestplane}| > |P^i| * t)$ **then**
      $new\ s : Segment$
      $P^i = P^i \setminus points_{bestplane}$
      $s \Leftarrow^{add} points_{bestplane}$
      $inc(planesfitted)$
      $fitmoreplanes = true$
      if $P^i \in P_s$ **then**
        $s \Leftarrow^{type} surface \cdot planar$
      else
        $s \Leftarrow^{type} edge \cdot planar$
      end if
    end if
  end while
  if $(planesfitted == 0 \wedge P^i \in P_s)$ **then**
    $P^i \Leftarrow^{type} surface \cdot complex$
  else
    $P^i \Leftarrow^{type} edge \cdot complex$
  end if
end for

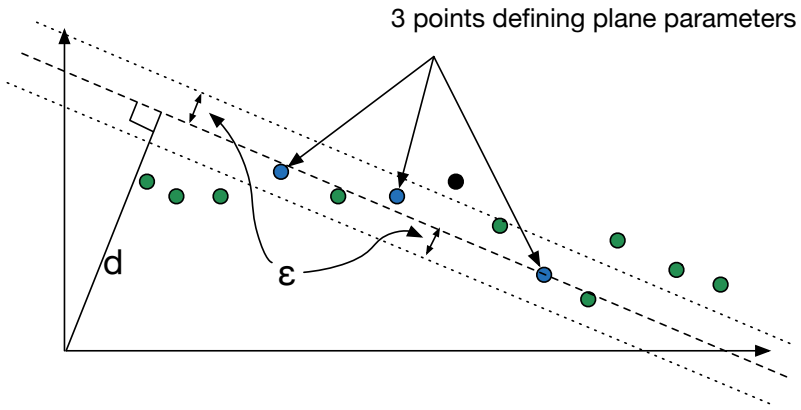


Figure 5.18: Plane parameters d representing perpendicular distance of plane from origin, and tolerance parameter ϵ

valid, the plane parameters resulting from them are used to calculate the percentage of points from the segment which fit this model. Given that any three points will fit a plane, a minimum percentage threshold t is set before accepting the model. Different triples of points are repeatedly chosen until this percentage gets stable, i.e. does not improve over a number of iterations. If the percentage of points fitting the plane is below a certain threshold, further plane fitting is carried out on the remaining points in the segment. Algorithm 8 illustrates this process which results in the creation of new nodes whose type is set to an element from the ones listed above. No connectivity information is set at this stage between the newly created segments. Note that when splitting a $surface$ node into for instance two *surface·planar* nodes no information is lost. Figure 5.19 shows how the previously segmented obelisk (five surface segments in S_s and one edge segment in S_e) is now fitted with nine planes each of type *surface·planar*. The four vertical segments in S_s have been fitted to two planes each, whereas the bottom horizontal segment also in S_s easily fits within one plane. Plane fitting is not applied on the element in S_e as the number of points in the segment is less than the average. The topmost graph $\mathcal{G}$ representing the region growing segmentation process initially consists of five nodes and five transitions. During the RanSaC plane fitting process eight additional nodes are created. In order to improve clarity, only 4 nodes are shown in Figure 5.19.

Nodes have an associated type and each should be reachable from a suitably selected root node. Each transition $e \in \mathcal{E}$ is also assigned a type depending on the way the connection between source and sink nodes is established. Initially,

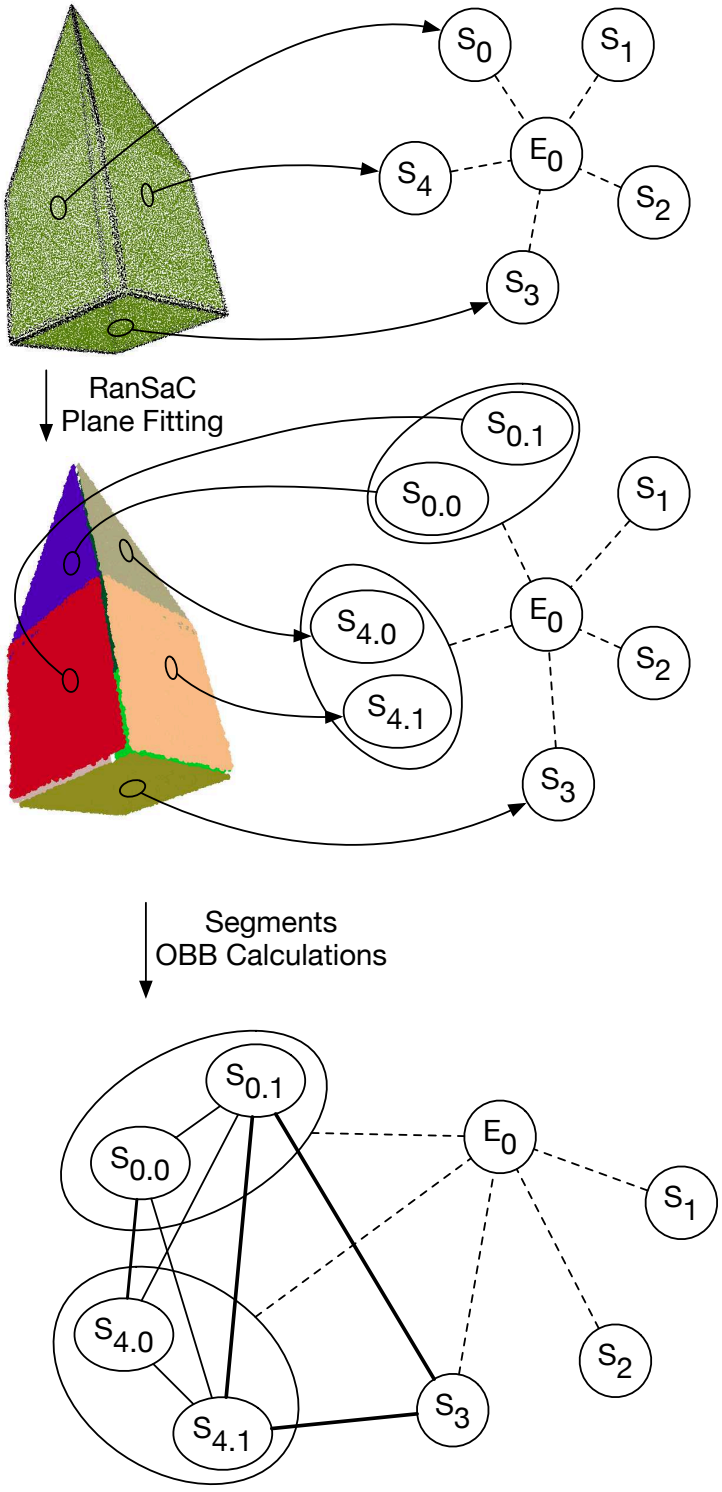


Figure 5.19: Plane primitives fitted to obelisk segments.

Algorithm 9 Finalise transitions for $\mathcal{G}(\mathcal{N}, \mathcal{E})$.

Input: $\mathcal{G}(\mathcal{N}, \mathcal{E})$, list $psegs$ initially empty to store planar segments.
▷ Compute OBBs for all planar nodes in $\mathcal{G}$
for each $(s \in \mathcal{N})$ **do**
  if $\omega(s) == (surface \cdot planar \vee edge \cdot planar)$ **then**
    Compute OBB and surface normal for s
    $psegs \Leftarrow^{add} s$
  end if
end for
▷ Establish connections between planar nodes in $\mathcal{G}$
for k=1 to $|psegs|$ **do**
  $s_{parent} = ParentSegment(psegs_k)$
  $nbr \Leftarrow^{add} ChildrenOf(s_{parent})$
  for each $(s' \in \mathcal{N})$ **do**
    if $(s', s_{parent}) \in \mathcal{E}$ **then**
      $nbr \Leftarrow^{add} ChildrenOf(s')$
      for j=1 to $|nbr|$ **do**
        if $OBBIntersect(psegs_k, nbr_j)$ **then**
          if $(psegs_k, nbr_j) \notin \mathcal{E}$ **then**
            $\mathcal{E} \Leftarrow^{add} (psegs_k, nbr_j)$
            Assign properties to transition
          end if
        end if
      end for
    end if
  end for
end for
▷ Connect disjoint segments to $\mathcal{G}$
$r = selectRootNode(\mathcal{G})$
$disjointNodes \Leftarrow^{add} notReachableFrom(r)$
$Sort(disjointNodes)$
for j=1 to $|disjointNodes|$ **do**
  if $NotConnected(r, disjointNodes_j)$ **then**
    $s = ClosestNode(disjointNodes_j)$
    $\mathcal{E} \Leftarrow^{add} (disjointNodes_j, s)$
    Assign properties to transition
  end if
end for

the region growing process creates transitions between *edge* and $surface$ nodes. During RanSaC plane fitting, additional nodes are created each with a link to its parent $surface$ or *edge* node. Algorithm 9 completes the creation of $\mathcal{G}$ by establishing connections for the newly created planar nodes. An oriented bounded

box (OBB) is first computed over nodes of type *surface·planar* or *edge·planar*, and then used to check the following segment connectivity information.

- Establish whether $surface$ segments connected to the same *edge* segment are really adjacent to each other
- Determine connectivity between the newly created planar segments

In the first instance, consider for example a point cloud of a box. Since region growing segmentation does not determine corners, it therefore does not create eight *edge* segments but just one. This results in a structure graph where all faces of the box are connected to each other irrespective of distance. OBB intersection tests provide the information necessary to discriminate between $surface$ segments that are really close to each other and others which are not, e.g. the parallel faces of a box. OBB volumes are incremented by a small percentage of their original total volume in order to make sure adjacent segments actually overlap. Figure 5.20 illustrates this simple example. $Surface$ segments S_2 and S_4 are connected through *edge* segment E_1, however they are not spatially adjacent to each other and therefore no transition is created between the two *surface·planar* segments. Note that in practice a new *surface·planar* segment is created for each of the six segments in this example but these are not shown here in order to maintain visual clarity. The new nodes, for instance $S_{1.1}$, are connected to the other newly created planar nodes and a parent-child link is established between $S_{1.1}$ and S_1. This information is used to gather the neighbouring segments for OBB intersection tests (lines 11,12 and 14 of Algorithm 9) and avoid having to carry out the test against all planar segments (line 16).

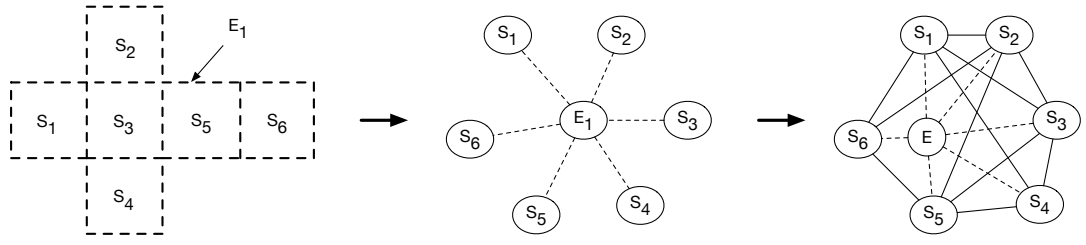


Figure 5.20: Initial connectivity information does not take in consideration distance between segments but connectivity between *edge* and $surface$ segments. OBB transitions take in consideration distance following the computation of OBB volumes around planar segments and transition relation $\mathcal{E}$ is updated accordingly.

In cases where a point cloud consists of spatially distant clusters of points, possibly resulting from occlusion, the transition relation $\mathcal{E}$ might not be enough to reach every node from the *root* which is established using a heuristic based on number of points and volume covered from amongst planar segments. In cases where no planar segments exist, the *root* node is chosen using the same heuristic from all segments. Given a node designated as *root*, PaRSe enforces the existence of a path between all pairs of nodes by enforcing the existence of a path between all nodes and the *root* node. The type of these connections is different from that created by the region growing process and OBB intersection tests and are mainly used to determine distances between clusters of points which could be useful when analysing the point cloud. Consider for instance, a point cloud representing a flight of stairs acquired from a birds eye view position as illustrated in Figure 5.21. In this case the transition relation $\mathcal{E}$ would be empty and $\mathcal{G}$ would consist of four disjoint segments. The last steps of Algorithm 9 ensures that a relation is created between the four segments.

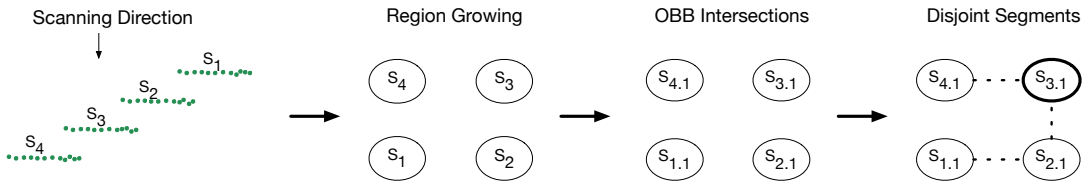


Figure 5.21: Initial connectivity information does not take in consideration distance between segments but connectivity between *edge* and $surface$ segments. OBB transitions take in consideration distance, following the computation of OBB volumes around planar segments and transition relation $\mathcal{E}$ is updated accordingly.

Besides types, both nodes and transitions in $\mathcal{G}$ have associated properties. In the case of nodes, properties include surface normal and number of points. In addition to types, properties are used to discriminate between nodes and sequences of nodes. For instance, the sequence of adjacent nodes whose surface normals differ by a maximum of 20°. In the case of transitions, properties include Euclidean and surface normal distances between the two connected nodes. In order to allow for extensibility for user defined properties, these are described in the form of $\langle key, value \rangle$ pairs, e.g. $\langle dot, 0.02 \rangle$ to indicate that the nodes connected by this transition are nearly orthogonal.

5.4 Point Cloud Queries

In a traditional relational database, a structured query language (SQL) is used to manipulate and select relevant subsets of data. A similar approach can be applied on the structure graph $\mathcal{G}$ produced by the segmentation process described above. In particular, the extracted planar and complex segments, provide the required structure necessary to enable reasoning about and querying of point clouds. Specifically, a query is encoded as a graph whose nodes and transition relation specify the constraints and predicates to be satisfied within the structure graph of P. A query can be applied in two ways as follows:

- by choosing a specific *seed* segment in G and recursively returning all nodes (segments) which satisfy the query graph as one transition tree (§2.2.2) with the *seed* segment as root,
- by searching for patterns matching the query graph in all G and returning a set of transition trees.

In the first case, the result consists of one set of segments starting from the chosen *seed* segment and including all nodes in the expanded transition tree. In the second instance, a set of transition trees each representing a set of segments matching the constrains imposed by the query graph are returned. Figure 5.22 illustrates the query graph used to extract cylinders from a point cloud. Node s_0 is the *root* of the query graph. Actions and predicates are attached to both nodes and transitions and are shown in Figure 5.22 in boxes, where actions are listed in the text above the horizontal line in the box, whereas predicates are listed below the line. For instance, transition (s_0, s_1) is followed, only if the angle between the two planar segment surface normals N_0 and N_1 is found to be greater than 100°. If the transition is followed, meaning that there exists a transition (n_0, n_1) in $\mathcal{G}$ which satisfies this condition, a transition tree starts to form with n_0 as root and n_1 as its only child. Since the orientation of a cylindrical object can be described using a plane, this is computed from the two surface normals N_0 and N_1. For any other planar segment, node in $\mathcal{G}$, to be added to the transition tree, it now has to satisfy two conditions, namely that the angle between adjacent segments is more than 100° and that the normal of the new segment N_{dst} is within the plane parameters of the cylinder. In a scenario where more than one transition

in $\mathcal{G}$ satisfies the condition attached to transition (s_0, s_1) in the query graph, for instance (n_0, n_1) and (n_0, n_2), then both n_1 and n_2 are attached to the root node of the transition tree and s_1 is mapped to the set $\{n_1, n_2\}$. Finally, if the next adjacent node being added is the root of the transition tree, the current branch in the transition tree is terminated and the result returned indicating that a full cylinder was detected.

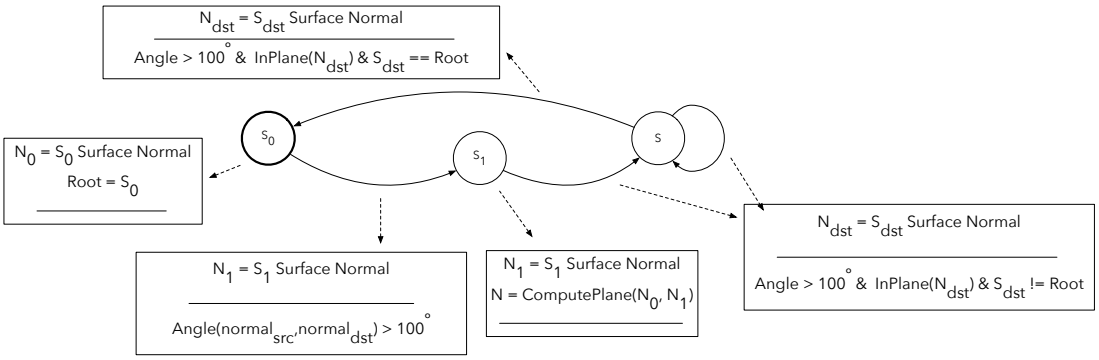


Figure 5.22: A query graph describing a cylinder using geometric constrains across connected planar segments. S_0 represents the initial state of the graph and root of the transition tree in $\mathcal{G}$ of the input point cloud P.

Queries such as the one just described are best suited to scenarios where a number of instances exist in the point cloud compatible with a particular shape, for instance when detecting columns or roof structures in large points clouds. In other cases, the selection of a portion of the point cloud given a specific starting point is useful, for instance in order to determine all objects placed directly on the ground or on a table. In these cases, a specific segment is chosen in $\mathcal{G}$, referred to as the *seed*, from which a transition tree is built by moving across adjacent segments which satisfy node and transition constraints similar to the ones discussed above.

Figure 5.23 illustrates two query graphs used to transitively select segments connected to a *seed* segment. In left-most query graph, surface normals and size in number of points in the segment are used as constraints between adjacent segments. On the other hand, the right-most query graph, the angle comparison is carried out between the normals of each segment and the *seed*. In both cases, the result of the query (or search) would consist of a transition tree with all the planes connected to the seed which satisfy the transition constraints.

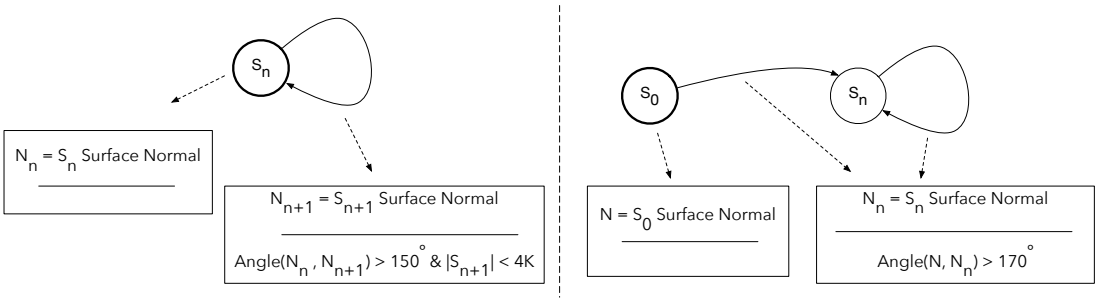


Figure 5.23: Two query graphs using seed segments. The first transitively returns nodes with more than 4K points which are connected at 150°. The second returns connected nodes which are at an angle more than 170° from the seed plane normal.

Reference	$\mathbf{k}_t$	$\mathbf{k}_f$	α	**r**	$\lvert P\rvert$	$\lvert P_s\rvert$	$\lvert P_e\rvert$	S_p	S_c	T_l	T_r	T_s
Mnajdra	24	6	12	1.2	593	320	226	823	211	2.18	2.13	11
Tarxien	18	6	20	1.2	2,515	2,119	387	977	142	18	5.2	152
Villard	36	6	8	1.2	7,222	5,459	1,762	1495	2391	47	291	99
Office Room 1	48	24	20	1.2	10,655	8,319	2,336	243	3293	82	64	42
Office Room 2	48	24	20	1.2	10,211	7,837	2,374	316	2853	79	64	16
Warwick Area	24	12	20	1.2	17,291	9,669	7,620	3787	9842	95	196	111
Airborne LiDaR	18	6	20	1.2	5,099	3,681	1,320	5822	149	23	111	289

Table 5.1: Point cloud segmentation parameters used in results section and resulting number of point and segment types. From left to right columns show; Reference: name of point cloud; k_t: the number of k nearest neighbours used in the labelling phase; k_f: the number of k nearest neighbours used in the region-growing phase; α: the eigenvalues ratio; r: the plane tolerance value used for RanSaC plane fitting; $|P|$: total number of point cloud samples (in thousands); $|P_s|$: total number of points labelled as surface points (in thousands); $|P_e|$: total number of points labelled as edge points (in thousands); S_p: total number of $* \cdot$ *planar* segments; S_c: total number of $* \cdot$ *complex* segments; T_l: time taken for labelling of points (in seconds); T_r: time taken for region-growing process (in seconds); T_s: time taken for RanSaC plane fitting (in seconds).

5.5 Results

PaRSe is evaluated on point clouds acquired using a variety of scanners. In the next section, segmentation is used for the extraction of cylinder primitives describing columns in a synthesised point cloud. Section 5.5.2 then looks into the application of PaRSe to point clouds representing three CH sites. The Mnajdra and Hal-Tarxien point clouds were made available by Heritage Malta, the national agency for museums, conservation practice and cultural heritage in Malta. Two indoor scenes from Mattausch *et al.* (2014) are then used to demonstrate the applicability of PaRSe to indoor environments. PaRSe is then used for the extraction of trees on a point cloud representing an outdoor area at the university of Warwick. Finally, PaRSe is applied on an airborne LiDaR acquired point cloud representing a section of the Maltese archipelago. Table 5.1 lists the segmentation parameters used and some statistics related to the number of points, segments and execution times of each example. In all cases the number of RanSaC trials c is set to 3000 and the fit ratio r to 10% of the size of the segment currently being processed (§2.7). An Intel Core-i7 960 machine (3.2GHz) with 8Gb of RAM is used for all examples.

5.5.1 Synthesised point cloud - Kalabsha Temple

In this section, PaRSe is applied on a synthesised point cloud representing the Kalabsha temple, mainly to demonstrate the cylinder extraction query graph. The point cloud was generated by sampling the triangle surfaces making up the Kalabsha 3D model (Sundstedt *et al.*, 2004) and consists of 1.9 million points. Each point is translated by a very small amount in a random direction to simulate sensor noise. Query graphs for cylinder and box fitting are used to extract the columns in the temple. Figure 5.24 illustrates the raw point cloud (top-left), the assignment of point types (top-right) and the generation of segments (bottom). Figure 5.25 top row shows the central part of the temple where the columns are located. A considerable amount of clutter is visible, with many segments generated by the region-growing and RanSaC plane-fitting algorithms. Figure 5.25 shows how planes are fitted along the columns present in the temple. Two different query graphs are used to extract columns, one for detecting cylinders and a similar query graph for detecting boxes. Results from these two queries are combined in order to filter the set of boxes returned by the set of cylinders using

a constraint where a box is accepted only if a cylinder is present beneath it. The middle row shows the segments (from region growing) and their partitioning into *surface·planar* segments. The bottom row shows another set of columns detected using the cylinder query graph. Note that in this case each column is actually made up of two cylinders, since the central portion of each column consists of an *edge·complex* segment. Additional work can be carried out in order to include points in these segments, by checking whether they are connected to cylinders, but this is currently not implemented.

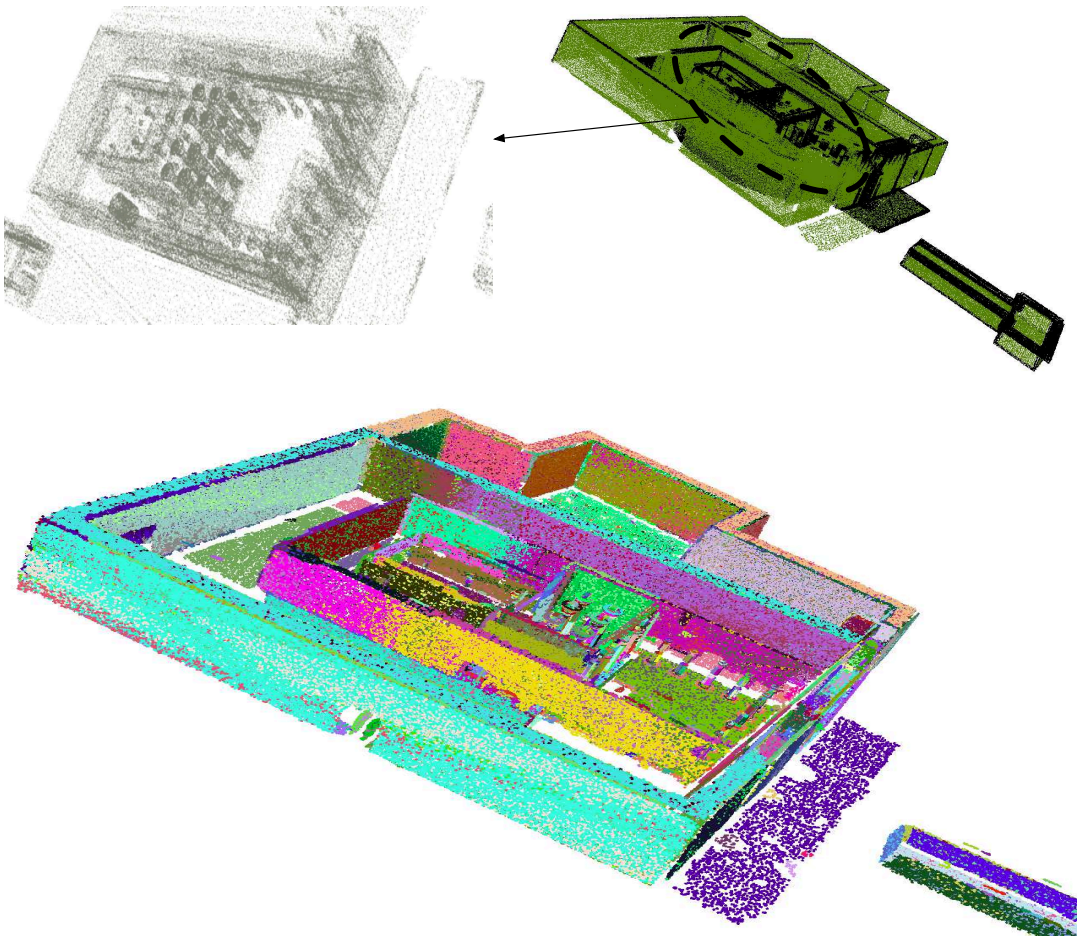


Figure 5.24: Kalabsha point cloud synthesised from 3D Model. The point cloud is shown rendered as raw (top-left), following point type assignment (top-right) and then region growing algorithm (bottom).

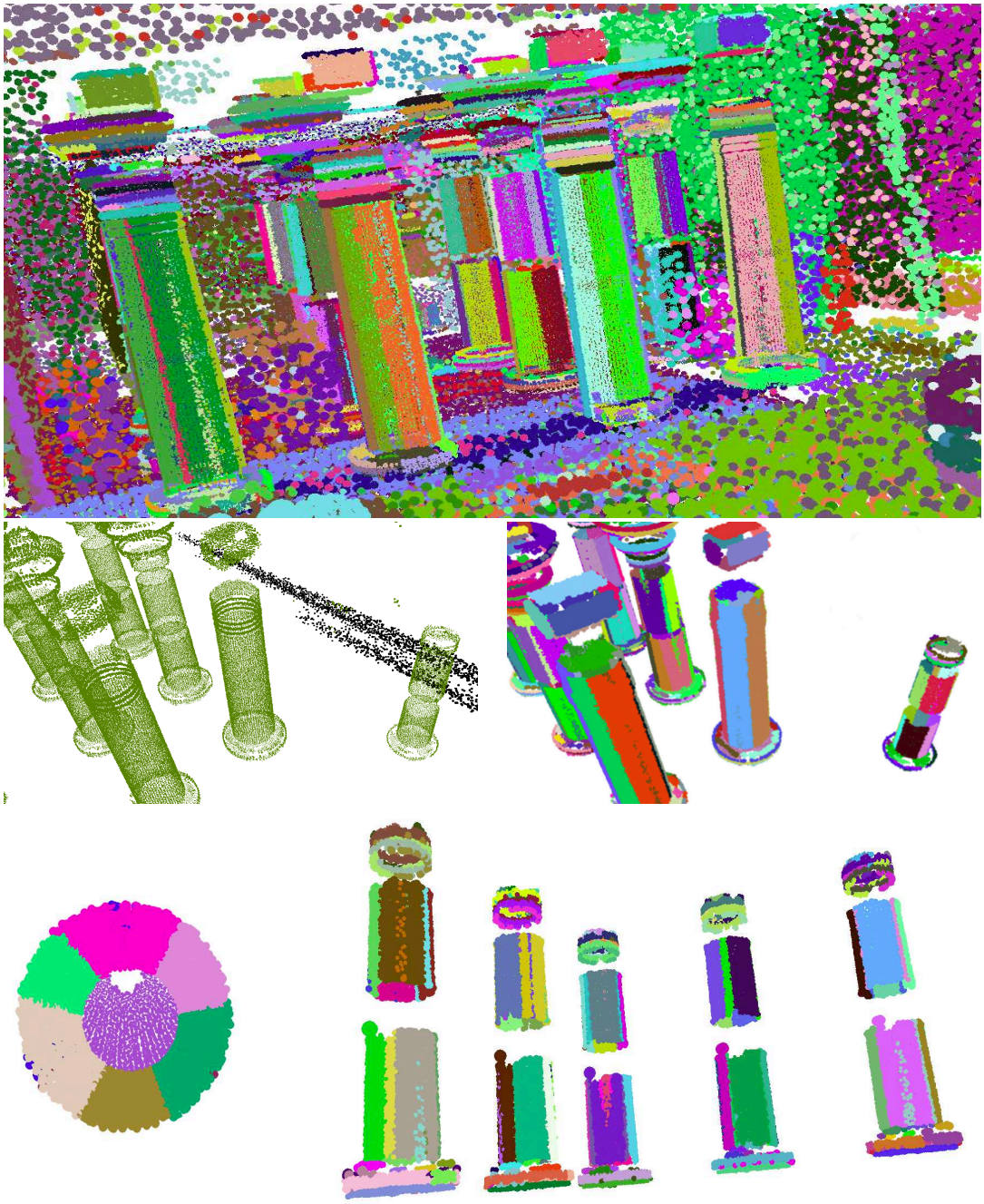


Figure 5.25: Individual columns extracted via the segmentation process are fitted with planes.

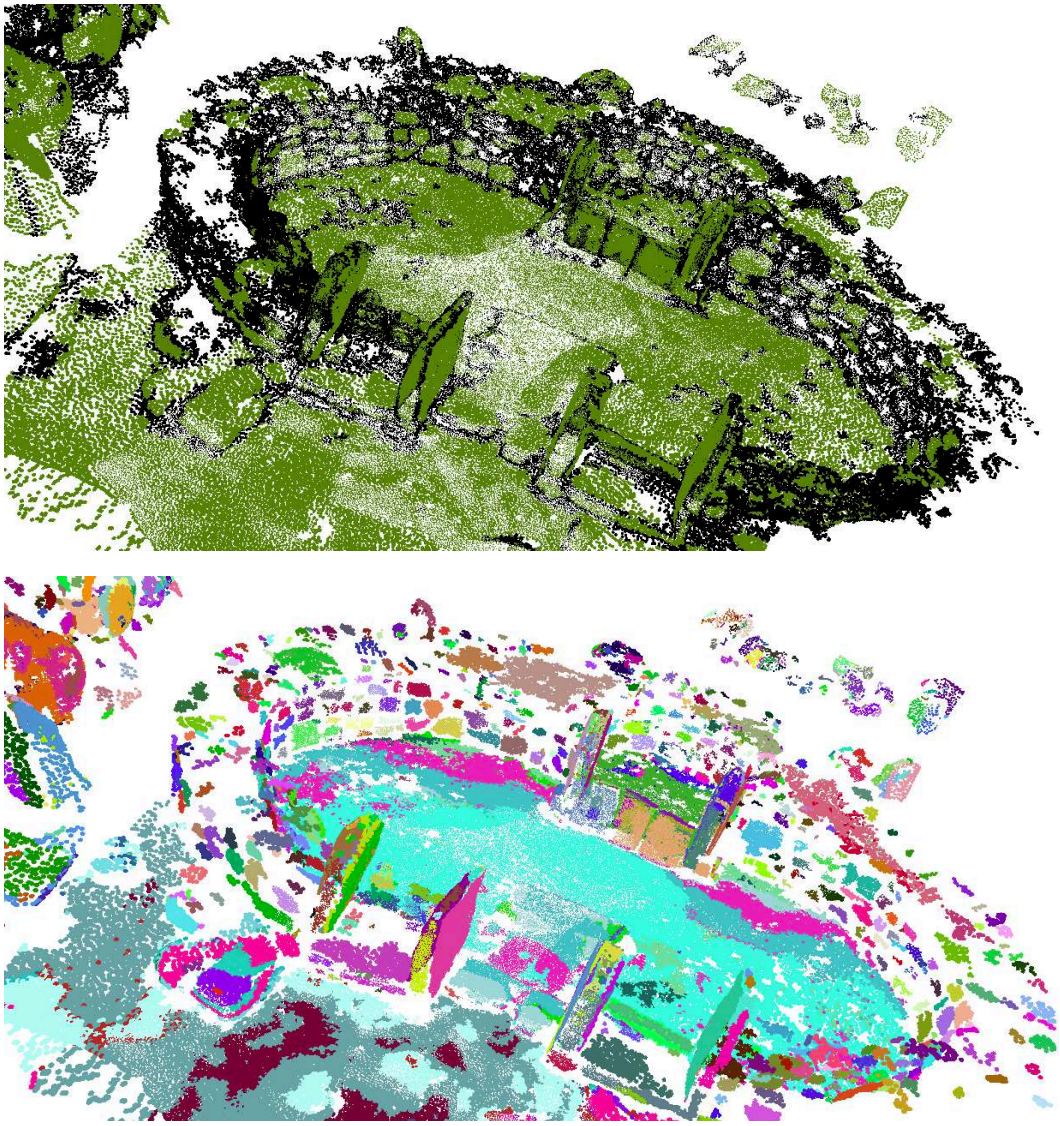


Figure 5.26: Point labelling (top) and *surface·planar* segments (bottom) extracted from the Mnajdra pre-historic temple.

5.5.2 Cultural Heritage Sites

Point clouds *Mnajdra* (Figure 5.26) and *Tarxien* (Figure 5.29) acquired from two Maltese CH sites are used in this section of the evaluation, in addition to a point clouds acquired from a stone church in the town of Lans le Villard, France. *Mnajdra* represents a section of the Mnajdra pre-historic temples site that was acquired in 2005 by Heritage Malta. The modelled surface precision was stated as +/- 2mm. Due to considerable stone erosion there are hardly any smooth surfaces present. In this case study, segmentation is used to discriminate between the various stones composing the temple apses. Figure 5.26 shows the point cloud rendered after point type labelling (top) and following RanSaC plane fitting (bottom). The image shows all the planar surfaces (rendered using different colours) fitted over all the different surface segments of the Mnajdra temple. The partitioning induced by RanSaC is efficient and reliable due to the fact that planes are fitted on subsets of related points (produced by the region-growing process) rather than the whole data set. The 98% of the surface points are fitted to 930 plane primitives. Without prior segmentation, RanSac does not converge properly and only two planes (see Figure 5.28 top right corner) are fitted to the floor of the temple in approximately 4 minutes. Moreover, each of the two plane primitives cover points which are found on different, unrelated parts of the temple. PaRSe outputs a set partition with geometric planes fitting the data in a very accurate way with the general structure of the temple clearly visible.

Two queries are carried out on the resulting structure graph to extract the walls and the floor of the temple. Figure 5.27 illustrates the results returned by these two queries. In the first instance, three seed segments are used in order to return the three components of the temple. Note how the query search follows the rubble wall until megalith stones positioned perpendicular to the wall are encountered. Figure 5.28 shows the largest surface and edge segments resulting from the region growing process. As would be expected the largest surface segment consists of points sampled from the floor of the main part of the temple, whereas the largest edge segment is made up of points connecting the entire rubble wall structure. The middle row provides a closer view (of part) of the rubble wall present in the site. Around 35 stones are automatically identified in this part alone. The edge segment contributes towards the partitioning of the rubble wall into a number of surface segments representing the individual stones making the apse.

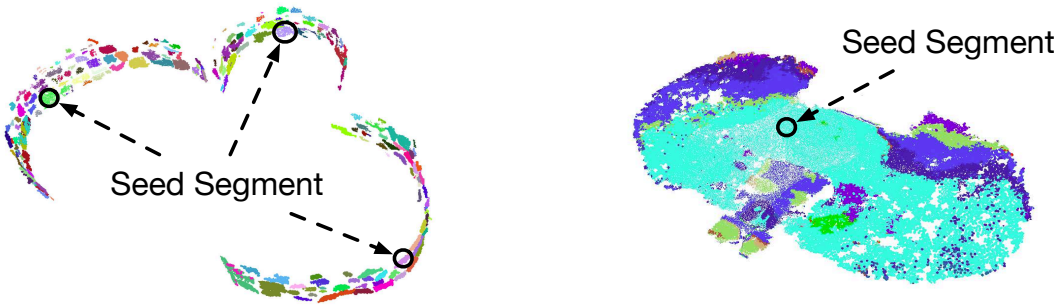


Figure 5.27: A query graph (using seeds) is used on segments from each apse wall to return the internal walls of the temple is shown on the left. Another query is used to return the points making up the main floor (within the apses) of the temple.

Figure 5.29 shows the segmentation results obtained for the Hal-Tarxien temples. As in the case of Mnajdra, the Hal-Tarxien data set was made available by Heritage Malta. 728 surface segments are directly planar segments and an additional 109 segments are split into **·planar* segments, resulting in a total of 977 **·planar* segments. Figure 5.30 shows the main structures within the Hal-Tarxien pre-historic site, with the apses and floors of both temples easily identified on the left-hand side as surface segments. The right-hand side image shows the edge segments, which effectively provide an outline of the temple structure. Figure 5.31 zooms into two particular features of the site, a cylindrical bowl and stairs. In the first case, the bowl's points falling on the irregular top structure are represented by a unique surface segment resulting from the region-growing process. In the second case, 20 *surface·planar* segments represent a series of steps inside the site. Each step is represented by approximately 100 points, totalling around 0.07% from a total of 2.7 million points covering the site.

The region-growing phase of PaRSe is essential to the successful extraction of meaningful elements in the site. Figure 5.32 demonstrates the results obtained (using the same parameters) when RanSaC plane fitting is directly applied to the point cloud. The largest segments consists of 281K points and consisting of several points sampled from unrelated parts of the site. The resulting set partition (only the largest 6 segments of 148 are shown) is not useful in distinguishing between the different components of the site and would certainly require a CH professional a considerable amount of time to work with.

Figure 5.33 illustrates the point cloud of a patrimonial stone church in the town of Lans le Villard, France. The raw point cloud is available on the Aim@shape

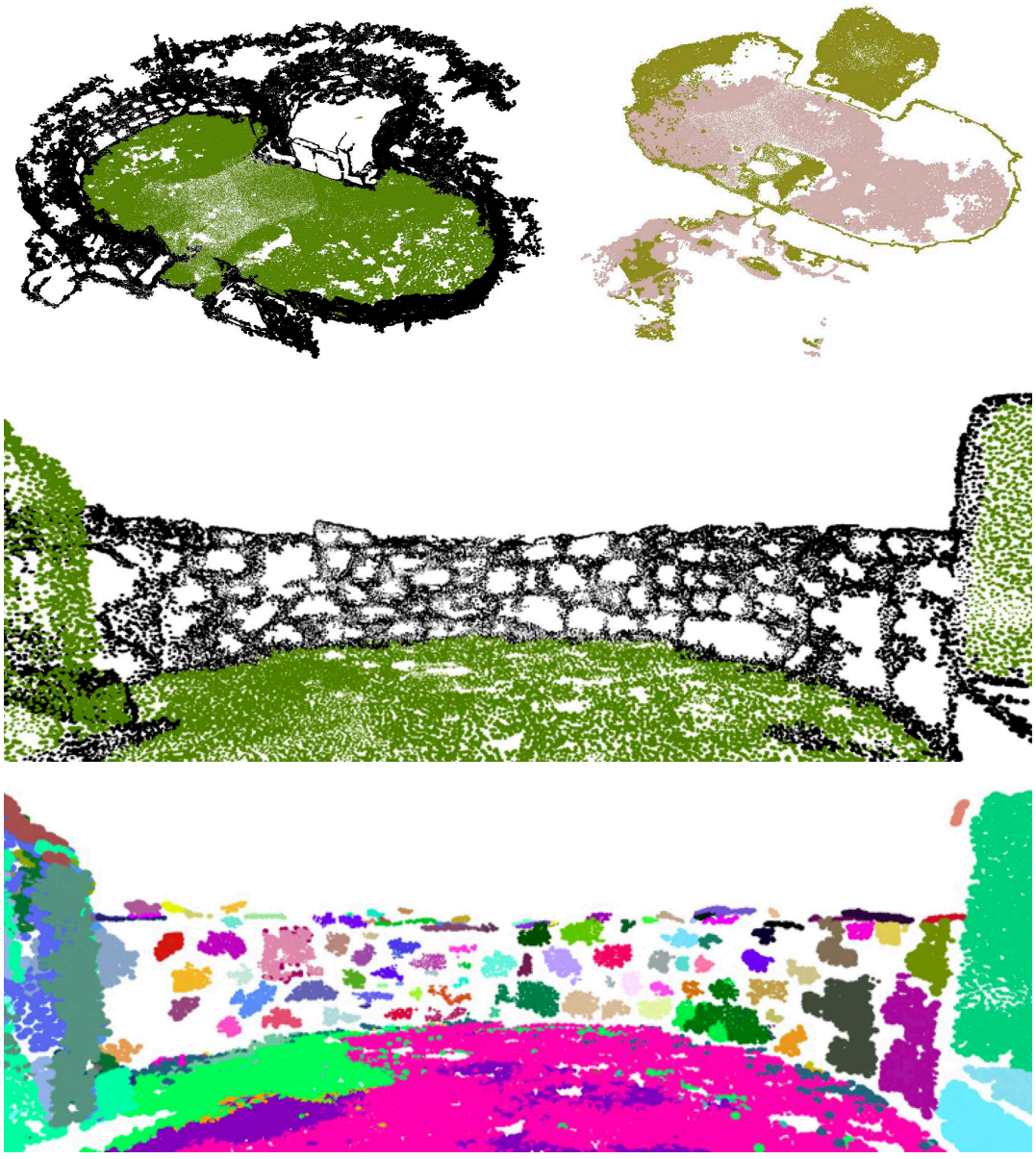


Figure 5.28: Top left images illustrates the largest surface (green) and edge (black) segments resulting from the region growing process. Top right image illustrates the results obtained when RanSac plane fitting is carried out **without** prior region-growing segmentation.

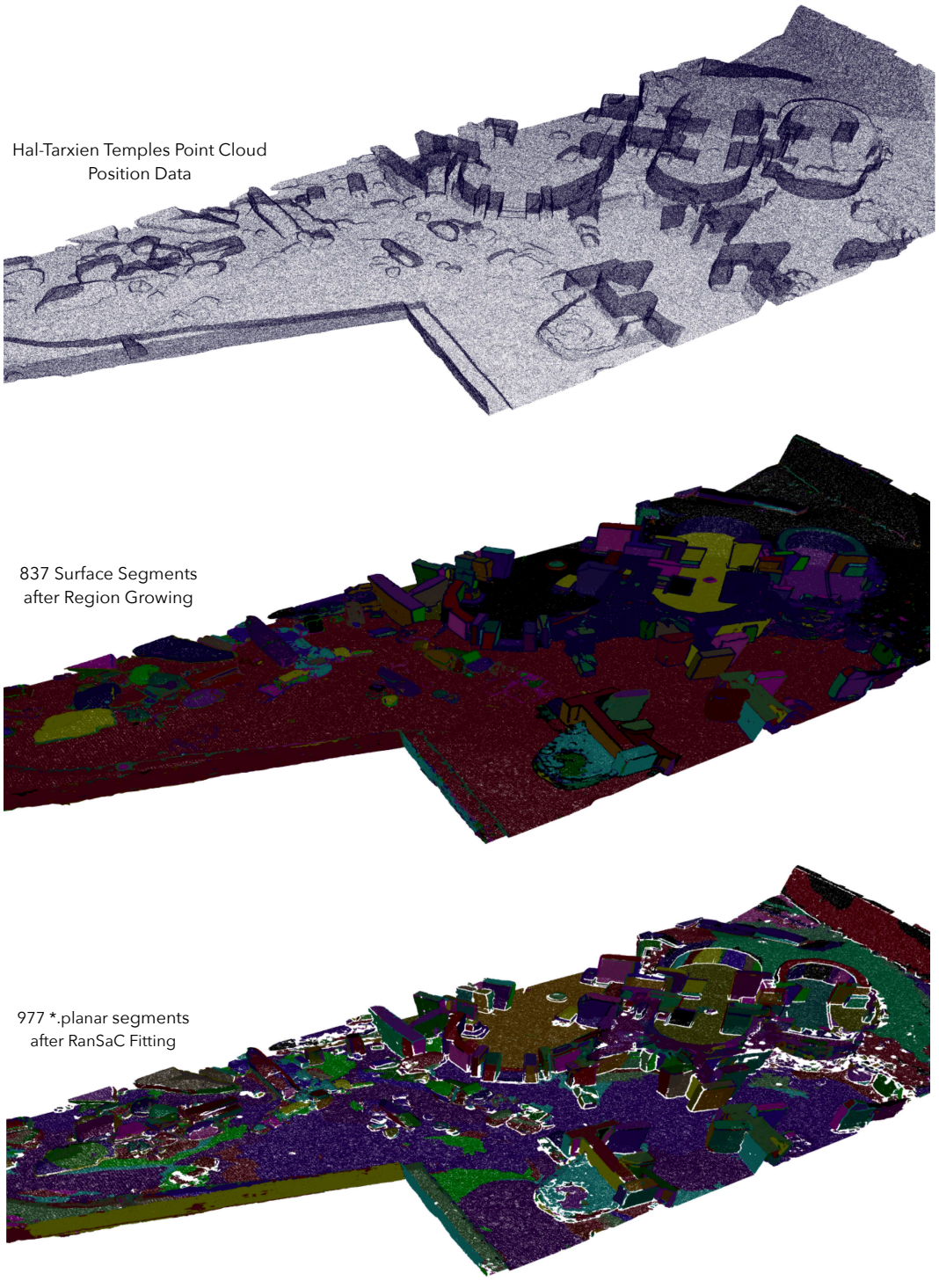


Figure 5.29: Hal-Tarxien point cloud; top illustrates raw data, middle illustrates segments produced after region-growing process, and bottom illustrates segments produced after RanSaC fitting process.

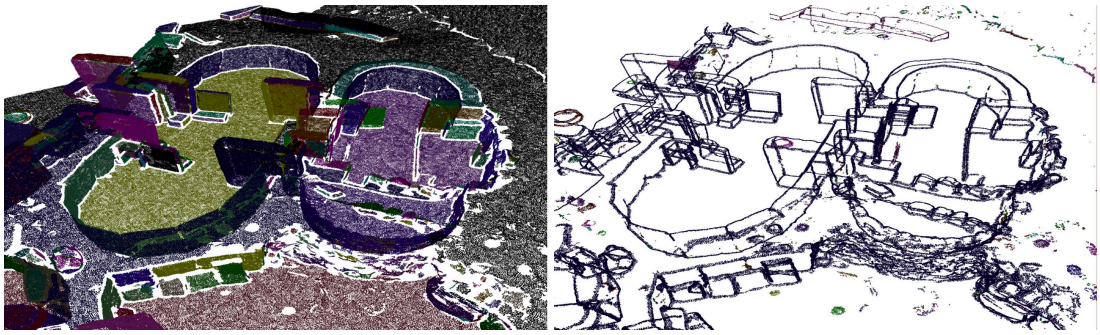


Figure 5.30: The main structures within the Tarxien pre-historic site. Left-hand side image illustrates the surface segments, whereas the right-hand side image shows the edge segments. Edge segments effectively provide an outline of the temples.

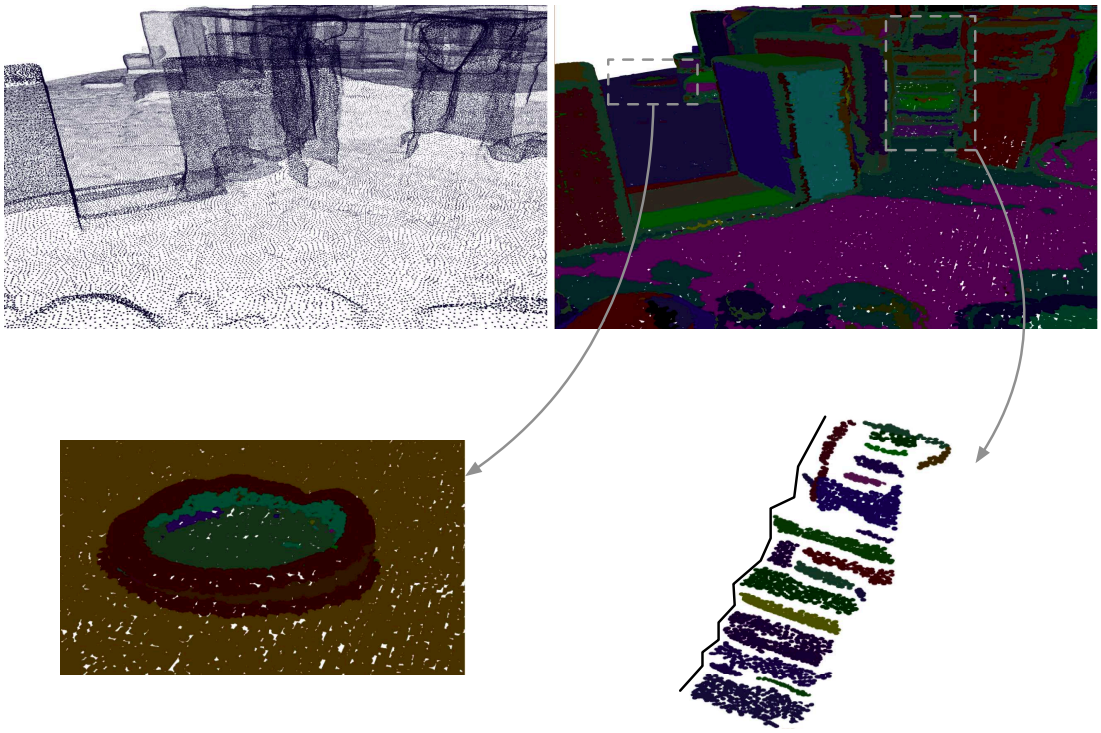


Figure 5.31: Top left illustrates a view of the Tarxien point cloud. This same view is shown on the top right hand side, this time showing the various surface segments produced. The bottom row zooms on two structures in the site. The bowl like structure is shown in the left hand side, with different segments highlighting its shape, whereas the right hand side illustrates segments representing a flight of steps.

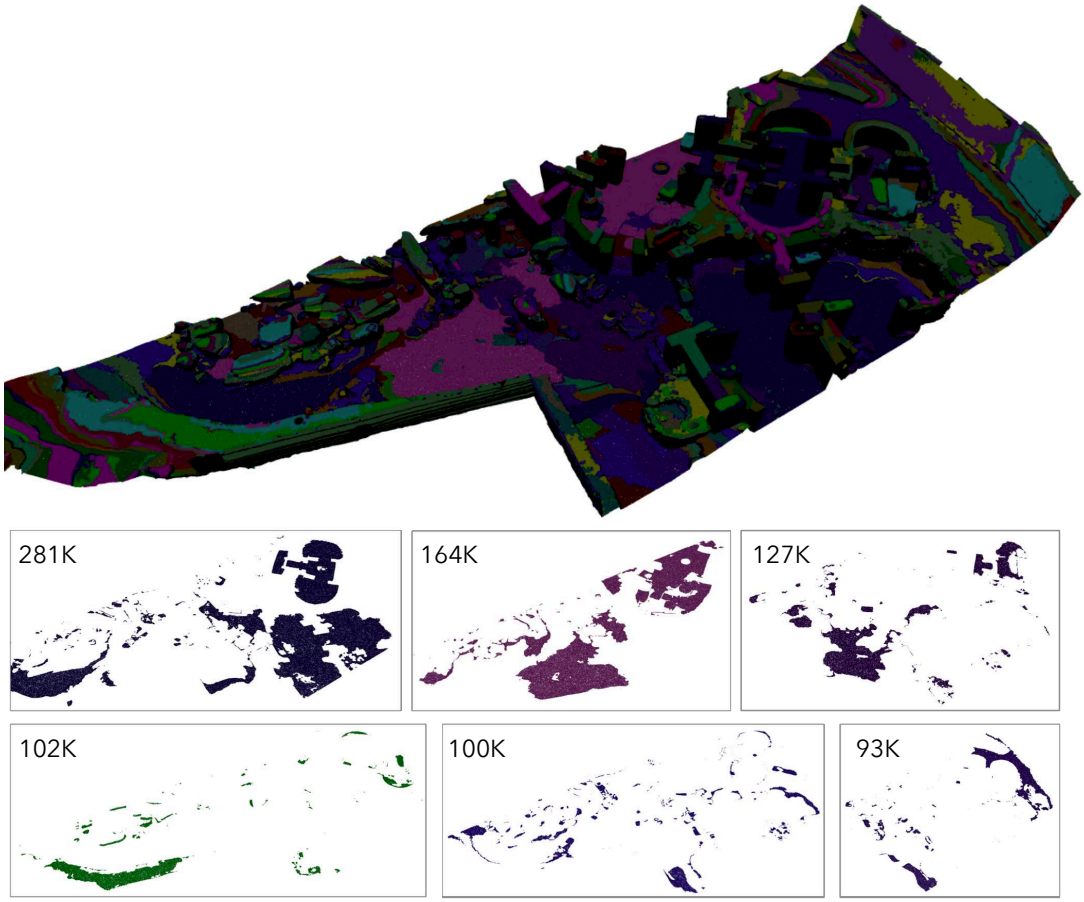


Figure 5.32: Top image illustrates the 148 segments extracted, whereas the second and third rows illustrate, individually, the largest six segments produced using the traditional RanSaC plane fitting process, segments which are clearly not very useful in term of describing the elements making up the site. Numbers within each box indicate the number of points in that specific segment.

website (Falcidieno, 2004) and was scanned with a Leica Cyrax scanner in 2006. The bottom row shows the raw point cloud (left), *edge·** (middle) and *surface·** segments (right). The top row shows a photograph of the church and three *surface·planar* segments extracted from one surface segment produced at the region growing phase. A considerable number of points are labelled as edge, especially on the roof on the church. This is evident in Figure 5.34 with the edge segment outlining the stone slabs making up the roof. This edge segment is useful in extracting individual stone roof slabs as shown in the bottom row. The site contains a number of tombstones directly adjacent to one of the walls. Figure 5.35 shows the extracted segments representing these tombstones. PaRSe creates two segments representing the face and side of one, whereas the other, due to considerable noise is over-segmented, and consists of 9 surface and edge segments. As shown in the same figure, edge segments representing window rails are also automatically extracted. Additional site details are shown in Figure 5.36 including the cross at the top of the church as an *edge* segment. Over-segmentation is clearly visible on some sides of the church tower resulting from a large number of points which are labelled as edge.

5.5.3 University Green Area

A point cloud representing a green area situated within the University of Warwick campus was scanned using a Faro Focus 3D time of flight scanner. Figure 5.38 top left, illustrates this point cloud consisting of a pond, several buildings and a number of trees. In this particular case, the query graph shown in Figure 5.37 is used to extract the trees from the 18 million point data set. Initially, segments of type *edge·complex* are searched for in $\mathcal{G}$. Those matching the constrains between S_0 and S_1 of the query graph are labelled as the canopy of the tree. The segment in $\mathcal{G}$ mapping to S_1, represents a portion of the trunk of the tree. Note that the query graph for cylinders can be composed with this query graph at S_1, in order to check whether the trunk consists of a full cylinder. In this particular case all tree trunks consist of one *surface·planar* segment. If another orthogonal planar segment is connected to the trunk, this is labelled as the ground. This example is used to demonstrate how a variety of post-processing tasks can be encoded as small query graphs using segment types and properties, which are then used on the structure graph produced by the PaRSe. Note that, the tree query graph has only been used on a point cloud which has similar trees and might therefore

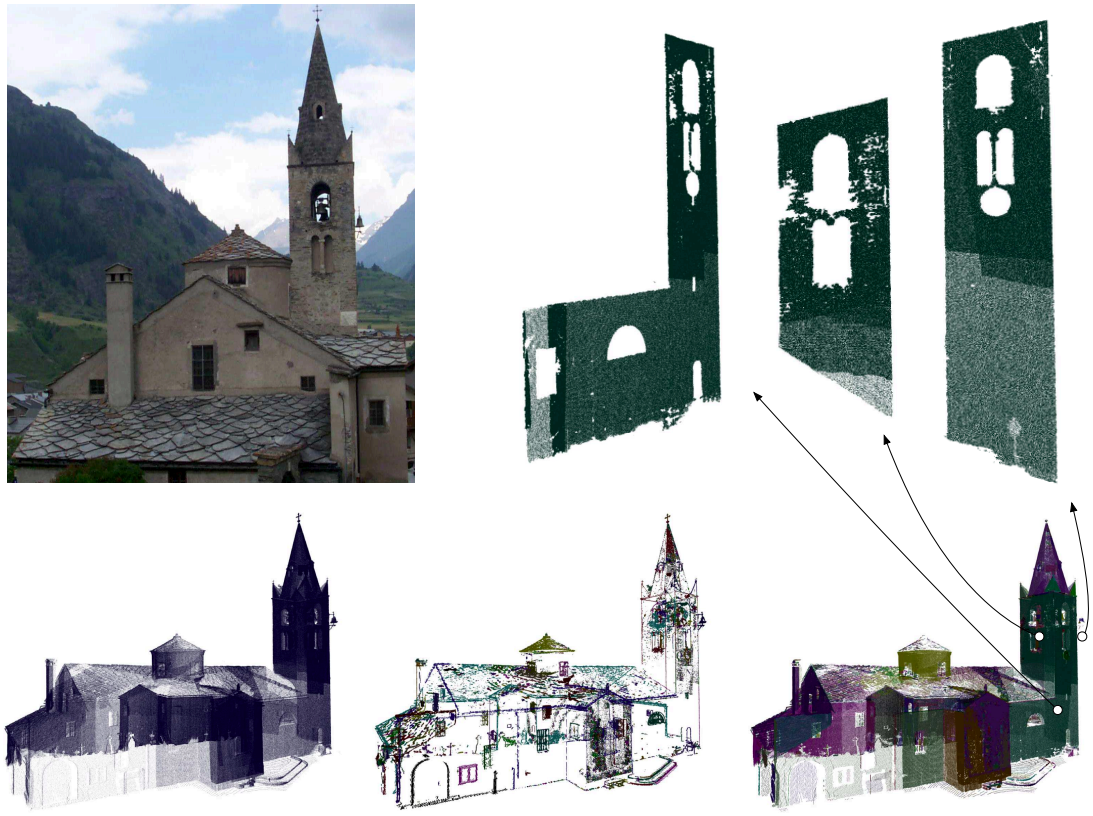


Figure 5.33: Top left illustrates a photograph of the Lans le Villard church. Bottom row shows from left to right, the raw point cloud, $edge \cdot *$ segments and $surface \cdot *$ segments. Top right illustrates three $surface \cdot planar$ segments extracted from one of the surface segments.

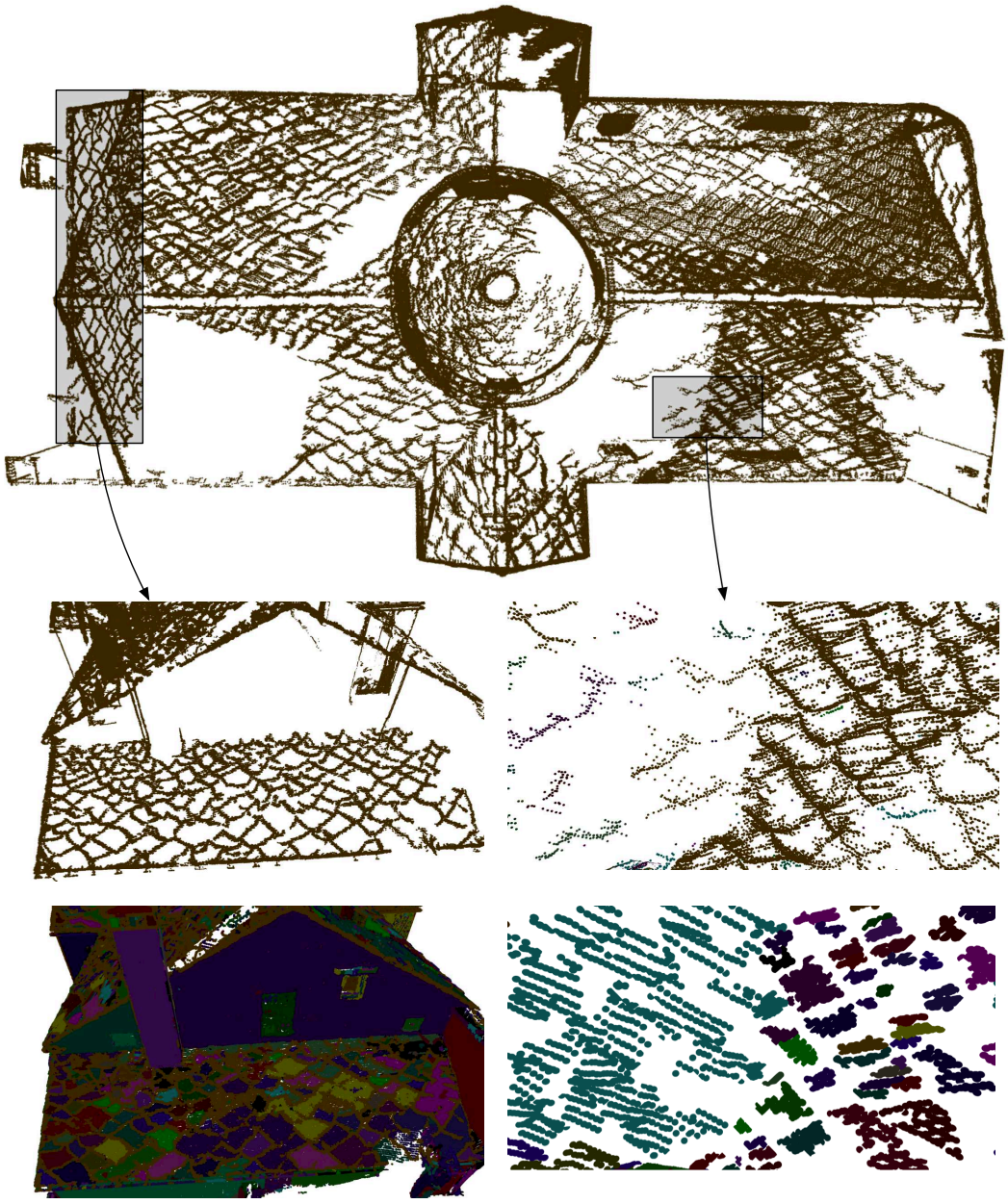


Figure 5.34: Top row illustrates a top-down view of the edge segment outlining the stone slabs making up the roof. This edge segment is useful in extracting individual stone roof slabs, as shown in the bottom row.

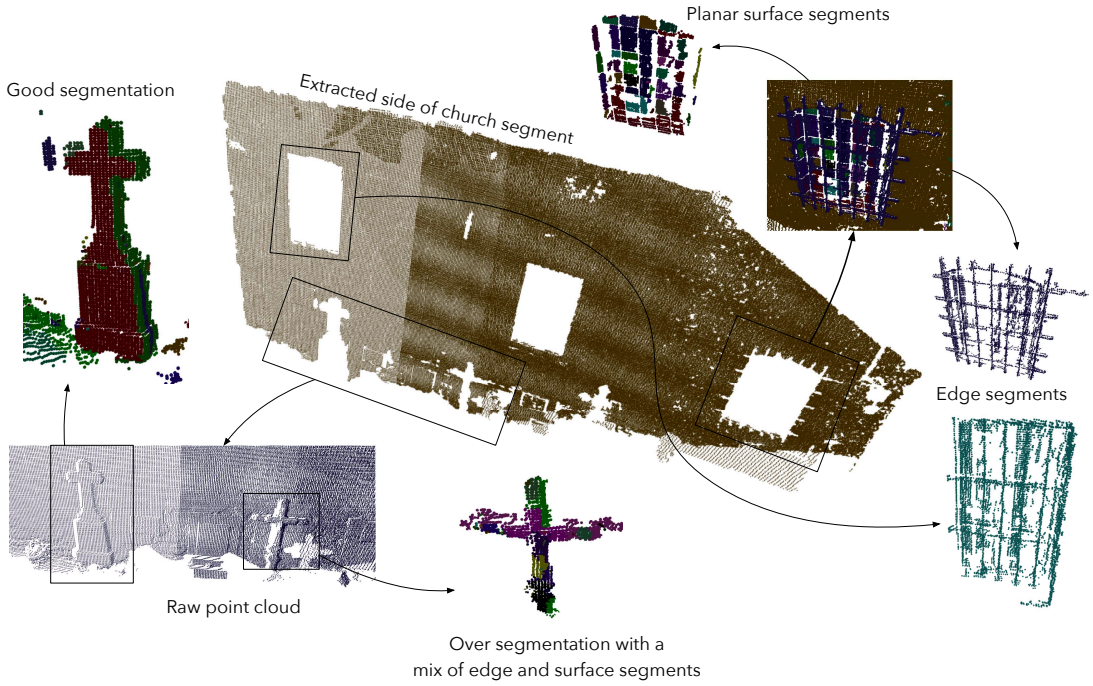


Figure 5.35: Surface and edge segments extracted from one side of the Lans le Villard church, showing tombstones and windows rails.

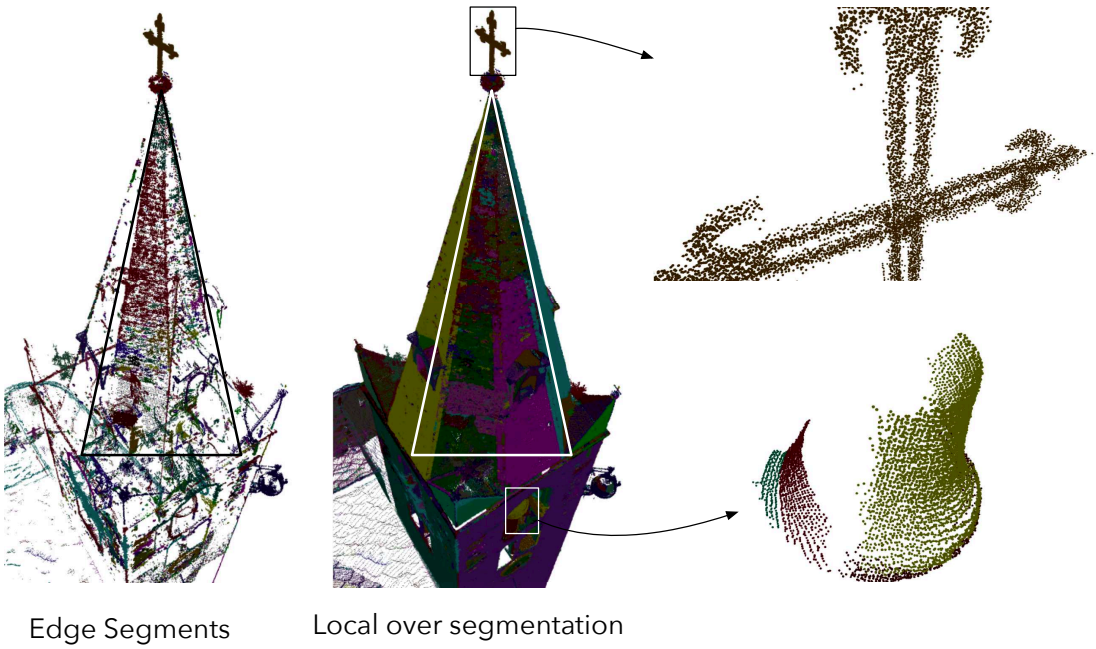


Figure 5.36: Over segmentation resulting from an excess of edge segments. One edge segment represents the cross at the top, whereas two surface and one edge segment represent the bell inside the tower.

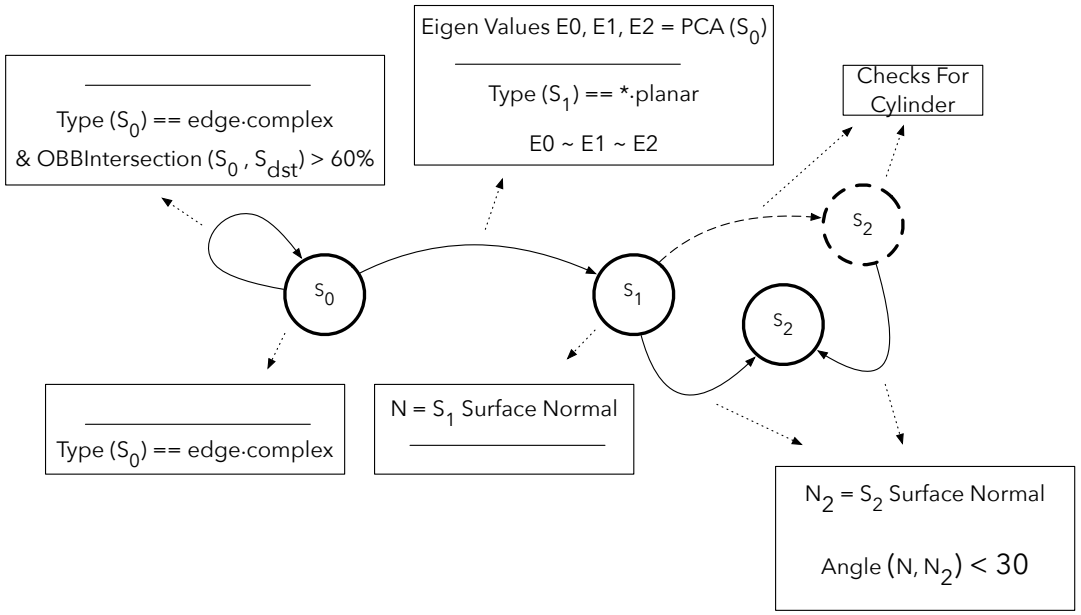


Figure 5.37: A trees query starts off from *edge·complex* segments checking for similar OBB Eigenvalues (effectively discarding flat segments found on buildings) and proceeds if a **·planar* segment representing the trunk is connected to it.

not be sufficient in cases where trees are more varied and complex, especially if a considerable amount of the tree samples including leaf details are available. The second row of Figure 5.38 illustrates the execution of the query, showing first the *edge·complex* segments (tree canopies), following by *surface·planar* segments (tree trunks), and the segments directly under these trunks.

5.5.4 Indoor Office Scenes

Point clouds of two office scenes from Mattausch *et al.* (2014) (approx. 10 million points each) are used in this section. Figure 5.43 illustrates the first of these point clouds highlighting segmentation results, with PaRSe automatically extracting a bin on the ground, heating elements from the walls and a variety of desktop items including webcam, monitors, keyboard and speakers from a desk. The application of RanSaC plane fitting, without prior region growing (bottom row) produces segments spanning parts of multiple objects. The second office point cloud, Figure 5.39, illustrates some limitations of PaRSe. The bottom row shows the raw, edge and surface segments point clouds of the shelving unit. Whereas edge points correctly delineate most of the unit, a number of problems occur

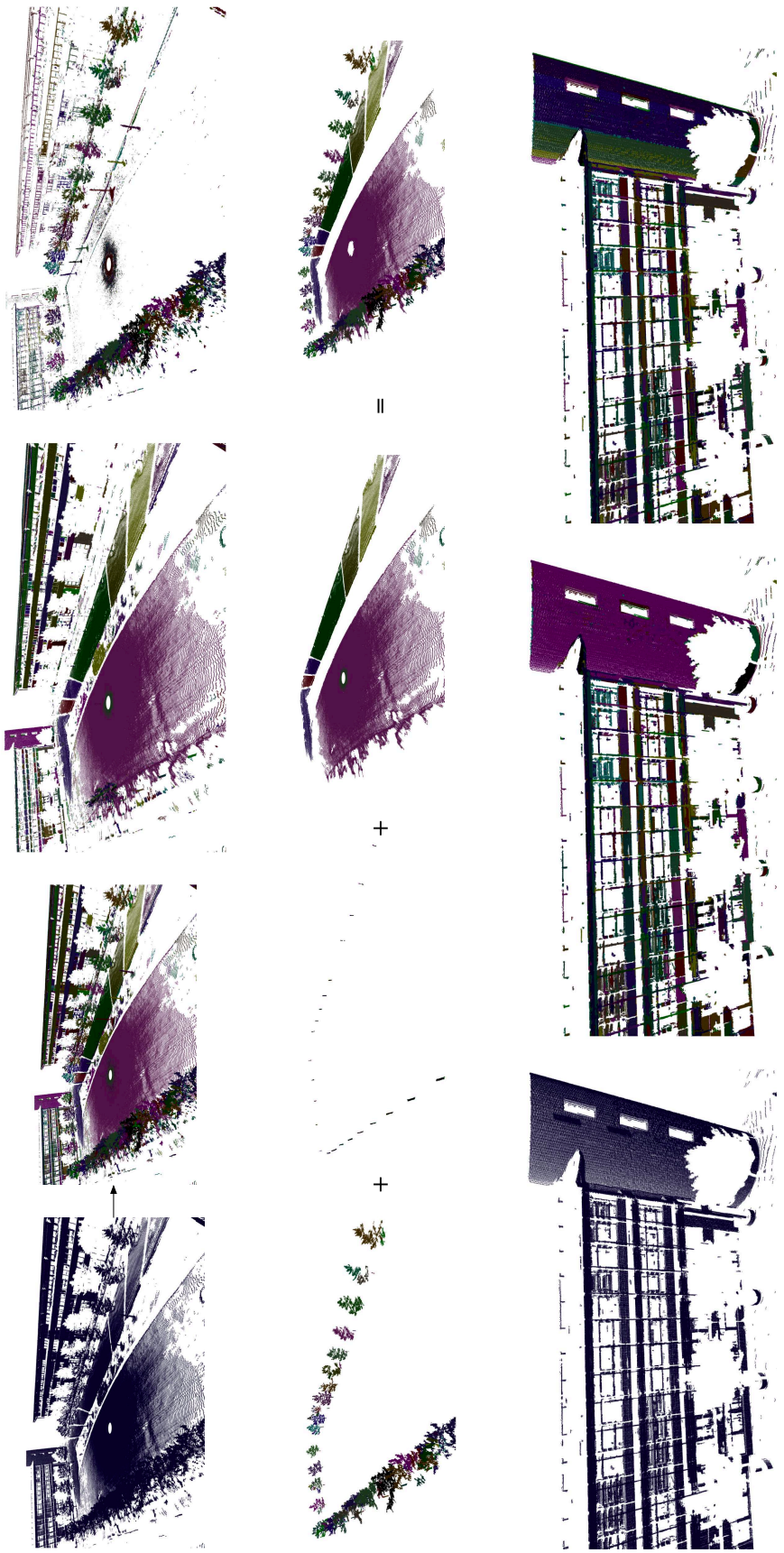


Figure 5.38: Top row from left to right illustrates raw point cloud of green area, segments produced after region growing, surface segments and finally edge segments which include tree canopies. The second row illustrates from left to right, the edge segments with similar Eigen values representing individual tree canopies, planar segment connected to each and the large surface segments connected to the trunks representing terrain. The third row focuses on the extraction of a cylinder primitive which represents part of the building.

on the box on the ground (marked with a dotted box) in that some points are incorrectly labelled as edge points. This happens because part (black triangle) of one side of the box is sampled from both sides (two scans), with the thickness of the box sufficiently small to include points from both sampled sides when applying k-NN searches during point labelling. The second row shows how the lamp and wiring on the desk are extracted as one *edge·complex* segment, whereas the top right image shows over-segmentation occurring between two desks.

5.5.5 Airborne LiDaR of Maltese Archipelago

A point cloud representing a section of the Maltese archipelago acquired using airborne LiDaR scanners with around 4 samples per m^2 is used in this section. The data was made available by the institute for climate change and sustainable development at the University of Malta. Figure 5.44 illustrates segmentation results, with the top left representing the raw position only data and the top left representing all generated segments. The bottom row illustrates the *edge·** and *surface·** segments on the left and right hand side respectively. The point cloud contains a variety of geographic features, including urban areas, trees and fields. Edge segments are more concentrated in the urban area, whereas the majority of large surface segments are located in the fields to the north of the point cloud. The rubble walls which are used to divide these fields are labelled as edge points and effectively contribute towards delineating a considerable number of fields. Figure 5.44 shows a section of the point cloud showing the segmentation of agricultural fields. Edge points in the urban section of the point cloud are useful to distinguish between the different non-regular housing units. Note that in the majority of cases, only a few points (around 30) are sampled from each house roof as shown in Figure 5.40 second row. The first row of the same figure illustrates a *surface·planar* segment representing a football ground, and two *edge* segments representing the goal posts (circled). The small segments on the third row, represent trees which are all connected to the same *surface·complex* (rendered in green in Figure 5.44, bottom right). Another important $surface$ segment is the one representing the road network, which extends from the urban to the agricultural sections and is shown in Figure 5.41.

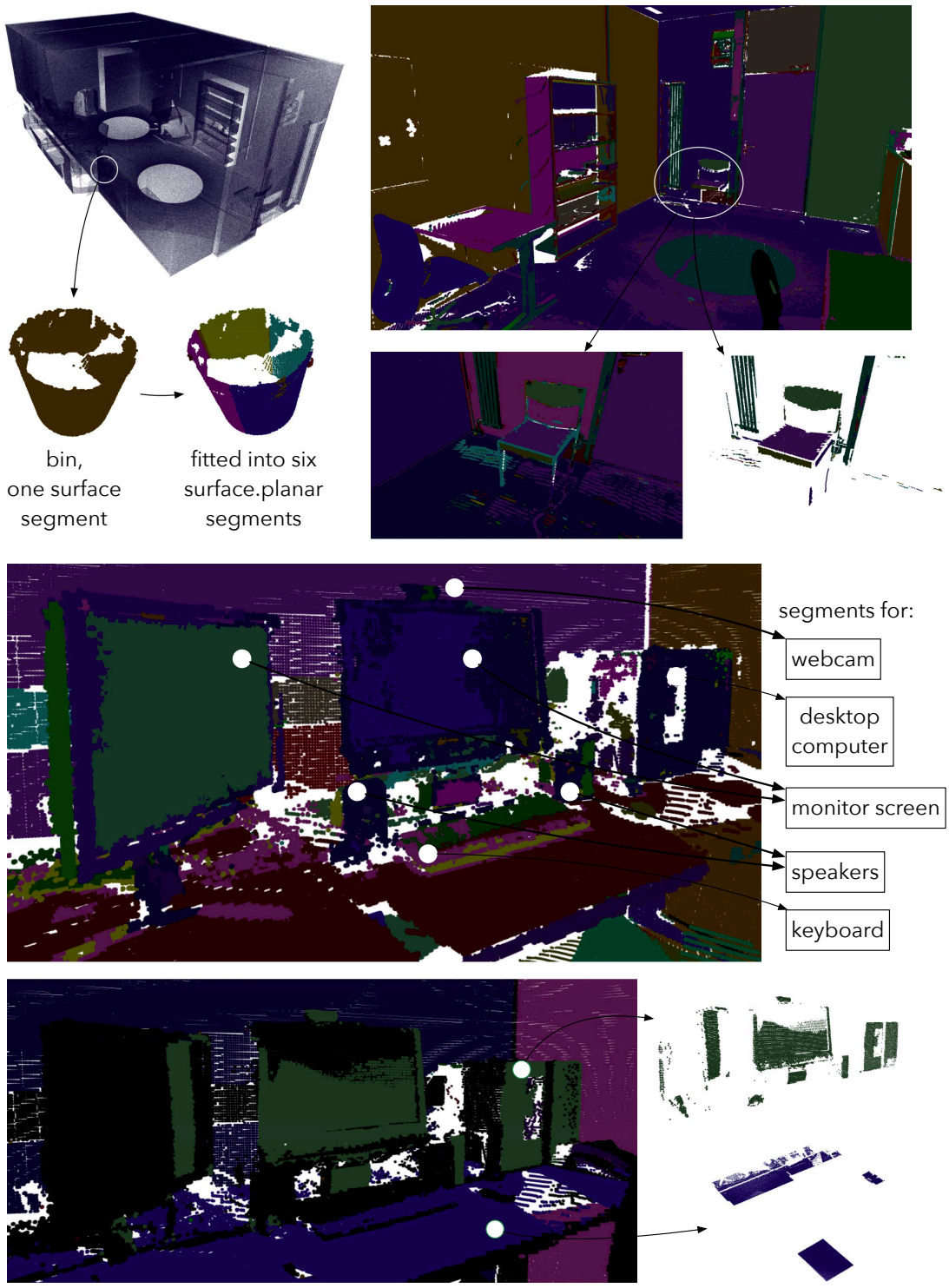


Figure 5.39: PaRSe applied on first office scene, extracts bin on the ground, heating elements from the walls and a variety of desktop items including webcam, monitors, keyboard and speakers from a desk. RanSaC without region growing (bottom row) produces segments spanning parts of multiple objects.

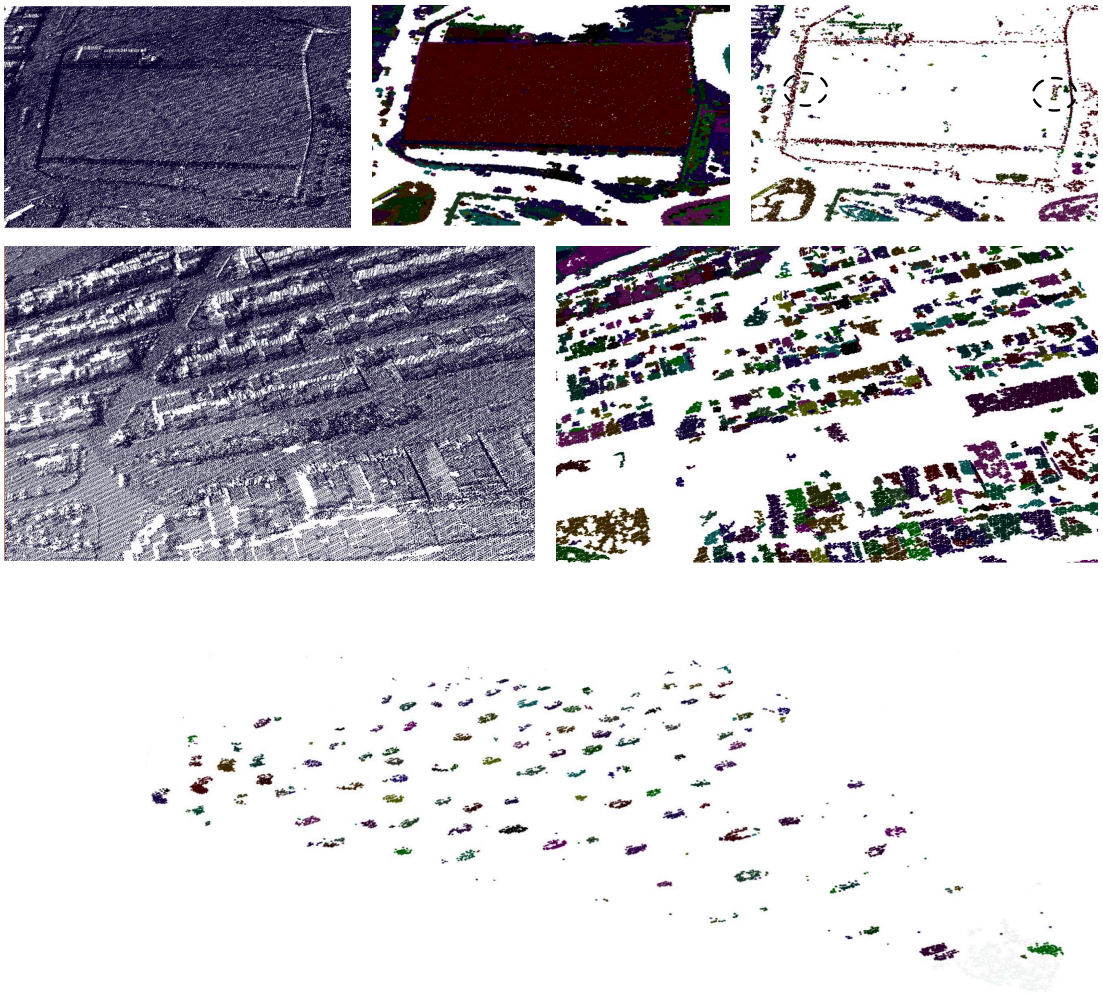


Figure 5.40: Top row illustrates from left to right, football ground structure raw points, the segment representing the pitch, and the boundary of the pitch. Note how some samples are acquired from the goal posts (circled) which are visible as two edge segments. The middle row illustrates a close-up view of the urban area, showing the extracted individual housing structures. On average each house is represented by around 100 points. The third row, illustrates the edge segments representing trees in the bottom part of the point cloud shown in Figure 5.44

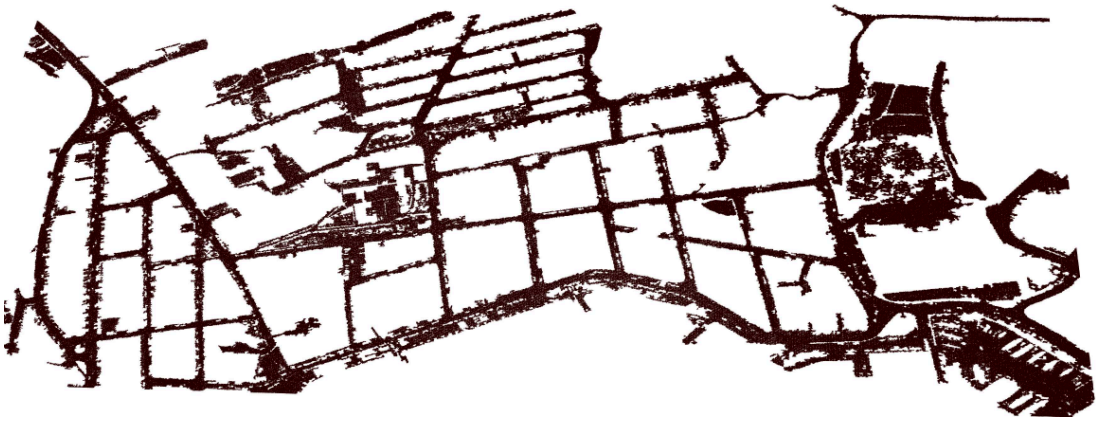


Figure 5.41: The biggest surface segment represent the main road network in the point cloud. Note how this segment spans different terrain elevations. A simple RanSaC plane fitting approach would not have been able to discriminate between points on the road and others within house complexes.

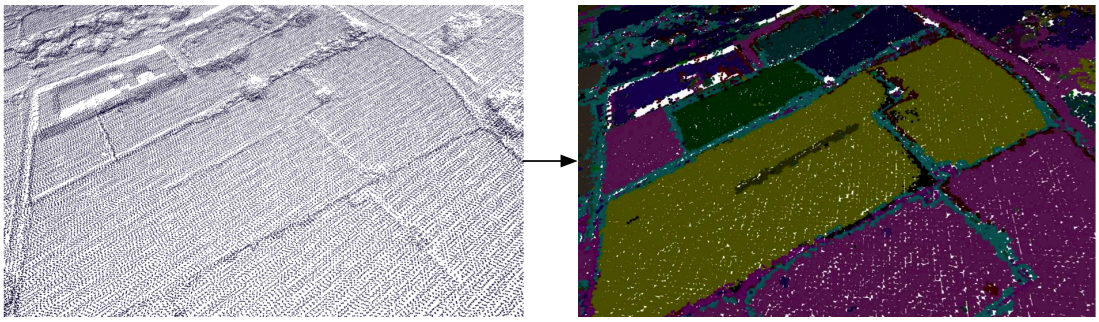


Figure 5.42: Small presence of rubble walls between fields are sufficient for the segmentation algorithm to distinguish between fields which are rendered using different colours.

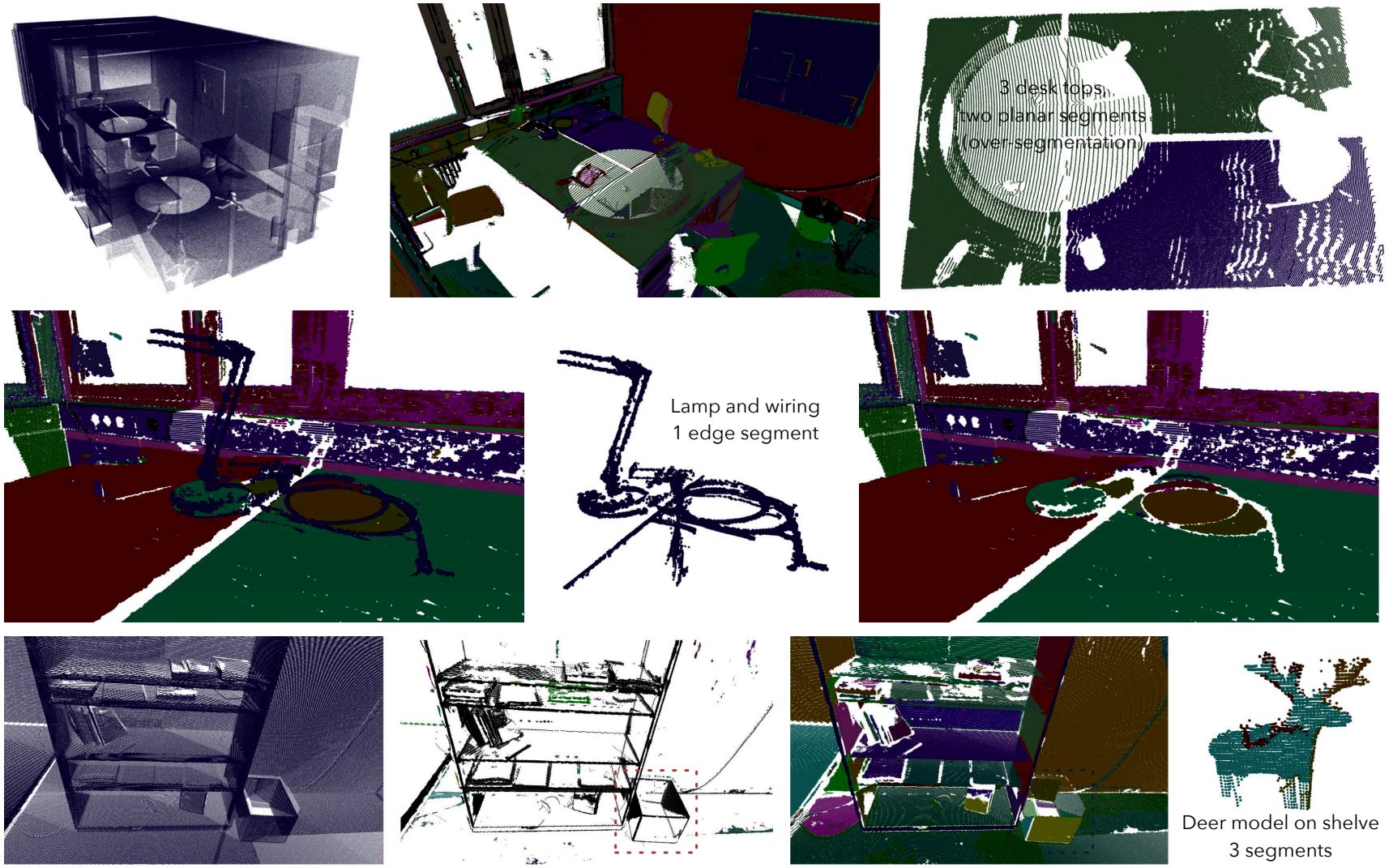


Figure 5.43: PaRSe applied on second office scene, extracts desk lamp and wiring from a desktop, and multiple books and files from a shelving unit. Over-segmentation occurs when producing segments for desktops due to points falling between the desks (top left). Note that two additional surface segments are produced on the desk near the lamp wiring. A deer model located on one of the shelves is extracted as one surface segment and 2 edge segments.

Figure 5.44: Segmentation results for a LiDaR data set representing a portion of the Maltese Archipelago, consisting of both urban and agricultural terrain. Top left-hand side illustrates raw point cloud. The overlap (increased sample density) between successive aerial scans is clearly visible. Top right-hand side shows all segments produces by our segmentation process, whereas the bottom row left illustrates the edge segments and right illustrates the surface segments.

5.6 Discussion

The generation of 3D point clouds has become increasingly common in many areas of research. Given this huge amount of data, algorithms are required which are able to process, organise, cluster and extract important information about it in order to help in the post-processing effort. Our results have shown that the proposed segmentation method, PaRSe, is a feasible approach towards achieving this goal. For each of the point clouds used, the segmentation primitives proposed in this chapter have been able to represent some meaningful structure. In the case of the Mnajdra temple (§5.5.2), segmentation produces segments containing surface points from individual stones on the apse. An *edge·complex* segment is extracted which represents the contour of these stones. Given the complex nature of the point cloud, standard region-growing and shape fitting algorithms are not able to produce these segments. The Hal-Tarxien case study shows how small details in the site are identified in the segmentation process and represented using the segment primitives used (Figure 5.31). A graph query is used to identify segment patterns representing trees in the point cloud acquired at a University of Warwick green area. In this case, *edge·complex* segments are used to determine the location of trees. Segments resulting from the segmentation of a LiDaR point cloud include meaningful objects such as trees, houses, fields and streets. Similarly, on a smaller scale of two office environments, PaRSe computes a set partition whose elements represent a variety of objects which are typically found in an office including computer desktop, monitors, shelving, chairs, tables and other small objects. The automatic partitioning of the input point cloud into meaningful smaller segments helps in reducing the post-processing effort required to process the raw data.

With segmentation algorithms using RanSaC shape fitting, as the number of iterations/trials computed is limited, the solution (fitted planes in our case) obtained may not be optimal (Figure 5.28) and it may not even be one that fits the data in a good way. This is shown for a number of examples. In our case, since RanSaC fitting is done over segments which are previously established from a region-growing process and not the entire point cloud, this whole process is much more efficient (lines 13 and 14 of Algorithm 8) and the solution obtained for a complex site such as the Mnajdra temple is repeatable, i.e. given a number of runs of PaRSe and that the same input parameters are used, the same (or very similar) set partitions are produced fitting the raw data. Moreover, the

structure afforded by this repeatability, results in a higher level of abstraction over the data, which allows for the creation of query graphs which can be used to efficiently select different parts of a point cloud. For instance, all the trees in the point cloud of the Warwick university green area are segmented in a very similar way, which enables the query graph to identify all the trees.

The segmentation pipeline presented is efficient both in terms of memory and time complexities with data access patterns favouring parallelization. Region-growing may proceed concurrently (depending on the number of CPU cores) to produce a set partition of surface and edge segments. These segments are then analysed concurrently with the purpose of fitting plane primitives within them. An alternative task subdivision approach, since region-growing is generally less time consuming, is to adopt a consumer-producer scenario where region grown segments from a single thread are inserted in a pool for the rest of the threads to apply RanSaC plane fitting concurrently.

5.6.1 Further Applications

In this section further applications of PaRSe are proposed.

Texture mapping Aligning textures with geometry is an important post processing task especially if no colour information is available within the data. In this case, the ability of partitioning the point cloud into small meaningful segments can be used to find correspondences between photographs of the site and the specific parts of the point cloud.

Adding semantics CH experts would usually require that specific parts of the point cloud are tagged with some specific information. For example, one might label a particular segment of the point cloud as representing the ground, which is then linked to photographs from the site. A GUI would be required to allow a used to browse the point cloud structure graph and attach additional data to nodes.

Tessellation Segmentation results have the potential of being used to improve tessellation algorithms and rendering quality. For instance, automatically creating a reasonably accurate mesh of the Mnajdra temple is not a straightforward task. Segmentation results can be used to project the *$*\cdot planar$* segments extracted onto a flat surface, tessellate using traditional Delaunay

triangulation (Lee & Schachter, 1980), then use the topological information acquired to render the quasi-flat surface as a triangular mesh. Clearly, a mechanism to connect the various meshes would then be required.

Primitive shape fitting using **.planar* segments When additional shape fitting is required, rather than applying RanSaC on the entire point cloud, it would be interesting to investigate the outcome of choosing shape support points from the different $*.planar$ segments.

Reconstructing Symmetry The graphs produced by segmentation can be used to identify regions of symmetry, which could then be used for reconstruction purposes in areas which are not sampled.

GUI for structure graph navigation and visualisation A user-friendly GUI to enable interactive editing of the structure graph would give users with a variety of technical backgrounds the necessary tool to improve the manipulation/editing of point cloud data. Since the number of states in these graphs can be substantial, the visual presentation of the graph requires further work to be done.

GUI for query graph creation and composition Further work is being carried out on formalising a point cloud query language to automatically synthesise query graphs. In this regard, a user friendly GUI which can be used to create, execute and view results of these queries is required.

5.7 Summary

This chapter presented PaRSe, a novel general-purpose segmentation method for raw point clouds which enables easier manipulation of these data in a variety of tasks. The results show that the approach, which combines region-growing and RanSaC plane fitting, results in the generation of segments which describe meaningful parts of a point cloud. The structure graph produced during the generation of these primitive segments is then used to further facilitate the management of related segments by providing a mechanism for composing them into larger entities. Whereas segmentation is context-free, the grouping of segments is user-driven and depends on the specific context. An important direction in the further development of PaRSe is the provision of a user-friendly and efficient

GUI by which professionals in various fields can select interesting parts within a point cloud. This includes the design of a graph query composer, which automatically parses a visual description of the query graph and produces code which implements the required functionality.

The acquisition of larger areas at higher densities results in gigabytes of point cloud data. These data sets may not fit entirely in main memory, making their processing impractical. In particular, the essential k-NN operation which is carried out multiple times during segmentation, requires that all points and the associated kd-tree acceleration structure are loaded in main memory. The next chapter presents a novel extension to PaRSe, that allows processing of massive data sets on devices with minimal primary memory.

CHAPTER 6

Fast Scalable k-NN Searches for Very Large Point Clouds

The process of reconstructing virtual representations of large real-world sites is traditionally carried out through the use of time-of-flight laser scanning technology (§2.3.2). Recent advances in these technologies has led to improvements in both sample quality and speed of acquisition resulting in scanners capable of considerably higher sampling rates. The raw data resulting from the acquisition process usually needs to be processed in order for important topological information to be extracted. For instance, in the case of robot navigation, this processing might be required in order to determine the location of a particular object in the environment and guide the robot around it. In many cases, these large point clouds require cleaning through the application of numerous post-processing algorithms, for instance normal determination, clustering and noise removal. A common factor in these algorithms is the recurring need for the computation of point neighbourhoods, usually by applying algorithms to compute the k-nearest neighbours (k-NN) of each point (§2.5.1). PaRSe, the segmentation method presented in Chapter 5, carries out k-NN to first determine the type of each point and then the segments during the region growing process.

In some cases, the size of the data set acquired is so large that it does not entirely fit in main memory. This is particularly true of outdoor cultural heritage sites (e.g in Ruther (2010b)) acquired using professional grade 3D scanners capable of generating highly accurate data at sampling rates of close to a million points per second (Elseberg *et al.*, 2011). The majority of post-processing algorithms work under the assumption that the data sets operated on can fit in main memory, while others take into account the size of the data sets and are thus

designed to keep data on disk. When the size of the point cloud is very large, a considerable amount of time is spent searching for the neighbours of each point. Moreover, many of these post-processing operations (e.g. noise removal) may be applied on the same data set more than once using different input parameters. This is especially true in the case of a number of segmentation algorithms where differing input parameters may produce widely varying results. Even if these operations are usually carried out offline, execution time is still an important factor to take in consideration. Optimal performance results are achieved when the k-NN computation is carried out in-core, i.e. when both points and acceleration structure are stored in main memory. On the other hand out-of-core techniques take into account the size of the points but are much slower due to overheads related to disk I/O. PaRSe as presented in Chapter 5, loads all points in main memory and assumes that k-NN computations can be done in-core. The method presented in this chapter addresses this limitation.

The development of algorithms for the efficient determination of the k-NN of points in a point cloud has been an active area of research for many years (Clarkson, 1983; Vaidya, 1989; Sankaranarayanan *et al.*, 2007). In most cases memory-based space subdivision data structures are used to help quickly determine neighbouring points. One such acceleration structure is the kd-tree (§2.4) which is used in many prominent libraries such as Muja & Lowe (2009a) and Rusu & Cousins (2011), to provide a spatial subdivision over the input point cloud. Search algorithms, mostly based on either depth-first (DFS) or best-first (BFS) traversals are then used to efficiently determine neighbours (Algorithm 1). These search algorithms can either compute the exact nearest neighbours or else the approximate nearest neighbours (ANN). In the case of ANN, an error threshold ϵ is used to speed up the computation of neighbours at the expense of correctness. Significant speed-up can be achieved when the data set consists of higher dimensional data points Arya *et al.* (1998). In the case of 3D scanned point cloud data, the difference in performance between ANN and k-NN is minimal. In our approach both approximate and exact nearest neighbours can be determined, but since there is only a minimal difference, in the results presented here only the exact k-NN are computed.

An important consideration which is addressed by Sankaranarayanan *et al.* (2007), is the size of these point clouds. As size increases, search algorithms based on in-core data structures, such as kd-trees, are limited by the amount of memory present in the computer on which they are deployed. Sankaranarayanan *et al.*

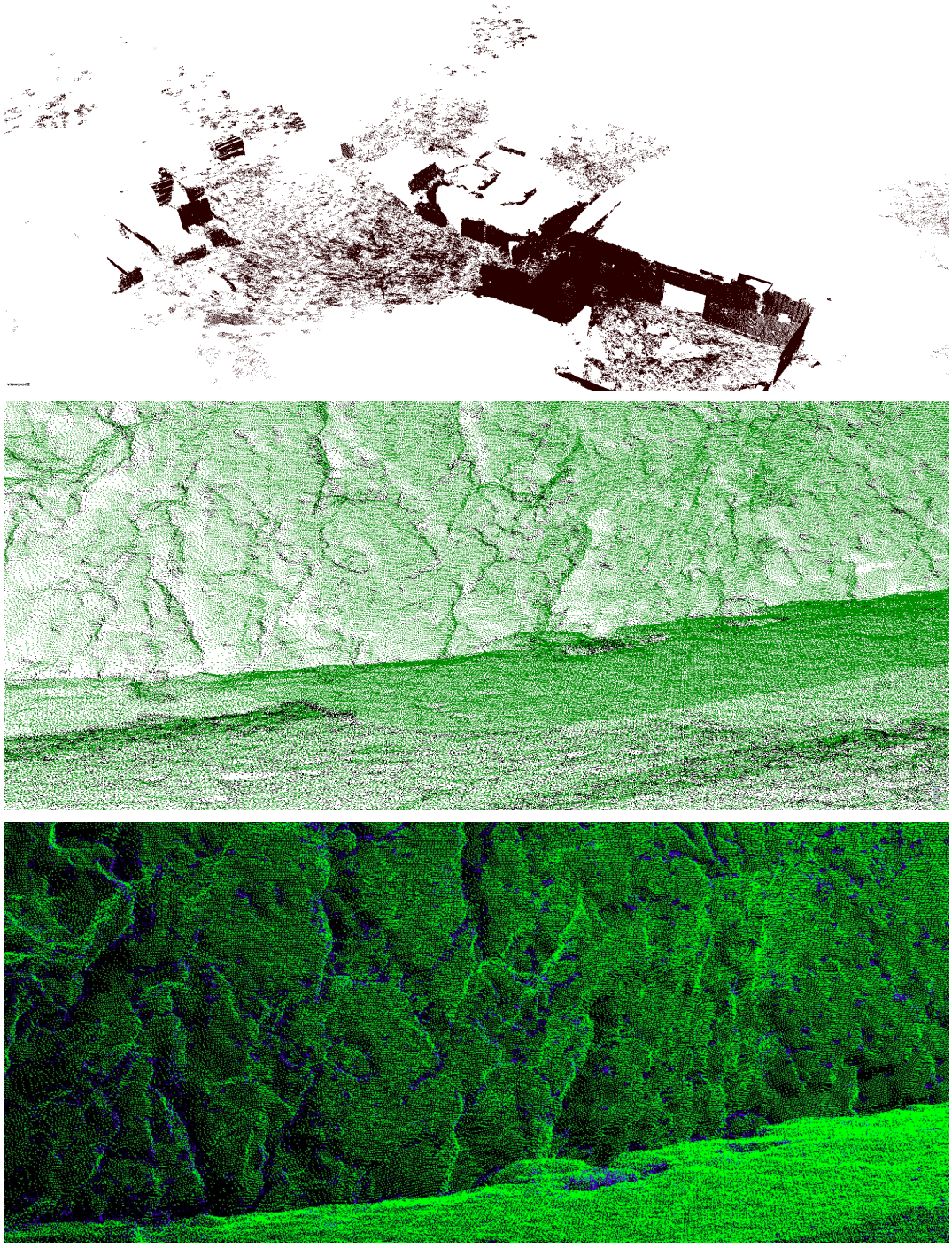


Figure 6.1: Top row image shows the entire Songo Mnara point cloud (Ruther, 2010a) consisting of 45 million points. Middle row illustrates one of the walls with point types assigned and the bottom row image illustrates the same wall rendered using higher contrast colours to enhance the details on the wall.

(2007) describe an all nearest neighbour algorithm for applications involving large point clouds. Their method makes use of disk-based out-of-core data structures and is thus not limited by the amount of system memory available. They first determine localities, for blocks of points, which are then used to decrease the range of candidate neighbour points to search. Even though their algorithm is designed to work with multi-dimensional data sets, evaluation is carried out only on 3D point clouds and report significant improvements over previous methods with respect to the time it takes to compute k-NN. For example, when using a data set of 50 million points, 7999 neighbourhood/s are computed on a machine with 1GB of system memory.

In this chapter, PaRSe is extended with a novel external memory algorithm using a hybrid of spatial subdivision techniques for out-of-core fast k-NN searches on point cloud data. This hybrid approach exploits the spatial locality of point clusters in the point cloud and loads them in system memory on demand by taking advantage of paged virtual memory in modern operating systems. In this way, processor utilization is maximised while keeping I/O overheads to a minimum. This approach is evaluated on point cloud sizes ranging from 50K to 333M points on machines with 1GB, 2GB, 4GB and 8GB of system memory, taking advantage of all system memory available but never exceeds it. On a 1GB machine with similar specifications to the tests carried out by Sankaranarayanan *et al.* (2007), PaRSe extended with out-of-core capabilities achieves approximately 100,000 neighbourhoods/s using a data set of 166 million points.

6.1 Data structures for out-of-core processing

In order to design a fast k-NN computation procedure, the extension to PaRSe takes advantage of two important concepts, namely, spatial subdivision and memory mapped files. The first is used to reduce the time complexity of the nearest neighbour algorithm, whilst the second is used to maximise the use of available memory.

6.1.1 Spatial Subdivision

Regular grids subdivide 3-space into regions of equal volume where each region can be uniquely addressed by an index (i, j, k). If the regions operated on are known, one doesn't need to be concerned with the whole grid, but can concen-

trate instead on the said regions. The straightforward subdivision afforded by regular grids allows us to maximize memory utilization by loading in core only the affected regions. The point clouds used are not uniformly distributed in 3-space and partitioning these data sets into regular grids yields a large number of empty regions. Thus, the regular grid is implemented as a sparse map storing only the regions which contain interesting information. The time complexity for lookup and insertion of a region, or cell, is in both cases O(log n), since the sparse grid is implemented using red-black trees (Bayer, 1972). A lookup for the nearest-neighbour of a point within a region runs in linear time; kd-trees are then used to store points within a cell, reducing the lookup complexity to logarithmic time in the number of elements (Friedman *et al.*, 1977).

6.1.2 Memory Mapped Files

Virtual memory (Denning, 1970) is a memory management technique which allows the execution of processes not entirely held in memory by separating the user view of memory from the actual physical memory and provides a mapping function from one to the other. Implementations for virtual memory require hardware support, typically provided by a memory management unit built into the CPU. Paged virtual memory is an implementation of a virtual memory system which divides the logical address space into equal sized memory blocks called pages, permitting the use of memory mapped files (MMF), wherein a file can be manipulated as part of the process address space. This is accomplished by mapping disk blocks to pages in memory using the virtual memory system. Access to memory mapped files uses a demand paging scheme, whereby a block is loaded in memory only if it is needed. The first time a block is accessed, a page fault is generated, and the respective block brought to memory. Subsequent accesses to the specific block occur as memory reads or writes, avoiding the overhead of read and write system calls. Moreover, files which do not fit in memory can still be manipulated with relative ease, as the paged virtual memory system, swaps blocks in and out as required.

6.2 Concurrent k-NN searches using MMF

The method presented addresses the problem of efficiently searching for the k-NN of all points in a point cloud P, when the size of P does not fit entirely

in main memory. In order to decrease the memory requirements of the process computing k-NN, all point information is stored on disk and iteratively load only those regions in the file which are required in the k-NN computation for a subset of points in P. Points are loaded in memory through the use of MMFs. In general, all memory available on a machine is used to achieve the best possible performance, however in order to mitigate I/O problems which could result from having a process using all system memory, a parameter M is used to indicate an approximate upper bound on the number of points which can simultaneously be present in system main memory. Decreasing the value of M will decrease the memory footprint of the entire process. Whenever PaRSe wants to use all system memory available, M is set to a value larger than the number of points in P. In order to speed up the time it takes to compute k-NN for each point, all processing elements (PE) available on multi-core computers are utilised.

Algorithm 10 describes the high level structure of our approach. The process starts by first creating and populating a uniform sparse grid G with a count representing the number of points in P which fall within each axis aligned cell in G. This is done by iterating once over all the points in P. Using this information, separate files are created each storing a cell ordered subset of points. Once these clusters of points (stored on disk) are created, they are iteratively loaded in main memory and k-NN is performed for points in these clusters.

Algorithm 10 High-level description of process which searches for the k-NN of all points $p_i \in P$

```
1: Input Point-cloud P, M, k.
2: Load Create sparse grid G storing counts for each cell.
3: Sort Partition P. Persist to disk ordered clusters OC_n.
4: for each cluster OC_j do
5:     Memory map points cluster OC_j to main memory
6:     for each non-ghost grid cell C_k present in OC_j do
7:         Create local kd-tree
8:         for each point p_i in C_k do
9:             Compute k-NN
10:            Perform operation on p_i using neighbours
11:        end for
12:    end for
13: end for
```

The following sections describe in more detail the stages *load*, *sort* and *compute*. The first stage reads a point cloud binary file and determines spatial lo-

cality for all points. This information is then used to sort and divide in clusters of points, depending on M, the input point cloud P. This spatially sorted point cloud is then used in the third stage to search for the k-nearest neighbours of each point.

6.2.1 Loading

The input to this stage are raw point clouds acquired from a scanning device and stored in binary format, with each point represented as a triple (X,Y and Z coordinates) of type $float$. Since one of the objectives is to decrease the memory footprint of the application used to process a point cloud, whenever the number of points in the cloud is larger than the value of M, which is specified in number of points, a point cloud iterator is used which does not load the entire point cloud in memory. Instead, M points are loaded iteratively from file using MMFs. Since not all points are loaded in memory at any one point in time, each point cloud is represented as a collection of segments. The maximum size (in number of points) of a segment is M. The index of each point is thus represented using a local offset within the segment and its global index (within the whole point cloud P) is computed from the segment number and local offset. Figure 6.2 shows the straightforward abstraction adopted.

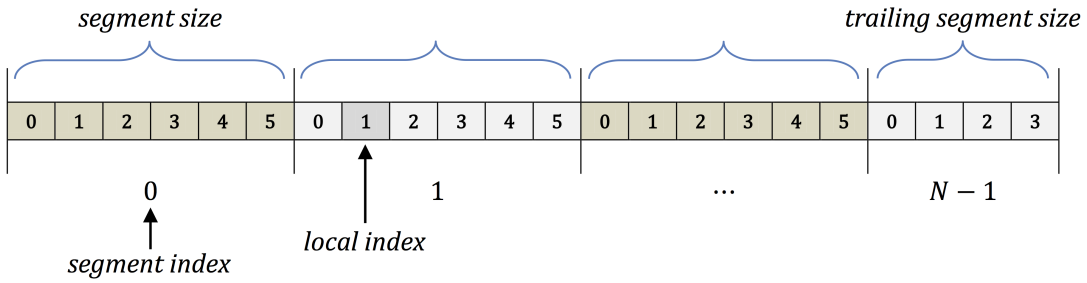


Figure 6.2: Input point cloud is loaded in segments.

A point cloud iterator $GetNext()$ first checks whether the next point to be returned is in the current segment, i.e. whether it is currently addressable in memory. If this is the case then values associated with the next point are returned, otherwise, if the end of segment is reached, the mapped region of the MMF is first deallocated then memory-mapped with the next segment. When the last point in the last segment is reached, $GetNext()$ returns $false$, indicating that all points have been read.

A uniform sparse grid is used to store the number of points contained within each axis-aligned cell in the sparse grid G. This information is used to persist the point cloud to disk ordered by cell index. For each point a key is computed which indicates the cell into which the point should be placed. The key is composed of three values representing cell indices along the X, Y and Z directions. The number of cells along each direction is computed from a user-defined value which specifies the size of each cell. Since points are fitted in a uniform grid, all cells have the same size. As outlined later on, this is an important consideration when searching for k-NN concurrently, and also to quickly determine if the correct k-neighbours have been chosen. In the experiments carried out, this value was set to 0.2 for all point clouds. The maximum number of cells in the sparse grid depends on the bounding volume of the input point cloud. For example if the bounding volume is (1,2,3) then the sparse grid would have a maximum of 5 cells along the X direction, 10 cells along the Y direction and 15 along the Z direction. Given that a sparse grid is used, only those cells where points are spatially located are created and stored in system main memory. In the largest point cloud (333M points) used to evaluate this approach, the number of cells in the grid is of 97,253.

6.2.2 Sorting

The output from the previous stage is a sparse grid G holding a count of the number of points contained within each cell. Given this information, together with a value for the approximate number of points in memory M and a specific ordering over grid cells, an optimal set partition of points in P is determined. This set partition groups together clusters of cells, over which k-NN can be computed in-core while adhering as closely as possible to the value of M. The cell ordering employed in the implementation follows in ascending order the X, Y then Z axis as illustrated in Figure 6.3.

This ordering implies that the bounding volume of the entire point cloud can be seen as being composed of a number of slices along the x-axis (*X-slices*), where each slice would consist of a number of cells varying along the Y and Z axes. Hence, one valid set partition of P would consist of cells grouped by X-slice. However, since points are usually not distributed uniformly across the bounding volume of the point cloud, there will be X-slices with many more points than others. Thus, the partitioning process groups together as many X-slices as possible.

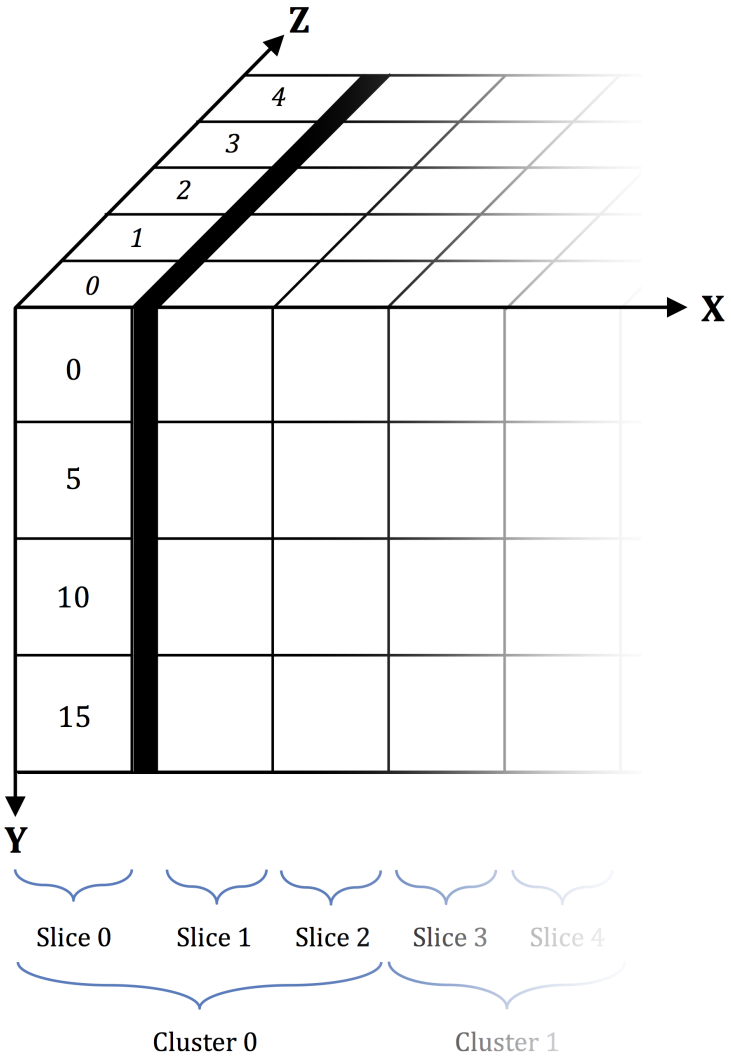


Figure 6.3: Sparse grid decomposition and cell ordering

M is used to determine the size of these clusters of X-slices, with each cluster having approximately M points. For example, if the axis-aligned bounding volume of the point cloud is divided into twelve X-slices, a possible set partition OC could consist of the four clusters {{1,2,3,g},{g,4,g},{g,5,6,g},{g,7,8,9,10,11,12}}. The partitioning process guarantees that the number of points present per cluster over which k-NN can be computed is approximately equal to M. In this case the number of points in the 4th X-slice (alone in the second cluster) is higher than the number of points in the rest of the slices. Hence, it is loaded in main memory on its own. An important aspect that needs to be taken into account when constructing this set partition, is the inclusion of ghost cells/points (represented using the letter g in the example) within each cluster, i.e. those points for which k-NN is not computed (within this cluster) but which may actually be one of the k-nearest neighbours for some of the points in the cluster. Figure 6.4 shows the ghost cells and respective ghost points for point p_i located in the central cell of the 3x3 grid. In the case of a 3D sparse grid, for every cell there can be a maximum of 26 ghost cells. For each cluster OC_i, the last X-slice from OC_{i-1} and the first X-slice from OC_{i+1} are added. Clearly, for OC_0 only the first X-slice from OC_1 is added, whereas for the last cluster OC_n only the last X-slice from OC_{n-1} is added. These additional cells representing the boundary points of the cluster are required to compute k-NN correctly. Since clusters are created over X-slices, the value of M must be reasonably chosen, i.e. it should not be very small. In the results section, the effect of changing this parameter is evaluated with respect to memory usage and performance.

The output from this stage is a file for each cluster of X-slices. Each file stores points following the cell ordering described in Figure 6.3. Point ordering within the cell is not important. Taking the example above, this stage would produce four files storing the points from clusters {1,2,3,4}, {3,4,5}, {4,5,6,7}, {6,7,8,9,10,11,12} respectively. In the next stage these files will be efficiently loaded in memory using MMFs.

Algorithm 11 describes the procedure used to sort the input point cloud P. For each cluster of X-slices in OC, a file is created. In order to write points at the correct offsets in each file, the information within each cell in G is augmented with file position offsets indicating at which location of the current file the next point contained in that cell should be written. Sorting is currently not very efficient since for each file written, the function $GetNext()$ has to iterate over all points in G. When the size of P is very large (e.g. 333 million points on a 1GB

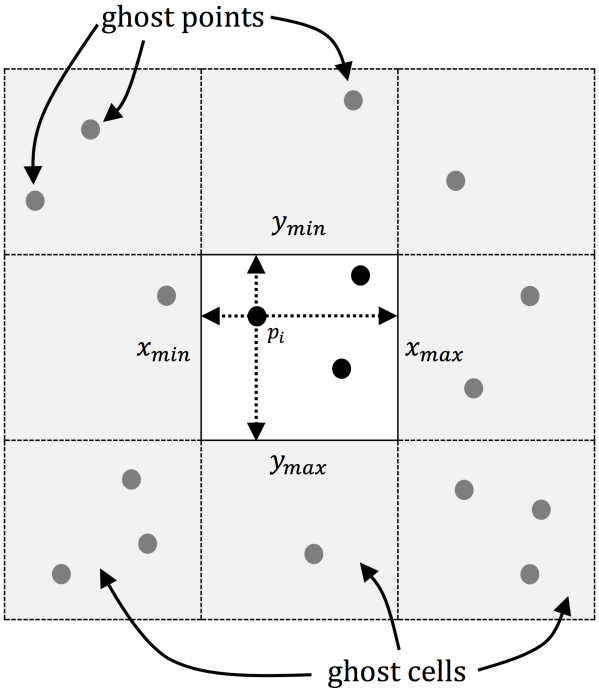


Figure 6.4: Ghost cells and points

machine) this actually becomes a bottleneck and ends up taking as much time as computing k-NN.

6.2.3 Concurrent search for k-NN

When computing the k-NN for a given point, our approach ensures that the correct k-nearest neighbours are actually returned. In general, given two sets of points P_g and P_n, with $P_n \subset P_g$, the method ensures that the set P_g contains the k-nearest neighbours of all points in P_n. As opposed to the work of Sankaranarayanan *et al.* (2007), i.e. pre-compute the set P_g before searching for the k-NN of points in P_n, the method verifies that this is the case for each point in P_n once the k-NN are determined. Since each point is located in an axis-aligned cell, the shortest distance d between the position of the point and any one of the boundary planes of the cell can be determined very efficiently. Figure 6.4 describes how this is done in 2D. After determining k-NN, the algorithm checks whether the distance between the k^{th} neighbour and the current point is smaller than d. If it is, then the currently chosen neighbours are correct and can be returned otherwise the point is flagged for re-computation of k-NN taking in consideration a larger set of adjacent ghost cells. Algorithm 12 describes in detail how the search

Algorithm 11 Sort points in P and persist to files

```
1: Input P, G with counts for each cell, Clusters OC.
2: for each cluster OC_i do
3:     Create MMF to store points in OC_i
4:     Update G with file position offsets of cells in OC_i
5:     cnt_total = number of points in OC_i
6:     cnt_written = 0
7:     for each point p_j ∈ P do
8:         if p_j falls within this cluster then
9:             Retrieve cell C_k where p_j is located
10:            Write p_j to file at position offset indicated at C_k
11:            cnt_written = cnt_written+1
12:            Increment offset at C_k
13:        end if
14:        if cnt_written == cnt_total then
15:            Flush MMF of OC_i.
16:            Continue.
17:        end if
18:    end for
19: end for
```

for k-NN works.

Each processing element (PE) in the system atomically retrieves the next available cell in the currently active OC cluster and computes k-NN searches over all points in the cell. k-NN searches are carried out by creating a temporary kd-tree over points in the currently active grid cell. When all searches are done, the kd-tree is deleted from memory. Temporary kd-trees are created and deleted for all cells in G.

6.3 Results

The out-of-core extension to PaRSe is evaluated on a number of point clouds ranging in size from 53K to 333M points. All experiments are carried out on an Intel Core2Quad machine running Windows7 and SATA2 hard disks. In order to evaluate performance against different memory configurations, the same machine is installed with 1GB, 2GB, 4GB and 8GB of system RAM. Experiments are conducted in order to evaluate the scalability of the approach as the size of the point cloud is increased across these different memory configurations. In addition to an implementation of the concurrent grid based multi kd-tree ($GridXKd$)

Algorithm 12 Compute k-NN for all points $p_i \in P$

Input G, Cluster Set OC.
for each cluster OC_i **do**
    Memory map file with points in OC_i
    Update file position offsets of cells in OC_i
    Generate array $CellArr$ storing keys of cells in OC_i
    $cellCount$ = size($CellArr$) - no. of ghost cells in OC_i
    $crtCellIdx$ = index of first non ghost cell
    while $crtCellIdx < cellCount$ **do**
        Atomically assign to PE $crtCellIdx$
        PE generates kd-tree on points in $CellArr_{crtCellIdx}$
        for each point p_j in $CellArr_{crtCellIdx}$ **do**
            Search for k-NN of p_j
            d = shortest dist(p_j,$CellArr_{crtCellIdx}$ planes)
            if dist(p_j, NN_k) $> d$ **then**
                Add p_j to k-NN re-computation list RL
            end if
        end for
        while sizeof(RL) > 0 **do**
            Update kd-tree with points from adjacent cells
            Compute k-NN for p_j
            d += extent of $CellArr_{crtCellIdx}$
            if dist(p_j, NN_k) $< d$ **then**
                Remove from re-computation list RL
            end if
        end while
        Delete kd-tree
        Atomically increment $crtCellIdx$
    end while
end for

approach described above, two further implementations are evaluated for comparison. The first implementation takes the traditional in-core approach where a kd-tree is constructed over all points in the data set and is referred as in-core kd-tree ($ICKd$). This implementation should provide the best possible performance whenever enough memory is available to hold the kd-tree. The PCL library Rusu & Cousins (2011) is used for this implementation which also uses memory mapped binary files to store points. The second implementation works exactly like $GridXKd$, but does not use memory-mapped files and instead loads all points in the sparse grid data structure (rather than just the required number of points) before starting to compute k-NN and is referred to in-core concurrent grid based multi kd-tree ($ICGridXKd$). In all cases the FLANN library Muja & Lowe (2009a) is used to implement kd-tree based k-NN searches. The error-bound parameter ϵ is set in all cases to zero. Moreover in all implementations all four processing elements available on the computer used are utilised to concurrently compute k-NN.

Table 6.1 lists the point clouds used in the experiments. In all cases (except for Mnajdra and Songo) the data has been generated from polygonal models. In the case of SongoX2, SongoX4 and SongoX8, the original point cloud was upsampled (using Algorithm 3) in order to increase the number of points. For each point in the original point cloud, an additional point is created as the spatial average of the two nearest neighbours. Figure 6.5 illustrates three of the point clouds used.

Model Name	Size(M)	Cell count in Grid
obelisk	0.053	1097
mnajdra	0.579	6087
conference	2.3	6338
sibenik	6	201,756
songo	41	95,999
songoX2	83	96,940
songoX4	166	96,853
songoX8	333	97,253

Table 6.1: Point clouds, corresponding number of points and number of cells created during loading phase in sparse grid

6.3.1 Execution Time

Execution times are first compared for all three implementations on a machine installed with 8GB of system memory and is done in order to first establish the best possible results for the three implementations. In the case of the $GridXKd$, parameter M is set to a value greater than the number of points in the cloud in order to maximise the use of system memory. $GridXKd$ is later evaluated with different values of M in order to establish how this constraint effects execution time. Table 6.2 shows the time it takes for each implementation to calculate k-NN with k set to 16 for the different models.

Model Name	ICKd	ICGridXKd	GridXKd
obelisk	0.127	0.193	0.241
mnajdra	1.164	2.068	1.748
conference	4.864	7.726	5.891
sibenik	12.032	20.039	15.911
songo	101	198	167
songoX2	207	420	353
songoX4	916	-	707
songoX8	-	-	1426

Table 6.2: Execution times (in seconds) using 8GB RAM

Note that the readings for $GridXKd$, also include the time taken to populate the sparse grid G and persist to file (or files depending on the number of clusters created at the sorting phase) a sorted version of the original point cloud. As the size of the point cloud increases so does the time taken to sort it. This is evident when working with the largest points clouds. As was to be expected $ICKd$ performs better in those cases were the acceleration structure can easily fit in main memory. However, as the size of the input data set increases, the performance of the proposed approach ($GridXKd$) is better than that of $ICKD$ and $ICGridKd$. Due to the in-core nature of both $ICKD$ and $ICGridKd$, both are not able to process the 333 million point data set songoX8. In the case of $GridXKd$ the execution time is linearly proportional to the size of the input. In the case of the point cloud songoX4, the proposed approach performs better than the in-core $ICKd$.

Execution times of all three implementations are also evaluated whilst decreasing the amount of system memory available. Tables 6.3, 6.4 and 6.5 show

execution times for all data sets with 4GB, 2GB and 1GB system memory installed. Table 6.3 shows the results obtained with 4GB system memory installed. When processing the largest point cloud (songoX8), in order to limit the amount of memory required by $GridXKd$, parameter M is set to 100 million. When M is not set to a value smaller than the size of the dataset, too many points would have been present in system memory resulting in our approach not being able to process songoX8. In order to be able to process this point cloud, M is set to a value smaller than the number of points in the cloud. With M set to 100 million points, $GridXKd$ computes all k-NN in 1909 seconds. Given the size of the point cloud, a considerable amount of time, 358 seconds, is spent on the sorting phase which partitions the dataset into 5 clusters. Table 6.6 shows the effect of varying M on both load and sorting times of $GridXKd$. The number of segments created at load time and the number of clusters created at the sorting stage are also listed. Once the point cloud is loaded, sorted and persisted to file/s the time taken to compute k-NN is the same across all variations of M with 1GB of system memory installed. These results show that with 1GB of RAM installed, the best results are obtained when setting M to 20 million with the sorting stage partitioning the input point cloud into seven clusters.

Tables 6.4 and 6.5 show execution times for all data sets with 2GB and 1GB RAM installed. In all cases $GridXKd$ is able to compute k-NN for all points. When using 1GB, with point clouds of more than 20 million points, M (values shown in table) is used to reduce the number of points which are simultaneously loaded in memory. In all cases the value is set to 30 million or less. As shown in Figure 6.6, as the number of points increases, a considerable amount of time is spent sorting the point cloud. In the current implementation loading and sorting is always performed, however this is not required. Once a sorted point cloud is persisted to file it can be reloaded without incurring the cost of re-sorting. In this case, the sparse grid G would need to be persisted with the rest of the data and reloaded each time. When processing large data sets this operation is much less expensive than sorting.

Results show that the $GridXKd$ implementation within PaRSe, is able to efficiently compute k-NN searches on very large point clouds with minimal system memory. In the case of small point clouds, $GridXKd$ results are comparable to the results achieved by an optimal in-core implementation of k-NN search. This demonstrates the scalable nature of the proposed approach. For a neighbourhood of size k=16, using either 8GB, 4GB or 2GB of system memory, $GridXKd$ is able

Model Name	ICKd	ICGridXKd	GridXKd (M)
obelisk	0.109	0.234	0.172
mnajdra	1.469	2.047	1.921
conference	5.046	8.203	6.031
sibenik	13.875	17.726	16.281
songo	114	205	169
songoX2	443	-	356
songoX4	-	-	786
songoX8	-	-	1909 (100M)

Table 6.3: Execution times (in seconds) using 4GB RAM

Model Name	ICKd	ICGridXKd	GridXKd (M)
obelisk	0.156	0.213	0.157
mnajdra	1.547	2.031	1.673
conference	5.219	7.609	5.957
sibenik	14.641	21.953	16.221
songo	238	-	170
songoX2	-	-	379
songoX4	-	-	1160
songoX8	-	-	1577 (60M)

Table 6.4: Execution times (in seconds) using 2GB RAM

to compute approximately 235,000 neighbourhoods/s on an 83 million point data set.

6.4 Discussion

The processing of very large point clouds is generally hindered by the amount of device system memory, with functions such as k-NN assuming that all points are loaded in memory. An extension to PaRSe is presented in this chapter, which addresses the problem of working with point clouds that do not entirely fit in main memory. In particular, the computation of a point's k-NN is essential for PaRSe in establishing point types which are then used by the region-growing process. An out-of-core algorithm which takes in consideration both the size of the input point cloud and available system memory is described, which gives PaRSe the capability of processing very large point clouds on systems with at least

Model Name	ICKd	ICGridXKd	GridXKd (M)
obelisk	0.147	0.243	0.171
mnajdra	1.648	2.323	1.673
conference	5.132	8.102	6.345
sibenik	17.231	24.252	16.454
songo	-	-	300 (20M)
songoX2	-	-	522 (20M)
songoX4	-	-	1541 (30M)
songoX8	-	-	5995 (30M)

Table 6.5: Execution times (in seconds) using 1GB RAM

M(million)	Segments	Load(s)	Clusters	Sort(s)
10	9	11.39	20	235
20	5	10.86	7	103
40	3	7.297	3	136
60	2	9.336	2	150
85	1	14.156	1	251

Table 6.6: Varying values of M on the songoX2 data set (with 1GB RAM installed)

1 Gigabyte of system memory. Results have shown that the method proposed compares favourably to an in-core kd-tree based implementation.

A number of future developments are required to improve on the current implementation. The initial sorting phase requires additional research aimed at reducing the time required for this to be carried out. Once the initial point cloud sorting is done, sub-sampling techniques can be used to appropriately reduce the size of the input to match the size of available system memory. In addition, an in-depth analysis of trade-offs between number of cells in the sparse grid and the average number of points in each cell would help in establishing optimal parameters. Another interesting future direction is that of extending the concept of the error-bound ϵ used for ANN in kd-trees, and include in the sparse grid subdivision of space.

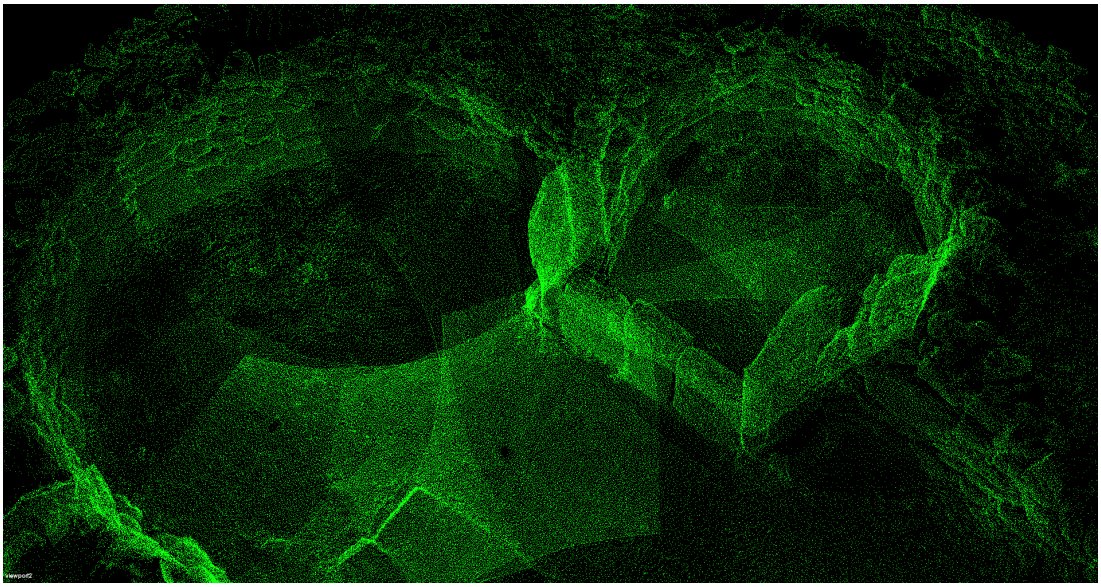
In the current implementation, kd-trees computed on points in a cell are not persisted to file with the points and therefore, when a cell is mapped to system memory, a new kd-tree structure needs to be computed. Whereas, results have shown that this overhead is minimal, further research should look into the possibility of lazy loading pre-computed kd-tree structures from file and thus

avoiding re-computations of kd-trees.

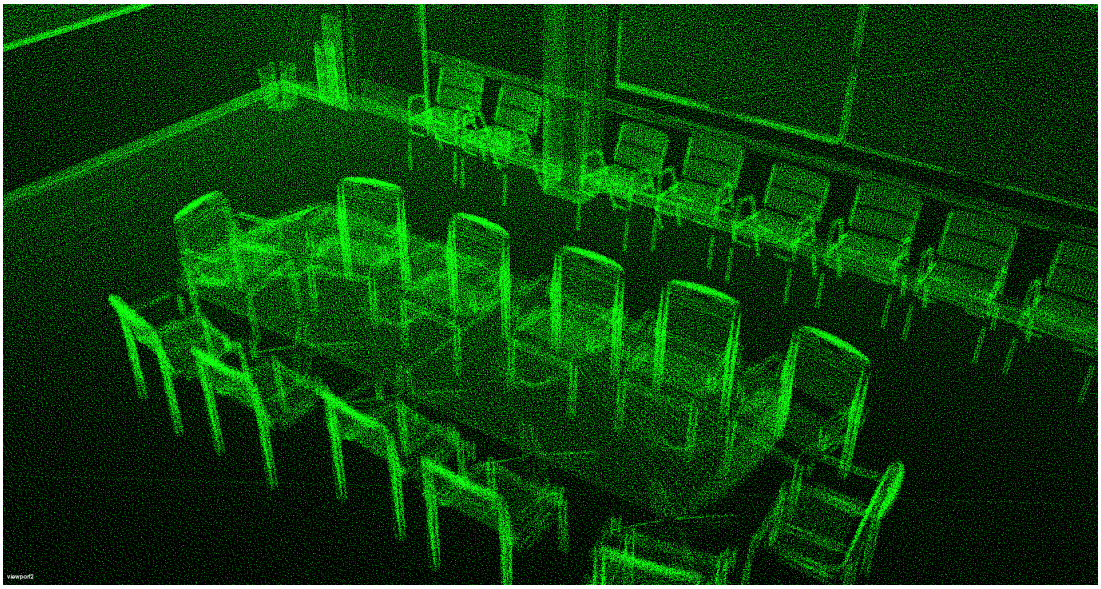
Variations of PaRSe may be used on mobile autonomous systems in order to acquire elaborate environments. In these cases, the typically limited amount of physical memory on these devices poses a limit on the size of point clouds in working memory which can be processed. The method presented in this chapter can be used as an initial platform to deliver such a system. Further research would look into the support offered by operating systems deployed on these devices.

6.5 Summary

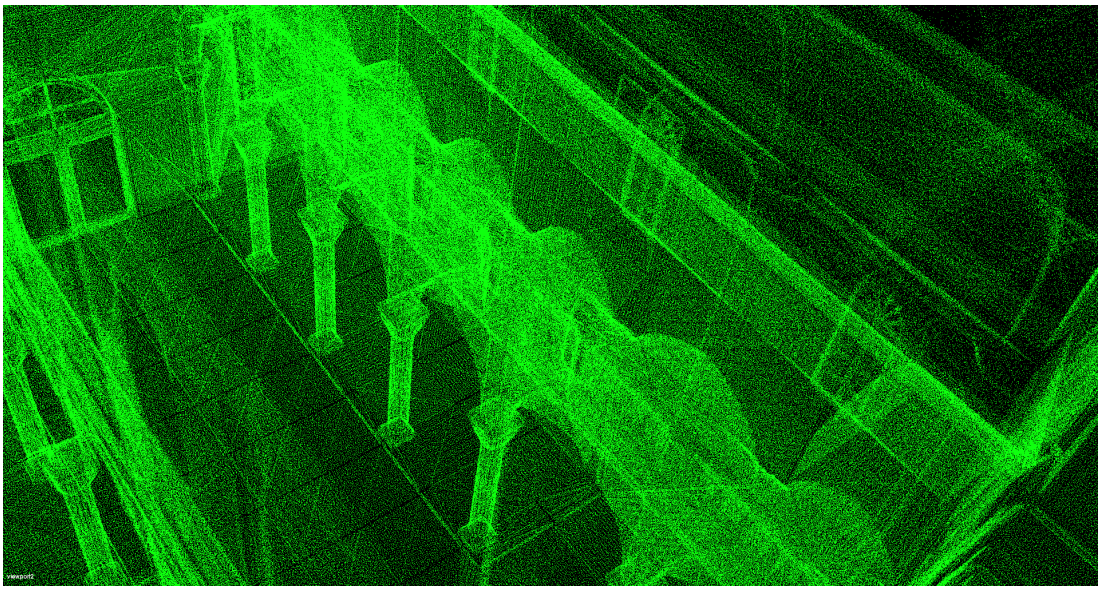
This chapter has presented an extension to PaRSe, consisting of a novel out-of-core algorithm which efficiently searches for the k-nearest neighbours of points over very large point clouds. Results have shown that with the proposed method, PaRSe can scale up from a few thousand points to several millions on devices with limited memory resources. This capability further widens the applicability of PaRSe in addressing challenges and tasks involving very large point clouds on devices with limited amounts of system memory. In the next chapter, PaRSe is used in the design of CoFFrS, a context-free scene understanding framework which is applied to points clouds of indoor scenes.



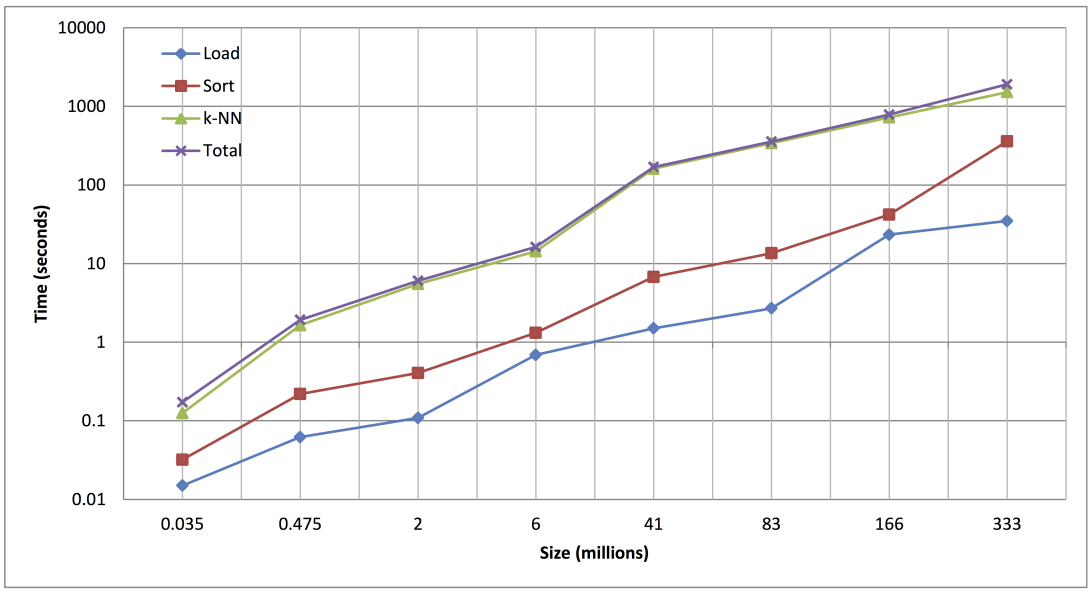
(a) Mnajdra 579K points



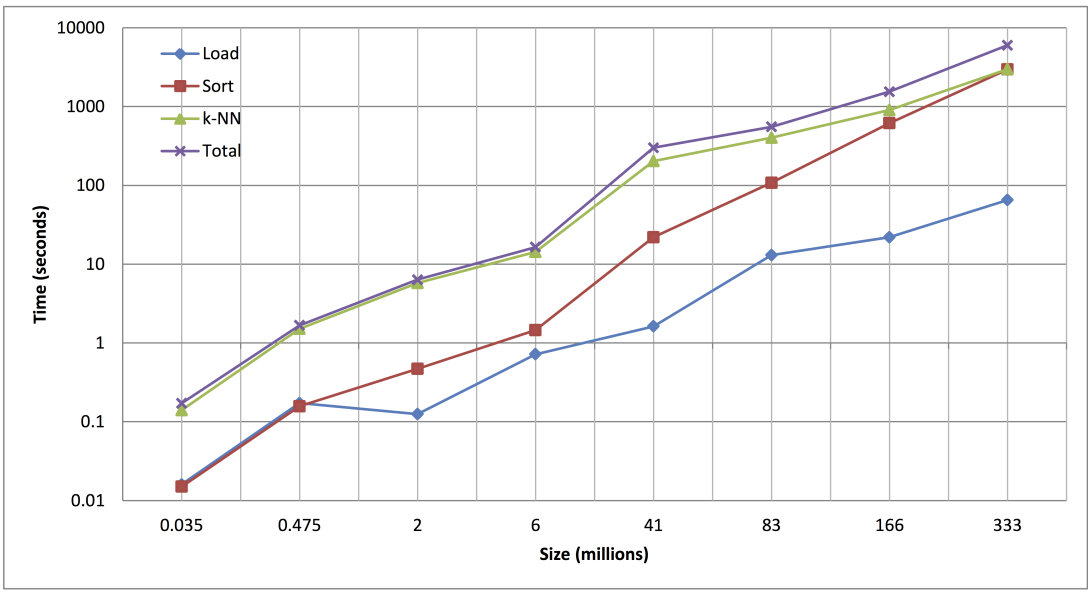
(b) Conference 2.3M points



(c) Sibenik 6M points



(a) Using 4Gb RAM



(b) Using 1Gb RAM

Figure 6.6: Execution times for load, sort and compute k-NN

CHAPTER 7

Structure Graphs for Indoor Scene Understanding

The growth in popularity of commodity hardware capable of capturing depth information is further widening the range of application of 3D data sets. In particular, scanners based on triangulation principles (§2.3.2) such as the Microsoft Kinect, Asus Xtion and Structure Sensor, are being extensively used to acquire indoor scenes. Recently, rapid advances in ubiquitous computing have also brought to the masses the possibility of capturing the world around us in 3D using smartphones and tablets (Google, 2014; Jared, 2014). As a consequence of these advances, there has recently been a surge in the development of scene understanding methods of indoor environments from point cloud data. This chapter contributes a novel approach, CoFFrS (Context-Free Framework for Scene understanding), which builds on the segmentation process PaRSe and generates successful results on indoor scenarios which were previously unsolved.

Point cloud segmentation methods, which partition point cloud data into smaller meaningful components, generally contribute towards improving the handling and further processing of this data. As described in Chapter 3, point cloud segmentation algorithms have traditionally used either a region-growing or parametric shape fitting approaches in order to partition the input. In the case of region-growing algorithms the resulting partitions are usually not very meaningful and therefore less amenable to further processing. Parametric shape fitting on the other hand returns a set partition with each element containing a set of points within the parameters of a specific shape. Given this information, the elements of the set partition can be used to describe higher-order concepts such as roofs or columns of buildings.

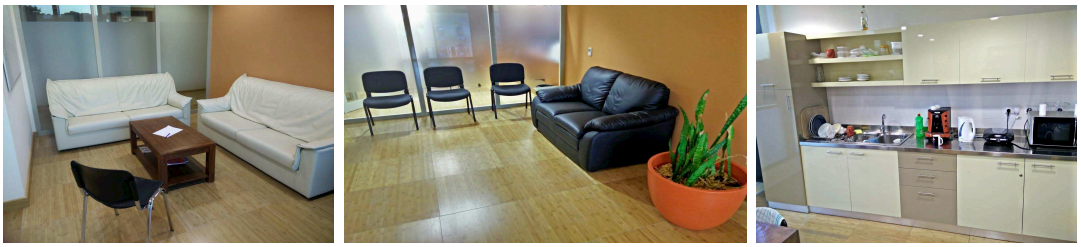


Figure 7.1: Photographs of environments typically used to evaluate scene understanding algorithms.

Segmentation, object recognition and indoor scene understanding techniques share many common aspects. Whereas traditional indoor scene understanding methods deal with the identification of structures and objects in an indoor environment, 3D object recognition techniques have traditionally been used to identify small objects on a surface given a set of trained object descriptors. In many cases, the segmentation of the input point cloud P is not considered, since the identification process proceeds by matching previously trained point-based descriptors of objects to point-based descriptors computed in P (§4.1.1). Alternatively, simple point clustering algorithms are used to first delineate the individual objects, with the main goal in these cases being the identification of objects in P from different views and under a variety of occlusion and noise parameters. These point based descriptors generally assume that objects are uniformly scanned in sufficient detail. However, in an indoor scene understanding context using commodity hardware, this is generally not the case with small objects sampled coarsely (§2.3.2). To address this situation, the scene understanding methods described in 4.2 have resorted to using a segmentation process over P, in conjunction with a scene descriptor trained from the resultant segments.

Figure 7.1 illustrates three indoor environments which can easily be scanned using commodity hardware. Given point clouds representing these scenes, the task of a scene understanding method is that of identifying the different components and objects making up the scene (§4.2). For many scene understanding techniques, using either supervised (§4.2.1)) or unsupervised (§4.2.2) methods, segmentation is critical in order to establish an initial clustering of points in the scene. Supervised methods use a training process which results in a scene descriptor encoding information about the objects and scene (e.g. decision forests in Nan *et al.* (2012)). As such, these algorithms only work on point clouds representing environments which are very similar in terms of object positions and

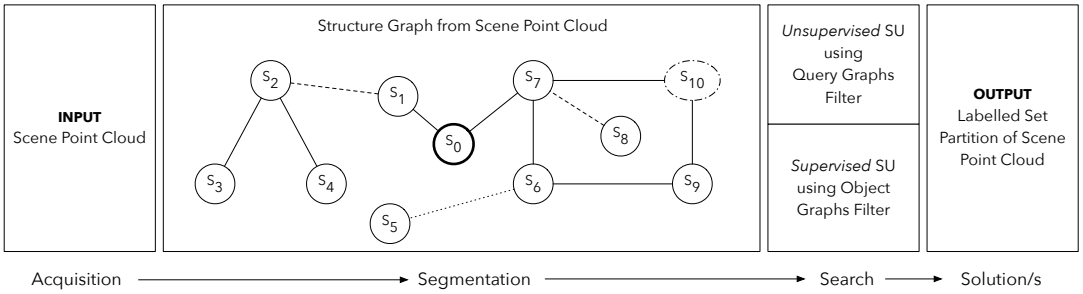


Figure 7.2: In CoFFrS, the scene understanding process uses query and object graphs to produce an interpretation of a scene. Query graphs first extract specific patterns, then object graphs determine the presence of objects. Point based object descriptors (§4.1) may be used on some of the extracted partitions.

pose to the ones used for training. Unsupervised methods do not utilise a labelled training set, and instead rely on the identification of symmetry and repetition in a scene (or multiple scenes) in order to produce a set partition matching these patterns (Mattausch *et al.*, 2014). However, establishing partitions based of the identification of hidden structures in a scene is not sufficient to properly associate object labels.

A considerable amount of work has recently been carried out in the area of indoor scene understanding from point cloud data. Segmentation is in some cases used for shape recognition (Schnabel *et al.*, 2008; Golovinskiy *et al.*, 2009; Lin *et al.*, 2013) and always required for indoor scene understanding (Nan *et al.*, 2012; Kim *et al.*, 2012; Mattausch *et al.*, 2014). A number of methods first apply the RanSaC paradigm to fit parametric shape primitives to unstructured raw point clouds (Dorninger & Nothegger, 2007; Schnabel *et al.*, 2007). Graph-based 3D object descriptors have been used to encode geometric and topological properties from the shapes extracted (Schnabel *et al.*, 2008; Golovinskiy *et al.*, 2009). Both supervised and unsupervised methods have been applied to search for object descriptors within point clouds. Whereas supervised methods utilise a training phase in order to synthesise descriptors of individual objects, unsupervised methods rely on the presence of patterns to automatically infer similar objects in a scene. Golovinskiy & Funkhouser (2009) presents a segmentation and scene understanding algorithm for outdoor scenes based on foreground/background identification. Indoor scenes however, usually present a harder segmentation challenge due to noise induced by added clutter, sensors and partial object occlusions and are not ideal for such an approach. Mattausch *et al.* (2014) addresses

the scene understanding task by exploiting similarities within indoor scenes and describes an unsupervised segmentation process for point clouds resulting in clusters of similar objects. When these similarities are absent, for instance due to low quality acquisition sensors, or simply because the scene lacks similarities and symmetries, the effectiveness of these techniques diminishes. With supervised methods, scene-specific knowledge is embedded in trained scene descriptors. Nan *et al.* (2012) propose a search-classify approach for interleaving segmentation and classification. Although managing to successfully classify complex scenes, their method fails when object placement in the scene differs in pose or scale to that used when training the scene-specific classifier. Kim *et al.* (2012) propose a system which also handles model variability modes. As opposed to our method however, they assume that the vertical direction of the models and the scene are fixed. This makes it difficult to detect overturned objects as opposed to our method which orients models in a scene according to the identification of dominant planar segments of the trained object descriptor. Shao *et al.* (2012) propose an interactive approach to indoor scene understanding, where users manually improve segmentation results prior to identification.

In this chapter, PaRSe structure graphs are extended and utilised in the design of CoFFrS, a framework for context-free scene understanding. The many operations that can be carried out on structure graphs can predominantly be formulated as search tasks. Therefore, given a point cloud P and structure graph $\mathcal{G}$ resulting from applying PaRSe on P, a method is required which traverses output $\mathcal{G}$ in order to identify as sub-graphs any objects and structures it may contain. The successful execution of CoFFrS, heavily relies upon PaRSe which minimises the grouping of unrelated points and consistently outputs similar set partitions and structure graphs given multiple runs of the process on a given input. Figure 7.2 illustrates the building blocks of the method presented in this chapter, which has been evaluated using examples of indoor scenes, with the input consisting of raw point clouds acquired using commodity sensors.

One of the main aims of this work is the design of a generic, context-free scene segmentation and understanding pipeline. In this respect, the technique presented in this chapter does not rely on a specific scene context, thus making it applicable to a wide spectrum of domains. In this chapter, the focus is primarily on indoor scenes which are either acquired specifically to evaluate CoFFrS or are available from literature. The contributions of this chapter are summarised as follows:

- address indoor scene understanding tasks using a generic point cloud segmentation pipeline which partitions raw point data into connected segments.
- extend query graphs into object graphs in order to describe the salient geometric features of a point cloud representing an object and how these are connected.
- an incremental scene understanding process which enumerates the space of solutions mapping objects to surface segments in the target scene.

7.1 Method Overview

The distinguishing features of an object may be perceived in a variety of ways including, for instance, variations in shape or colour. One such feature is described in the work of David Marr (Marr & Poggio, 1979), which links the perception of 3D objects to the saliency of flat surfaces. It builds upon the observation that many objects (especially man-made) present in a target scene can be segmented into a number of planar segments which exhibit specific connectivity patterns between them and that these patterns can be used to discriminate between different objects. The object representation scheme adopted by CoFFrS takes inspiration from this work and uses information about the extracted planar segments of an object to concisely describe it. In the descriptor proposed, if at least one planar surface segment can be identified in an object, its relationship with the other points and segments can be used to describe it. These object descriptors are synthesised from point clouds representing individual objects that may be present within a scene, and are encoded by first partitioning them into connected *surface·planar* segments using PaRSe and then extending the resulting structure graph to an object graph. Theses object graphs are subsequently used during the searching phase, for the automatic extraction of trained objects from target point clouds.

In PaRSe (Chapter 5), a number of segment types are defined resulting from region growing and fitting of plane primitive shapes. A structure graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is defined over these segments encoding different connectivity aspects between them. Both nodes and transitions store properties describing geometric details such as the volume spanned by the points in the node and the surface

normal of the plane. Given $\mathcal{G}$, representing the input point cloud, query graphs encoding higher-order concepts are used to group together segments in $\mathcal{G}$ by matching the connectivity patterns of a query graph (§5.5). Query graphs allow for a variety of tasks to be carried out on the input point cloud. This is important in cases where a specific task can be accomplished by searching for relatively simple patterns in a point cloud. However, in a indoor scene understanding context, query graphs can be difficult and time consuming to create. Therefore, whereas there is scope for manually crafted query graphs, there is also scope for automatically producing these query graphs for more complex scenarios as those associated with a typical scene understanding context.

The scene understanding framework presented in this chapter addresses the following design goals:

- Identification of objects does not depend on specific global scene parameters which are encoded in the scene descriptor, and therefore enables *pose invariance* for objects.
- *No user input* is required during the segmentation process.
- Provides mechanisms allowing user input to *guide the searching process* and prune the search space.
- Provides a framework with the possibility of returning *multiple ordered solutions* to the scene understanding problem.
- Object matching *robust to both noise and occlusion.*

Figure 7.3 illustrates an overview of CoFFrS starting from the acquisition of a point cloud P and resulting in a labelled set partition of P. The rest of this section describes the input to this pipeline and outlines the different levels of quality obtained when using triangulation-based commodity hardware to acquire indoor scenes. Section 7.2 then briefly describes the transformations carried out on a point cloud of an indoor scene until this results in a structure graph (details in Chapter 5) and provides examples showing how this segmentation process is adequate for indoor environments. The training phase used to encode object descriptors is then described in §7.3, whereas the scene understanding phase using these descriptors is described in §7.4.

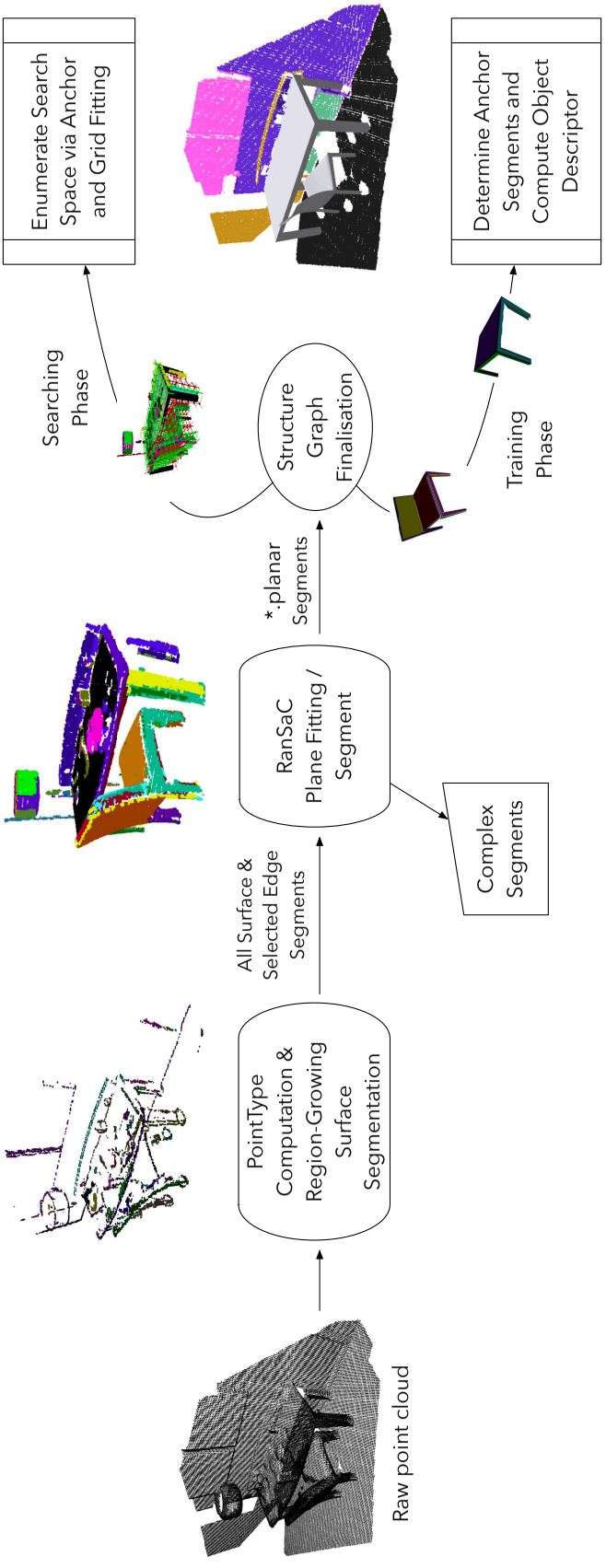


Figure 7.3: Scene Segmentation and Understanding Pipeline Overview

7.1.1 Scanning of Indoor Scenes

The results obtained by a scene understanding method are impacted by the quality (§2.3.3) of the input point cloud. Figure 7.4 illustrates examples spanning a range of quality values. The conference room (bottom row) has been synthesised from a dense triangular mesh, and thus all surface information is present and accurate. On the other hand, the office room (Nan *et al.*, 2012) in the top right corner, table top point cloud and office room in the middle row are acquired using commodity depth sensors resulting in relatively noisy and incomplete scenes. This is particularly visible in the larger office room which contains a table, a desk, shelving on one side of the room, a variety of small objects and a number of chairs. Figure 7.5 shows close-ups of the table and chairs in this point cloud, highlighting how the same scene can be represented at different levels of quality. For instance, additional important surface samples are acquired if the scanner position, in this case the Asus Xtion, is moved closer to the table and chairs during acquisition. This is evident in the middle row of Figure 7.5 which illustrates exactly the same office scene with additional detail captured closer to the table and chairs. This difference in point cloud quality is a consequence of the sensor location from where the acquisition process is carried out. The bottom left hand corner image shows a top-down view of the office with red and blue octagon shapes showing the positions of the sensor in both cases. In terms of the three quality criteria described in §2.3.3, and if only the table and chairs are considered as the original signal to be re-constructed, then the top-row point cloud is clearly inferior in the first two criteria. Namely, less patches are acquired (e.g. no legs) and the signal to noise ratio for each patch is lower.

7.2 Segmentation of Indoor Scenes

CoFFrS builds upon and depends on the results of the segmentation pipeline described in the previous chapter. This section provides a number of examples showing the behaviour of the segmentation process on a number of indoor scenes which are either taken from previous work by Nan *et al.* (2012) or are newly acquired. The first three columns in Figure 7.6 illustrate the segmentation process carried out on point clouds representing two separate chairs and an indoor office scene (Nan *et al.*, 2012). The third column illustrates the **·planar* segments resulting from this process visualised using different colours.

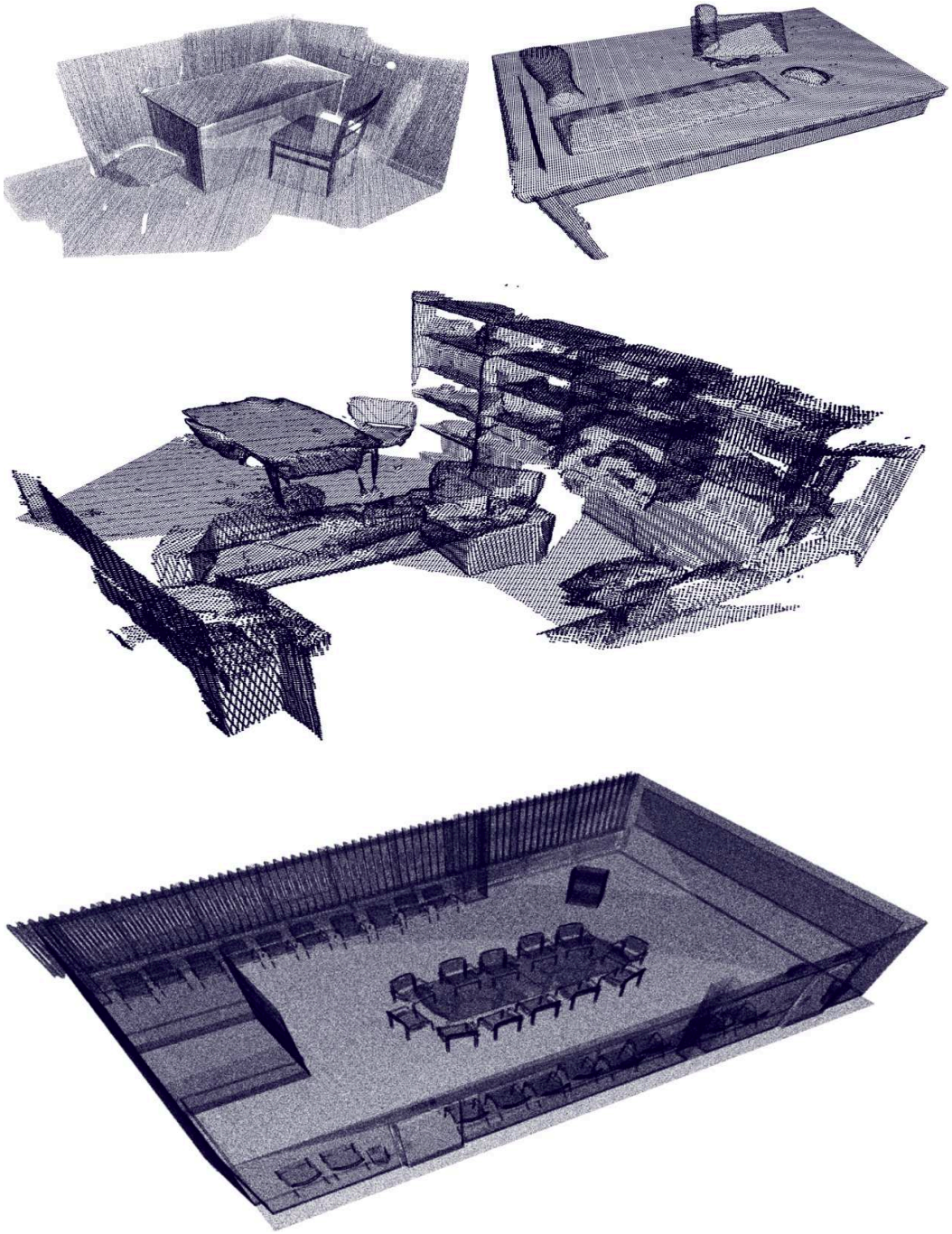


Figure 7.4: Top left corner shows office room from previous literature (Nan *et al.*, 2012), top right corner shows table top scanned using the Asus Xtion sensor with the Skanect software, middle row illustrates the point cloud of an office acquired using the Structure sensor and Skanect software and, bottom row illustrates the point cloud (synthesized from a triangle mesh) of the conference room

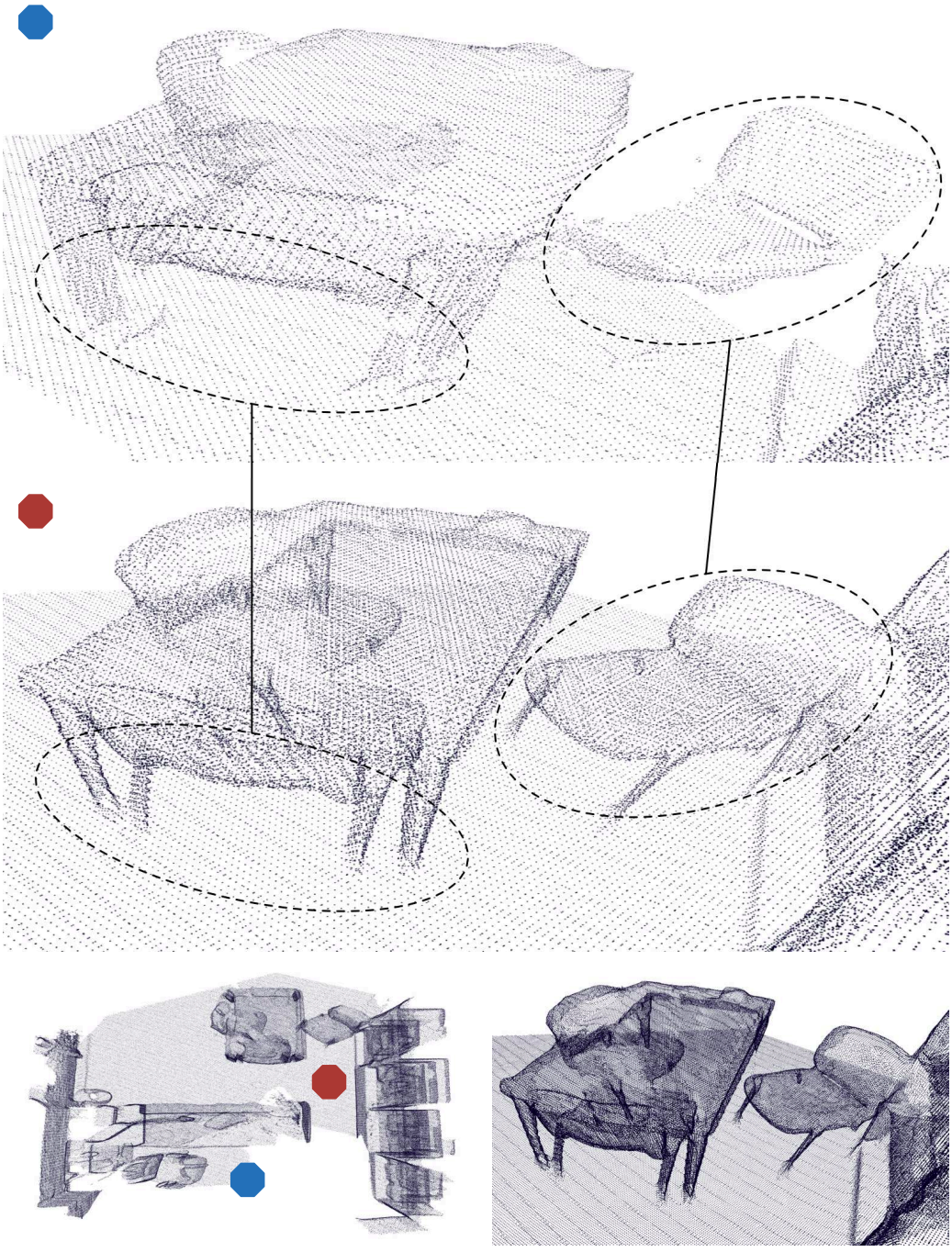


Figure 7.5: Close up on table and three chairs of office room in figure 7.4

Figure 7.6: Segmentation process on two chairs (single objects) and office (multiple objects in enclosed space) scenes with columns from left to right (a) all points, (b) edge type points, (c) segmentation result - shown as coloured surfaces, (d) close-up view

PaRSe first labels points, then partitions P using a region growing algorithm and applies a RanSaC plane fitting process over resulting regions. This is particularly useful in the case of indoor scenes where noise in the acquired point cloud and variability in point density can lead to scenarios where points which should be tagged as edge are actually tagged as surface and vice-versa. In the former case, this leads to situations where a surface segment spans over multiple object surfaces. Figure 7.7 on the left, illustrates an example where only one surface segment is created for the right-most blue couch following the region growing process. This segment effectively contains all the points on the couch and could be useful on its own. However, in order to be able to describe a couch in terms of segments and connectivity between them, all couches in the scene need to be composed of a similar set of segments. Figure 7.7 on the right, illustrates the segments resulting from the RanSaC process. Note how on both couches, the seat is further split in two *surface·planar* segments to match the slight curvature.

PaRSe produces a set partition $\mathcal{S} = \{s_1, s_2, s_3, \ldots, s_n\}$ of P consisting of elements with type *surface·planar*, *edge·planar*, *surface·complex* or *edge* (§5.1.2). A structure graph $\mathcal{G}$ is built over these segments using adjacency information obtained during the region growing process and OBB (§2.5.3) intersection tests during RanSaC plane fitting. Segments of type **·planar* play a critical role in our scene understanding approach. Transitions in $\mathcal{G}$ between these segments are augmented with properties in the form of $\langle key, value \rangle$ pairs, e.g. $\langle dot, 0.02 \rangle$ to indicate that the nodes connected by this arc are nearly orthogonal. $\langle key, value \rangle$ pair properties are also attached to nodes and include number of points, plane orientation, area spanned by OBB, points coverage on surface and spatial context

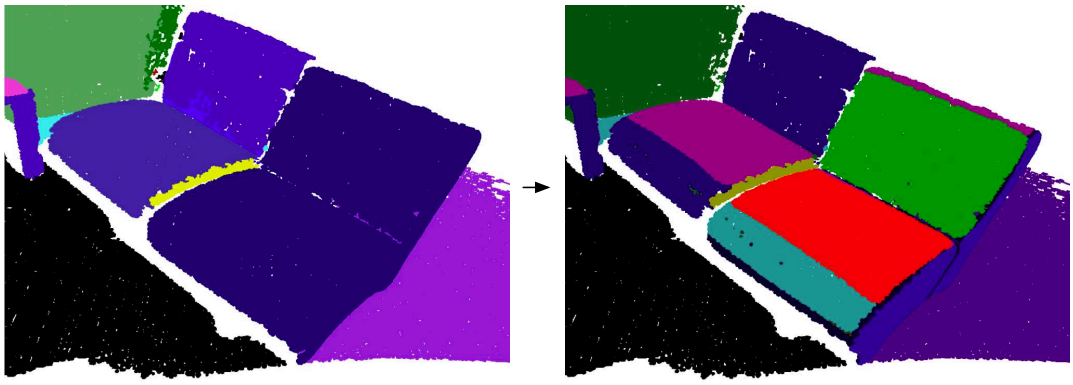


Figure 7.7: Scene from Nan *et al.* (2012) - a) over segmentation of right sofa and b) new segments created after RanSaC plane fitting

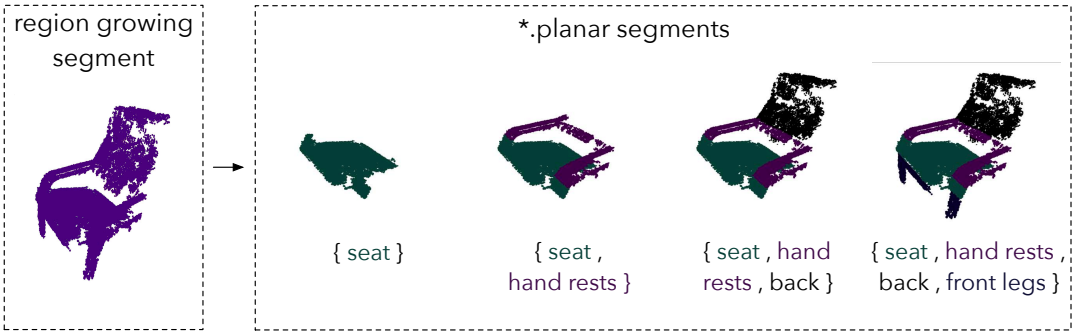


Figure 7.8: Region growing segment includes the whole armchair, whereas RanSaC plane fitting subdivides this region into four **. planar* segments.

information. Spatial context is used to determine the approximate location of the **.planar* segment along its normal direction within the object or scene, and is set to either *boundary*, *central* or *boundary central*. Figure 7.9 illustrates an example showing how spatial context is computed. In the case of object segmentation, which is described in the next section, spatial context is computed with respect to the entire point cloud. In the case of scene segmentation, spatial context for **.planar* segments is also computed with respect to the region in which they belong. Points coverage is used to measure how points are distributed over the **.planar* segments. This is done to further discriminate between segments which might have similar OBB areas but with points clustered in specific parts of the OBB. Figure 7.10 illustrates a number of examples showing how point coverage is computed. The upper part of the figure shows four OBBs enclosing the side of a chair, a triangle, a circle and a rectangle. The areas of the OBBs of the rectangle

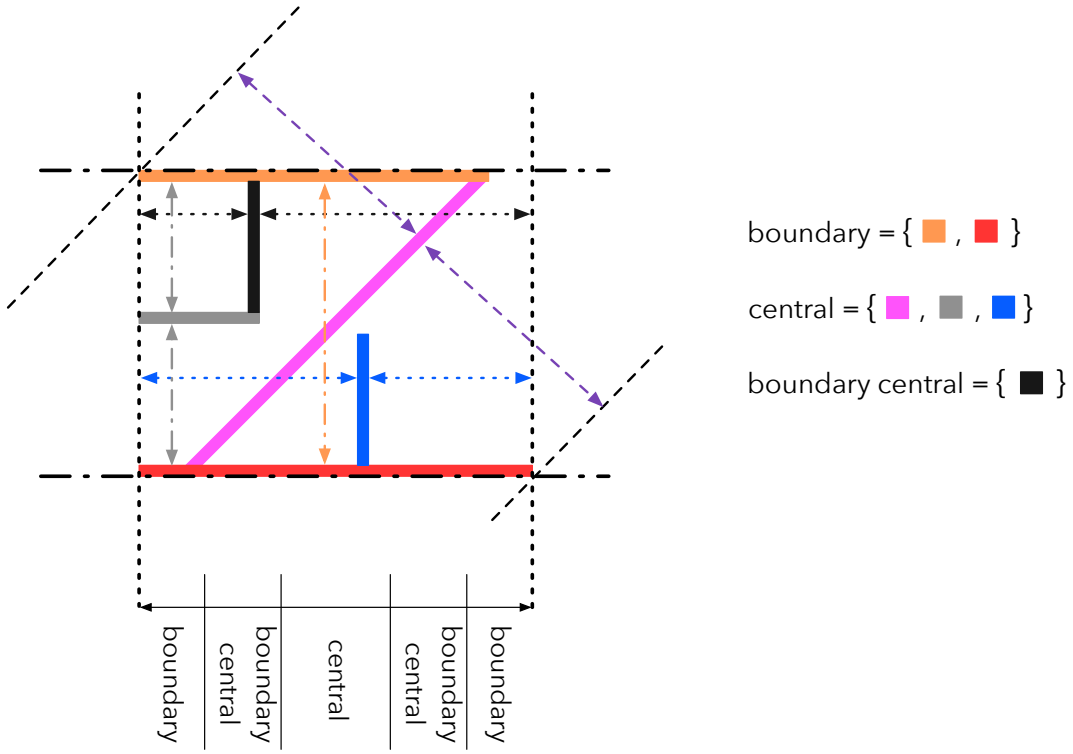


Figure 7.9: Spatial context indicates the approximate location of each segment within the point cloud being scanned which could either represent a scene of an individual object. In the case of a scene, if the planar segment forms part of a region induced by the region-growing algorithm, spatial context is also computed with respect to the points in the region.

and chair are very similar, however the point coverage for the latter is lower. Similarly, for the triangle and circle shapes, where the circle has a slightly higher point coverage value. Point coverage is a normalised value and is computed by tracing orthogonal rays from a moving virtual camera above the area spanned by the OBB. A low discrepancy sequence, generating a number of camera positions above the OBB, is used to make sure that sampling is distributed over the OBB. Point coverage is calculated as the ratio of rays traced from the moving camera over the number of point intersections. If no intersections occur, points coverage for the segment is assigned a value of 0, whereas if all rays intersect a point in the OBB, point coverage for the segment is assigned a value of 1.

The bottom left hand side image of Figure 7.5 shows an up-sampled version (§2.6.2), from ∼200K to ∼600K points, of the middle row point cloud. Clearly, simply increasing the number of points by interpolation does not contribute additional information. Note however, that there might be situations where up-

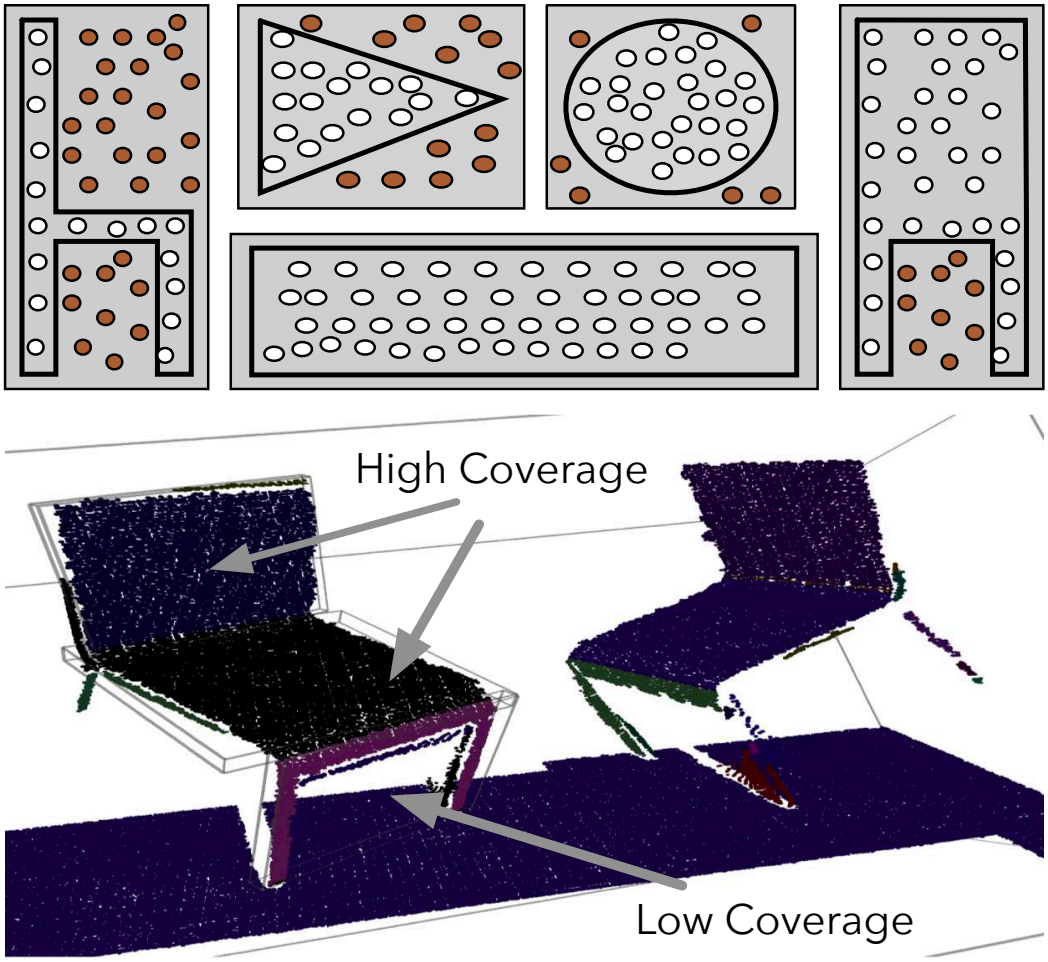


Figure 7.10: An OBB does not provide information about the distribution of enclosed points. Coverage is used to provide an indication.

sampling can affect the outcome of PaRSe. If interpolated points are added over a planar surface, for instance the top of a table, then these will not make any difference in terms of segments produced but only increase the number of points in specific segments. However, when interpolating points on a curved surface, additional planar segments may be introduced thus affecting the resultant set partition and the mappings carried out by CoFFrS. In the results section, no up-sampling is carried out and CoFFrS is directly applied on the raw point clouds produced by the scanner.

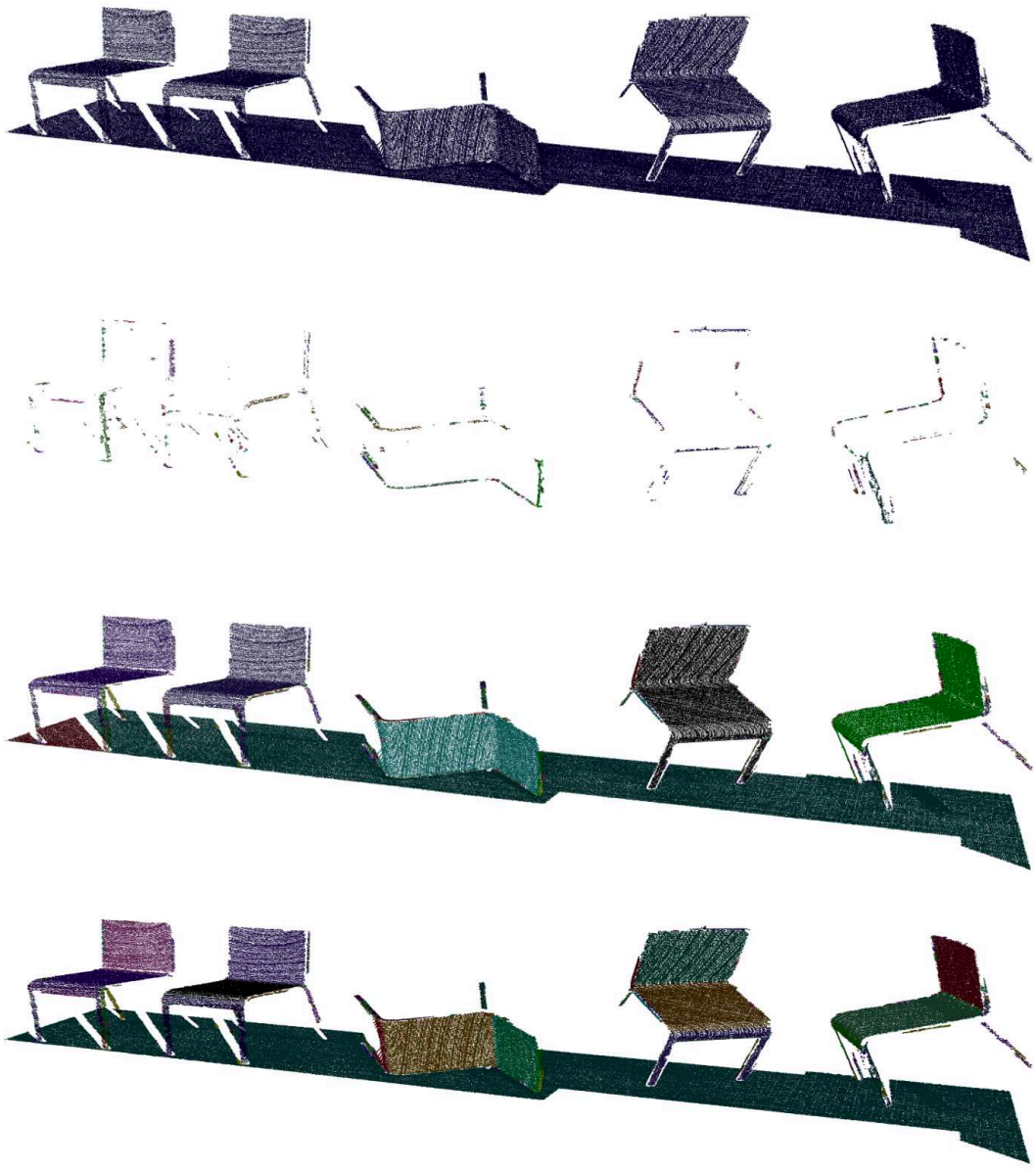


Figure 7.11: Example point cloud from Nan *et al.* (2012) representing five chairs with different poses. Point in raw data are first assigned to either P_s or P_e. Second image from top is showing points in P_e. Region growing process generates the regions shown using different colours in the third image and finally RanSaC produces $*.planar$ segments.

7.3 Object Graphs

Structure graphs form the basis of the object descriptor used in CoFFrS. Their topology remains unchanged irrespective of pose changes for a specific object,

i.e the computation of the structure graph is unaffected by scale, rotation and translation of an object. The structure graph for a chair on a table is identical to the structure graph for the same chair toppled on the floor. Moreover, a single structure graph can represent multiple similar objects (e.g. chairs with different non-uniform scaling factors) and can easily tolerate noise in a point cloud as long as the resulting segments are similar. Manually crafted query graphs (§5.4) have been used to search for specific patterns in structure graphs. For instance, a typical flight of stairs in a room can be described as a sequence of connected orthogonal planar segments. In this section an object descriptor is introduced, namely the object graph, which is an extension to query graphs and enables the identification of previously trained objects in a point cloud. In addition to properties of query graphs, object graphs include:

- the automatic determination of *anchor* segments as the most salient/visible segments in an object point cloud.
- the computation of a voxel grid around each of these anchor segments, describing the shape of the object with respect to each anchor.

In order to apply object graphs to the structure graph of a target point cloud, in a similar fashion to query graphs, a root node needs to be identified in the object graph. In the case of query graphs this is done manually, however in the case of object graphs the selection of the root node is automated. The set of $* \cdot planar$ segments resulting from PaRSe, are not all equally important when trying to identify an object. For instance, if one of the four legs of a chair is occluded and therefore not sampled, one can typically still recognise a chair. On the other hand, if the back of the seat is occluded then it is much harder to recognise. Moreover, the larger the segment, typically, the lower the potential for total occlusion of the segment. For these reasons, segment saliency is established by taking in consideration both number of points and OBB point coverage. Specifically, the saliency of a segment s is computed as:

$$s.saliency = s.point\ count * s.OBB\ point\ coverage$$

Since OBB point coverage ranges between 0 and 1, this metric favours segments which have higher point coverage given the same point count. Intuitively, the higher the saliency score, the higher the probability of the surface segment being visible in the target scene. Currently, three anchor segments are selected,

with additional transitions added to the object graph transition function connecting these anchor segments which together define the *support* of the object. The object support represents local (to the object trained) planar segment connectivity which is used to quickly give an indication of whether an object is present in the target scene.

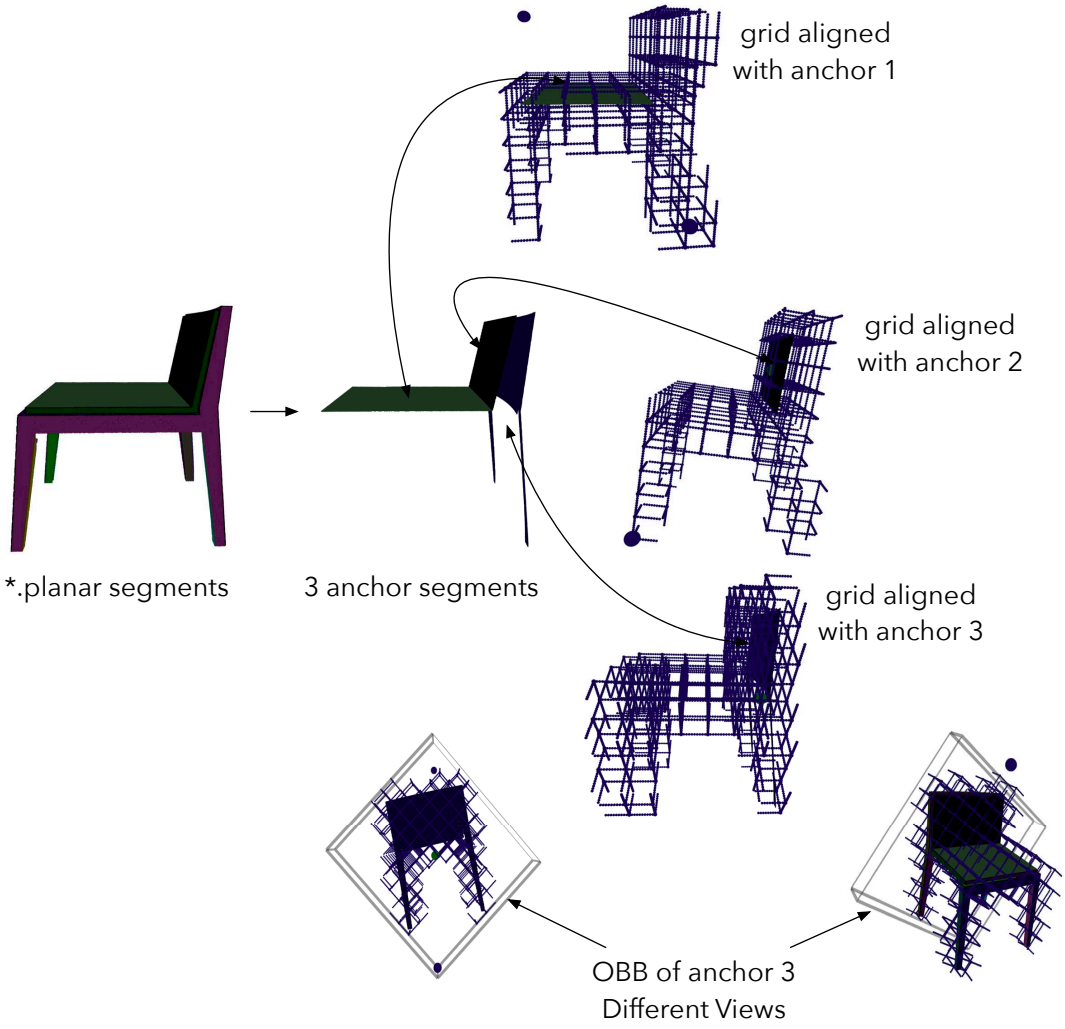


Figure 7.12: A chair 3D model is scanned and segmented. Three anchor segments are then chosen, depending on the saliency score and grids computed around these segments. The images at the bottom shows the OBB of the 3rd anchor segments from two different views, highlighting how the grid is aligned with the OBB.

In addition to connectivity information, a voxel grid is computed around each anchor segment. Each grid approximates the shape of the object around the anchor segment and is used whilst searching to determine whether the segments identified using an object graph actually correspond to that object. The grid is

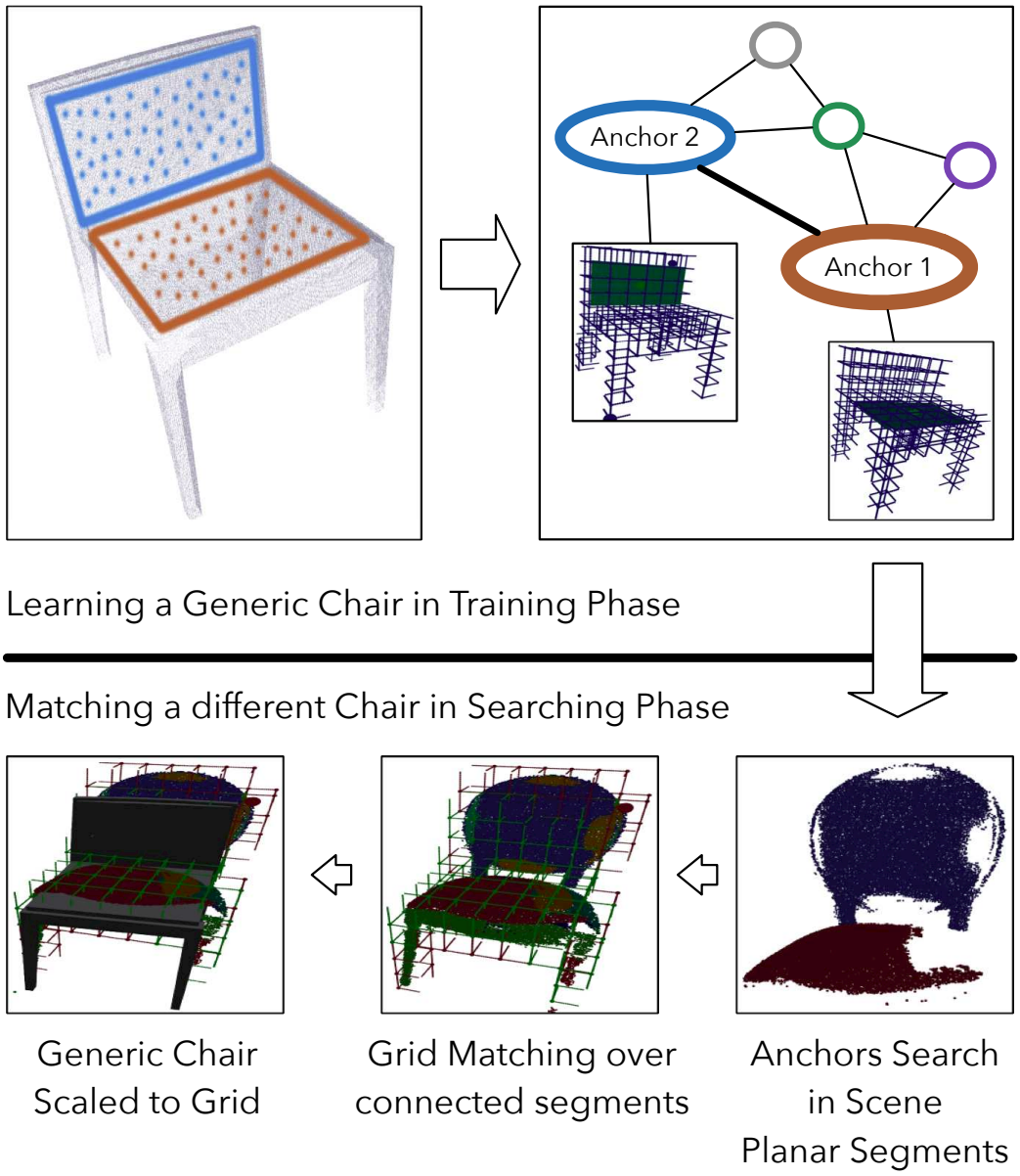


Figure 7.13: Connectivity between anchor segments in an objects' structure graph is used to locate similar objects in a target scene.

oriented in world-space by the orthonormal basis formed by the anchor segment OBB. Figure 7.12 illustrates a chair point cloud together with the three segments having the highest saliency score and therefore selected as anchors. Grids are rendered around each of the anchor segments. The bottom two images illustrate the OBB of the third segment, which includes the back of the chair and the two back facing surfaces of the legs. Note how the computed OBB (using PCA as

described in §2.5.3) does not always result in an ideal bounding volume. Clearly, rotating the OBB of the anchor segment by a few degrees around the surface normal of the anchor segment, would result in different cells of the voxel grid being active. This problem is addressed in the design of the search and matching process (§7.4), which ensures that the OBBs computed for the segments in the target scene can be matched with those in the training process. Whereas increasing the grid resolution improves the grid approximation to the shape of the object, in order to improve search performance and make it more generic, a low resolution grid is used to only capture the salient shape features without capturing too much detail. Each grid cell stores information about which segments are present in it. In our case-studies, the number of cells along the three axes of the grid are fixed across all objects and are manually chosen to best fit the models used.

Figure 7.13 illustrates the two extensions carried out on the query graph. The top left image, shows the point cloud of a chair with two segments (back and seat) selected as anchor segments. The third anchor segment, the other side of the back of the seat, is not visible. Clearly, many different chairs exist which are similar in shape. As illustrated in the bottom row of Figure 7.13, in cases where different chairs are present in a target scene for which that specific chair descriptor was not previously trained, as shown in this example, CoFFrS is designed to fit a trained generic chair which has similar anchor segments and voxel grids.

A training process is carried out to automatically synthesise object graphs. Algorithm 13 illustrates the steps involved. The algorithm takes as input the 3D model of the object, virtual camera, segmentation and training parameters and a random sampler, and returns an object graph describing the input 3D model. A point cloud of the object is acquired by casting rays from a virtual camera positioned around the 3D model and storing depth information. A cosine hemisphere sampler is used to position the camera in different positions. Figure 7.14 shows the depth images produced by the virtual scanning process. Different structure graphs are computed for each partial point cloud and for the point cloud resulting from merging the points in the different views. Instead of using a 3D object, from which a point cloud is produced, the training process can be directly applied on a scanner acquired point cloud of a real object. After applying PaRSe on the object point cloud, three anchor segments are chosen according to their saliency values and voxel grids are computed for each. The OBB of the entire point cloud of the object is used to determine the size of the voxel cells, given as input the number of cells along the three orthonormal vectors of the OBB. This training

Algorithm 13 Training of object graph.

Input: 3D mesh of object o, virtual camera c at distance d from o, point cloud segmentation parameters σ, number of scans n_{scans}, empty list of point clouds l of size n, empty list of structure graphs s of size n, random sampler rs, object graph $og = \{\mathcal{N}, \mathcal{E}\}$, segment saliency heuristics h, empty list of anchor segments $anchors$ of size $n_{anchors}$.
▷ Generate point clouds of object from n different views
for $k = 1$ to n_{scans} **do**
    $sample = rs$.Get2DSample()
    $c_{pos} = d$ * CosineSampleHemisphere($sample.X$, $sample.Y$)
    $c_{look} = Vector3(0, 0, 0)$
    $l_k \Leftarrow^{add} c$.RenderDepth
end for
    ▷ Generate structure graphs for all elements in l and create p for object.
for $k = 1$ to n_{scans} **do**
    $s_k \Leftarrow^{add}$ GenerateStructureGraph(l_k, σ)
    $p \Leftarrow^{add} l_k$
end for
    ▷ Initialise og as a structure graph, establish anchors, update transition function of og and compute grids.
og = GenerateStructureGraph(p, σ)
$anchors \Leftarrow^{add} og$.EstablishAnchorSegments(h, $n_{anchors}$)
for $k = 1$ to $n_{anchors}$ **do**
    $anchors_k$.ComputeGrid()
    for $j = 1$ to $n_{anchors}$ **do**
        if $(j \neq k)$ **then**
            $\mathcal{E} \Leftarrow add(anchors_k, anchors_j)$
        end if
    end for
end for

process is carried out once for each object and the resulting object graphs are loaded whenever these are required during the searching phase.

Figure 7.14: Three depth images from virtual camera positioned around the object. The partial point clouds from each view are merged together to form the point cloud of the office chair.

7.4 Scene Understanding

This section presents the object and pattern searching phase of CoFFrS. The office room point cloud in Figure 7.4 middle row represents a typical indoor scene containing both objects (e.g. chairs, table, monitors, glasses, desk, etc.) and structures (e.g. floors, walls and shelving). Structures such as shelving, which are relatively simple to describe in terms of planar surfaces will most probably vary between rooms, for instance in the number of shelves and the vertical distance between them, and are therefore are not suitable candidates for representation using object graphs. In these cases, query graphs are used to infer their presence in a scene. On the other hand, chairs and tables cannot be easily described in terms of seat, back, and legs. In these cases, the object graph descriptors of these objects are used. By using both query and object graphs, CoFFrS seeks to produce a rotation and scale invariant scene understanding process. Previous work (e.g. Nan *et al.* (2012) and Lin *et al.* (2013)) has produced solutions which target very specific environments, which on one hand make them very efficient within that specific environment, but on the other limit their adaptability to other scenarios. In the approach proposed, CoFFrS is designed as a generic scene understanding solution which uses common segmentation and training processes

but can be adapted, at the understanding (searching) phase to unseen environments by simply changing the search mechanism to best suit the environment.

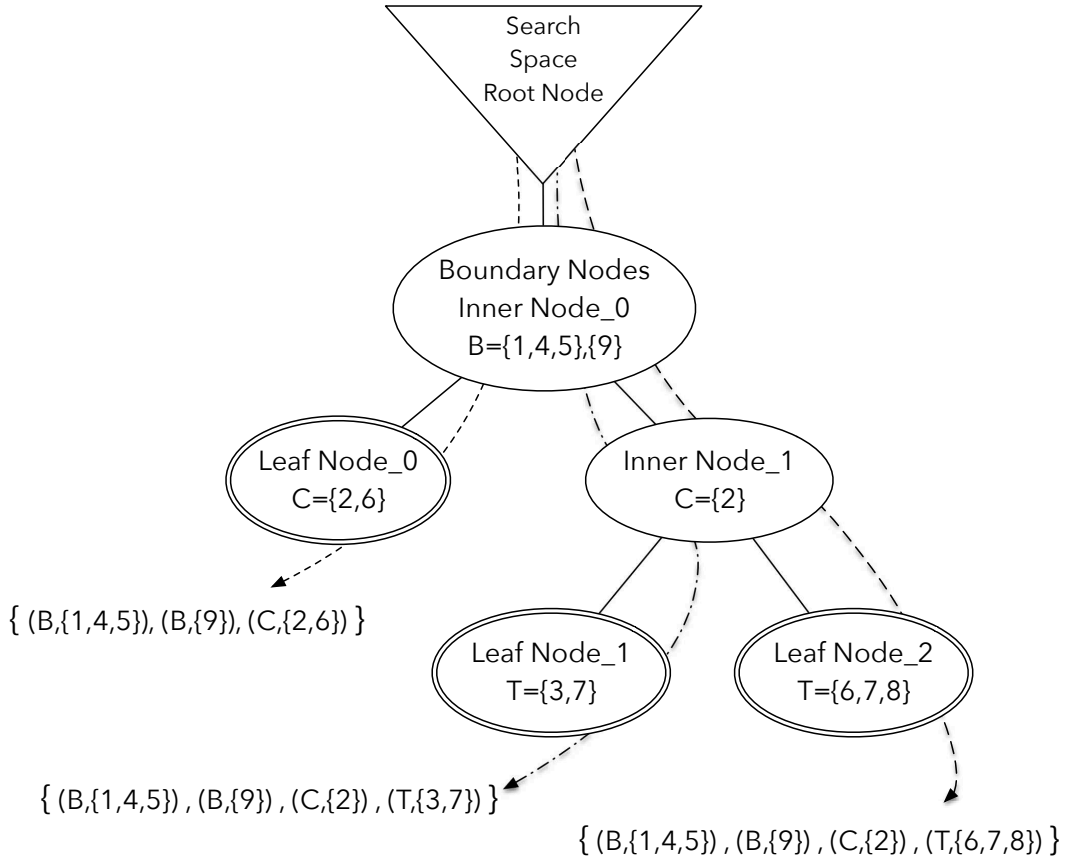


Figure 7.15: A simple Markov decision process tree illustrating a number of solutions (each a depth first traversal) to the scene understanding task.

The approach adopted by CoFFrS, reformulates the supervised scene understanding problem as one which seeks to maximise matches between anchor **·planar* segments in object descriptors and **·planar* segments in the target scene. In general, this is bound to be an unconstrained problem, especially in the presence of noise and partial object occlusions, where multiple seemingly valid mappings may exist. A solution consists of a subset of possible mappings between the set of objects used and the set of surface and edge segments in the structure graph of the target scene. Query and object graphs are used to constrain this space, whilst a Markov decision process (Puterman, 2009) is used to enumerate this search space using a number of heuristics intended to quickly provide a list of valid solutions. CoFFrS creates a solution tree $\mathcal{L}$, where each solution associates labels from object and query graphs to sets of scene segments and is obtained via

a depth first traversal of $\mathcal{L}$. Figure 7.15 shows a simple example of $\mathcal{L}$, where planar segments with ids 1...9 in the target scene are mapped to objects C (chair), T (table), and boundary structures (e.g. wall, floor, ceiling) B. Each of the three leaf nodes describe a solution, whose score is an aggregate of the scores obtained at all inner nodes (individual object mappings) along each depth first traversal path. In order to decrease the number of solutions, a constraint on the number of inner node children can be imposed. If this parameter is set to one, as is the case with the examples presented in the results section, then only one solution is produced. The Markov decision process model is set up as follows:

- A set of possible world states $S = \mathcal{P}(P_s \cup P_e) \times L$,
- A set of possible actions A,
- A real valued reward function $R(s, a)$ which gives a score for the transition from s to the state returned by the function when following action a,
- A total function T, such that for each $s \in S$ and action $a \in A$, a new $s' \in S$ is defined,

where P_s, P_e represent all the surface and edge segments respectively produced by the segmentation process, $L = labels(G_{obj}) \cup labels(G_{query}) \cup \{unidentified\}$ is the set of labels, G_{obj} and G_{query} represent the sets of object and query graphs respectively. Each $g_{obj} \in G_{obj}$ and $g_{query} \in G_{query}$ is assigned a unique label and $labels(G_{obj})$, $labels(G_{query})$ return the labels of the sets G_{obj} of object graphs and G_{query} of query graphs used during the searching phase. Consider the sets $L = \{chair, floor, unidentified\}$, $P_s = \{P_s^0, P_s^1, P_s^2\}$ and $P_e = \{P_e^0\}$. The set S of possible world states is as follows:

$$
\begin{aligned}
\mathcal{P}(P_s \cup P_e) =& \{\emptyset, \{P_s^0\}, \{P_s^1\}, \{P_s^2\}, \{P_e^0\}, \{P_s^0, P_s^1\}, \{P_s^0, P_s^2\}, \{P_s^0, P_e^0\}, \\
& \{P_s^1, P_s^2\}, \{P_s^1, P_e^0\}, \{P_s^2, P_e^1\}, \ldots, \{P_s^0, P_s^1, P_s^2, P_e^1\}\}. \\
S =& \{(chair, \emptyset), (chair, \{P_s^0\}), \ldots, (chair, \{P_s^0, P_e^0\}), \ldots, \\
& (chair, \{P_s^0, P_s^1, P_s^2\}), \ldots, (floor, \emptyset), (floor, \{P_s^0\}), \ldots, \\
& (floor, \{P_s^0, P_s^1, P_s^2, P_e^0\}), \ldots, (unidentified, \{P_s^0, P_s^1, P_s^2, P_e^0\})\}.
\end{aligned}
$$

The solution space consists of a tree enumerating subsets of S such that a path is traced from the root of the search space, where all segments are without a label,

to a leaf node where all segments are now given a label. Clearly, there can exist only one optimal solution which correctly labels each point, and this *may* not even be included in the solution space, unless each segment is actually made up of only one point. If a particular segment includes points located on two distinct objects with different labels, then the optimal solution is not contained in the space, since some of the points will always be labelled incorrectly. In this sense, over-segmentation of the input point cloud is preferred over under-segmentation. On the other hand, the run-time of the matching process depends on the time taken to compute function $R(s, a)$, and therefore increasing the number of segments, results in an increase in the time required to compute solutions.

Two elements are included in the set of actions A, namely the application of either a query or an object graph on the structure graph of the input point cloud. Paths are traced in this solution space, via label assignment of segments and is decided via the application of these actions and the reward function. This effectively implements the function T between states in S as follows:

Definition Given the set $P_{se} = \{p_{se} : P_s \cup P_e | noLabel(p_{se}) \cdot p_{se}\}$, $s \in \mathcal{P}(P_{se})$, actions $a \in A$, a reward function $R(s, a)$, the next state $s_{nxt} \in S$ satisfies the following set comprehension:

$$s_{nxt} = \{a : A, s : \mathcal{P}(P_{se}) | max(R(s, a)) \cdot (label(a), s)\}$$

The transition to the next state s_{nxt} labels the chosen segments in s with the label a and effectively removes them from being considered in the next application of the reward function at s_{nxt}. In practice, function $max()$ may be replaced by a function which returns the set of the best n scores. Branching in the solution tree is introduced by making use of this set of n (greater than 1) of best scores. Algorithm 14 outlines the steps carried out in building the solution tree. The design of function R is obviously key to the success of the understanding process and is described in more detail in Algorithm 16. The scene understanding process, starts building the solution tree by initialising a root node which includes all segments in P_{se}. The output of the process, consists of at least one set partition associating labels of objects in G_{obj} and structures in G_{query} with subsets of these segments. The search algorithm is composed of two main consecutive phases, first applying an unsupervised process to determine general structures such as boundaries, and then applying a supervised process to match the rest of the segments with previously trained object graphs. The unsupervised phase consists

Algorithm 15 Construction of Solution Tree using Markov Decision Process

Input: Set of edge and surface segments P_{se}, structure graph $\mathcal{G}$ of input point cloud, empty solution tree $\mathcal{L}$, set G_{query} of query graphs, set G_{obj} of object graphs, empty lists $segs_{query}$ and $segs_{obj}$ of segments, empty list $nodes_{leaf}$ of leaf nodes in $\mathcal{L}$, list $scores$ of triples (scr, g_{obj}, $segs$) where $g_{obj} \in G_{obj}$ and $segs \subseteq P_{se}$, minimum score value scr_{min}.

▷ Initialise solution tree with root node.

l_{root} = CreateTreeNode(P_{se})
$\mathcal{L}$.SetRootNode(l_{root})

▷ Apply query graphs in sequence.

$nodes_{leaf} \leftarrow \mathcal{L}$.GetLeafNodes
for all $l_{leaf} \in nodes_{leaf}$ **do**
    $node_{crt} = l_{leaf}$
    for all $g_{query} \in G_{query}$ **do**
        $segs_{query}$ = ApplyQuery(g_{query},$\mathcal{G}$,P_{se})
        while $|segs_{query}| > 0$ **do**
            AssignLabel($segs_{query}$, label(g_{query}))
            $P_{se} = P_{se} \setminus segs_{query}$
            $node_{new}$ = CreateTreeNode(P_{se})
            Connect $node_{new}$ to parent node new_{crt}
            $segs_{query}$ = ApplyQuery(g_{query},$\mathcal{G}$,P_{se})
        end while
    end for
end for

▷ Match remaining unlabelled segments P_{se} to object graphs.

$sortedSegments$ = Sort(P_{se})
for all $s \in sortedSegments$ **do**
    $chosenGraphs$ = PatternMatch(G_{obj}, s)
    for all $g_{obj} \in chosenGraphs$ **do**
        $scores \leftarrow^{add}$ R(g_{obj}, s) ▷ see Algorithm 16
    end for
    for all $(scr, g_{obj}, segs) \in scores$ **do**
        if $(scr > scr_{min})$ **then**
            $nodes_{leaf} = \mathcal{L}$.GetLeafNodes
            for all $l_{leaf} \in nodes_{leaf}$ **do**
                AssignLabel($segs$, label(g_{obj}))
                $node_{new}$ = CreateTreeNode($P_{se} \setminus segs$)
                if Compatible($node_{new}$, Path(l_{root}, l_{leaf})) **then**
                    Connect $node_{new}$ to parent node l_{leaf}
                end if
            end for
        end if
    end for
end for

▷ Label remaining unlabelled segments P_{se} to unidentified.

of the application of query graphs in a specific order. Order is important, since segments are removed from further processing when labelled. Query graph application order is one of the parameters which is set by the user. In general, and if not otherwise specified in the evaluation section, three query graphs are used for indoor scenes. These include queries to identify shelving, stairs and boundaries. In all examples, the boundaries query graph is used just before the application of object graphs and tries to match the floor, ceiling or walls of a scene. Prior to determining boundaries, the shelving and stairs query graphs try to determine if there are any shelving units or stairs in the scene. Note that the application of these query graphs is carried out more than once (line 9 in Algorithm 14) until the query returns an empty list of segments. This is especially important for the shelving query graph, where multiple instances of shelving units may be present. Note that as opposed to query graphs, for instance the cylinder query graph (Figure 5.22), only one instance of the query is returned and attached to a new node in the solution tree. The main intuition behind the application of the query graphs before object graphs, is that in general, structures such as walls, floors, shelving units, stairs, etc. consist of a high percentage of points in the point cloud. Removing these points from consideration by the second phase, decreases the run-time required to produce the solution tree.

The second phase of CoFFrS searching process, first enumerates the remaining unlabelled segments and then tries matching them with anchor segments of objects graphs in G_{obj}. Segments in the scene are ordered by their saliency score, which is computed using the same criteria used when choosing anchor segments for object graphs. The order in which $*{\cdot}planar$ segments are matched with anchor segments plays a critical part in the correctness of the scene understanding process, since currently, segment labelling cannot be reverted. Therefore, if domain-specific knowledge of the target environment such as the distance from the floor of the chair seats and table tops is known, then a segment sorting function, in addition to saliency scores, can order horizontal planar segments according to their distance from the floor and try to match these with tables and chairs first. In order to provide for a generic scene understanding solution, CoFFrS allows for different sorting function implementations to determine the sequence by which $*{\cdot}planar$ segments from the target scene are visited. If no domain-specific information is available, $*{\cdot}planar$ segments are sorted according to their saliency scores. At the end of the process all segments are either labelled as part of specific objects, or identified as groups of segments which require further processing

using other point-based object recognition techniques (§4.1.1).

In theory, each **·planar* segment may be matched against each anchor segments of all object graphs in G_{obj}. In practice however, in order to improve matching/searching efficiency, only a subset of objects graphs in G_{obj} are considered. This subset is determined by function $PatternMatch(G_{obj}, s)$ which establishes which anchor segments in each of the object graphs is closest to the connectivity pattern around s. A variety of segment properties may be used to determine which object graphs to include in *chosenGraphs* (line 20 of Algorithm 14). For the example using in the work, the angle between s and connected segments together with the presence of other segments matching in orientation the anchor segments of a specific object graph are used.

Whereas anchor connectivity information is used to determine which object graphs to consider for labelling segment s, coarse resolution voxel grids created around segment a are used to determine which other scene segments make up the object and further discriminate between similar objects (e.g. two different chairs). Whilst the matching of voxel grids is relatively expensive, this additional data contributes important discriminatory information to the description of each object. The reward function R (line 22 of Algorithm 14) is implemented as an incremental grid matching process. It is used to determine which segments surrounding segment s in the scene best fit within the trained objects' voxel grid. Algorithm 16 describes the steps involved in computing the reward score given a set of segments in P_{se} and an object graph in *chosenGraphs*. At each step, a grid is computed around the segment matching the anchor and enclosing a number of connected segments. Grid compatibility measures point distribution similarities around the two matching segments and can be defined in a variety of ways. A compatibility score between scene and object grids can be as straightforward as calculating the set intersection between the two grids or else make use of some additional heuristics. Non-uniform scaling and rotations around the normal of s are performed until all points in the segments being tested are included. If the score decreases when adding a new connected segment, this is removed and other segments are added according to the structure graph of the target scene. Finally, when the best scene voxel grid is chosen, additional **·complex* segments from the connectivity graph connected to those in P_{se} are selected and any which fall within the OBB of the scene voxel grid are tested to check whether they consolidate the match. If the distance between two mappings is small (user-set parameter), a tie-breaker function is used to select the object mapping which

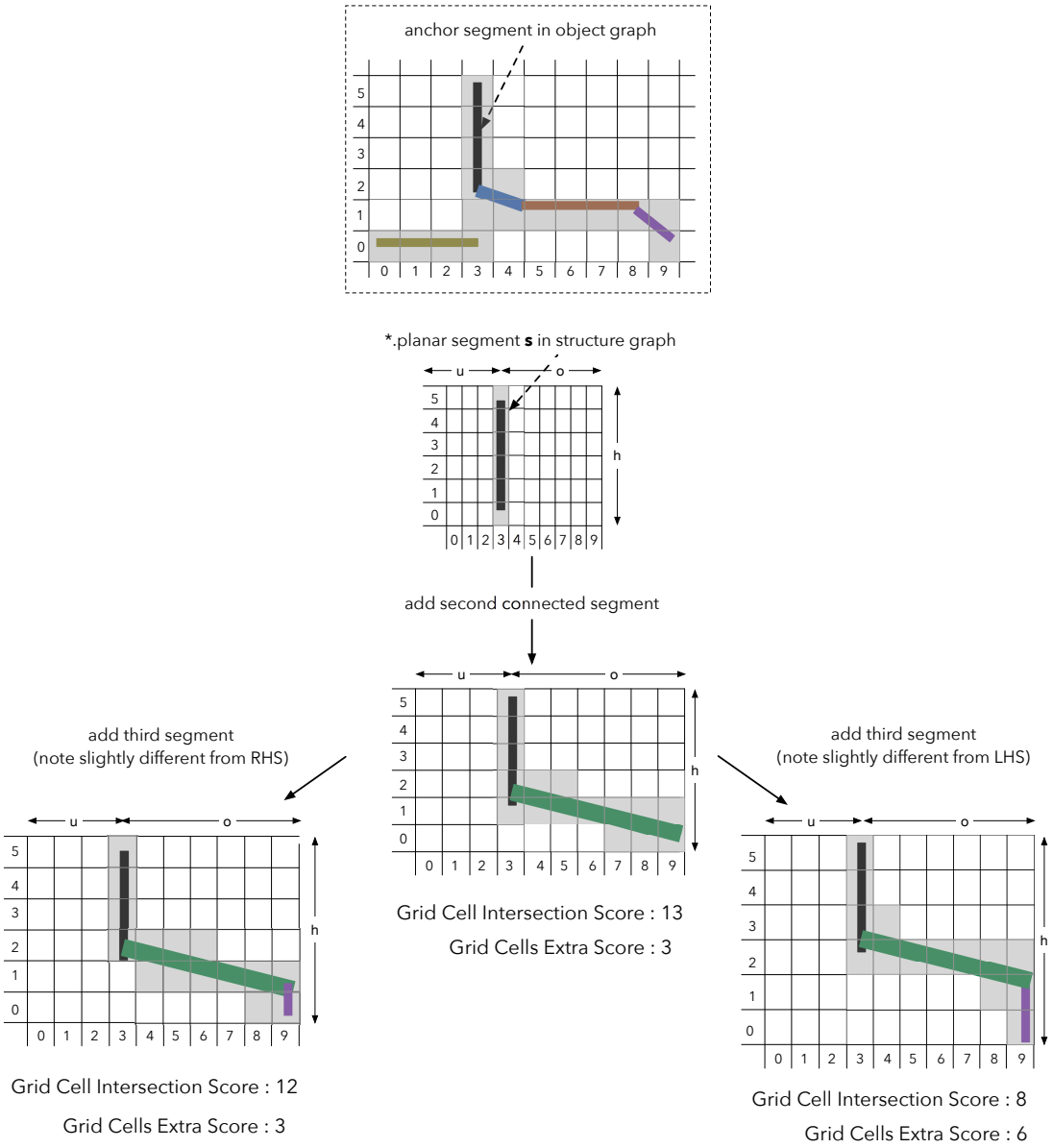


Figure 7.16: Top row shows 2D grid computed around the anchor segment of an object graph consisting of five segments. This object representation is compared to segments in a structure graph, in this case matching s is with the object anchor by iteratively computing grids around s and computing the distance between the two grids. Even a very small difference (two grids of last row), makes a considerable difference when computing the distance between the grids as a set intersection of occupied cells.

according to some heuristic has the highest probability of occurring, e.g. always prefer upright pose. The tie-breaker may not be used if multiple solutions are allowed, in which case the different mappings are attached as separate leaf nodes to $\mathcal{L}$. CoFFrS allows for and can take advantage of any additional constraints

available, for instance scene specific knowledge, to improve results.

Figure 7.16 demonstrates an example of a grid matching sequence. For illustration purposes a simpler 2D grid is used in order to better highlight a number of characteristics of this process. In the case of 3D grids, additional work is carried out to compute grids at small angle increments around the surface normal of segment s. The reward function R generates all these grids, centred as s in the target scene graph matching the coordinates in the trained voxel grid of the object graph. For each set of segments currently being tested P_{test} and angle θ, distances u and o are increased until all the points from segments in P_{test} are included and cell sizes computed accordingly (in this case (u+o) / 10). A grid scoring function is then used to determine the similarity between grids. In order to decrease computation time, when computing the grids around s, each segment in P_{se} is sub-sampled using a Poisson-Disc sampling approach (§2.6.1). Each of these points is then used to populate the sparse voxel grid constructed around segment s. The scoring function used in this example, and in the evaluation, makes use of the grid cell intersection set defined as follows:

$$cells_{cmn} = \{c_s : g_s | c_s \in g_{anch} \cdot c_s\},$$

where grids g_{anch} and g_s represent the object graph and structure graphs grids respectively, and c_s represents the spatial index along the three dimensions of a cell. Segments in P_{test} may or may not be contained within a subset of cells in $cells_{cmn}$. Using matching subsets, corresponding segments in the trained object graph are extracted in order to compare the OBB point coverage scores for matching segments. These values are factored into the grid intersection score in order to discriminate between pairwise matches as illustrated in Figure 7.17. If no pairwise segment match is determined for a particular segment in P_{test}, the point coverage for that specific segment in the point cloud is not used. In addition to point coverage, segment spatial context is also factored in the computation of the score between grids. Segment spatial context (Figure 7.9) is re-computed for each segment in P_{test} each time this set changes and provides additional confidence in the grid matching process. The scoring function, given grids g_{anch}, g_s, $cells_{cmn}$ and a list of pairs m of matching segments (s_{obj}, s_{pc}) with the first component chosen from the trained object graph and second from the point cloud structure graph is as follows:

$$score(g_{anch}, g_s) = |cells_{cmn}| \times \sum_{i=1}^{|m|} \left(\frac{s^i_{pc}.pointCoverage}{s^i_{obj}.pointCoverage} \times spatialContextSc \right),$$

where $spatialContextSc$ is set to 1 if segments s^i_{pc} and s^i_{obj} are the same and set to 0.8 if either of the two segments is *central* and the other *boundary*. If any one of the segments is set to *boundarycentral*, this value is set to 0.9. When all grid comparisons are done, the reward function R returns the segments P_{test} and matching object graph label with the highest grid matching score. This score is used as the reward score and assigned to the new state node attached to the solution lattice (line 31 Algorithm 14). The new node is only attached to leave nodes which are compatible with the path leading to that node. A leaf node is compatible if the segments used in P_{test} are not already labelled anywhere else in the path from the leaf node to the root of the solution tree. In cases where only one leaf node is allowed, compatibility can be guaranteed during the computation of the reward function R by only including segments in the transition tree of s which have not already been labelled (line 3 in Algorithm 16).

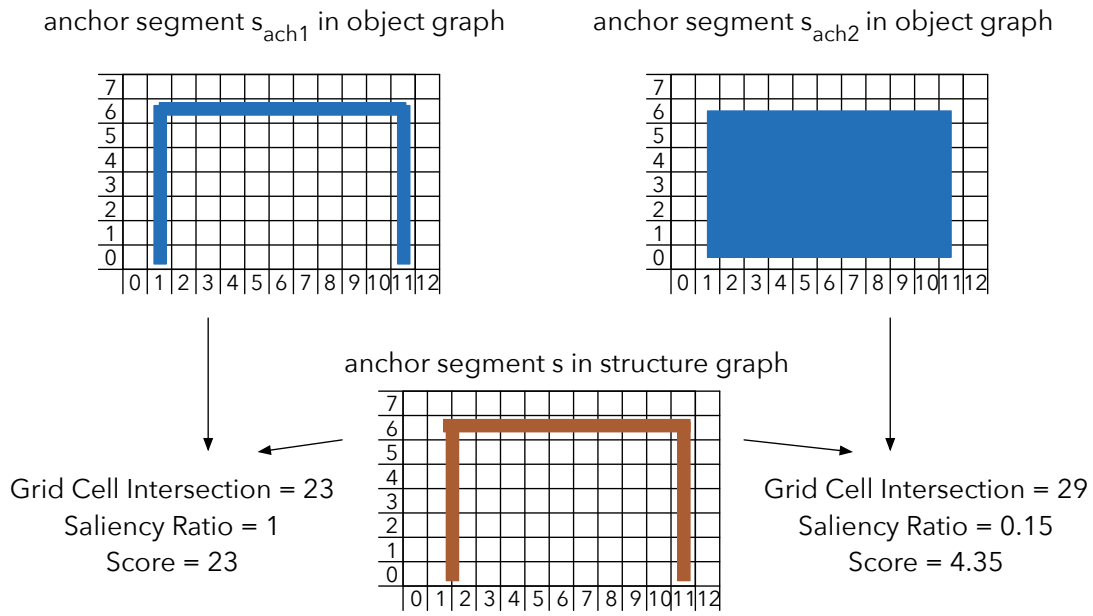


Figure 7.17: Top row shows two anchor segments possibly from the same object graph which are matched with segment s in the point cloud structure graph. Saliency values favour the anchor most similar to s on the assumption that both anchor segments are enclosed by similar OBB.

The reward function R (Algorithm 16), returns a list of 6-tuple elements containing information about the best grid matching scores for segment s and object graph g_{obj}. The assignment of P_{test} is carried out using the transition tree computed at s by iteratively adding segments from the tree which maximise the score of the grid matching function. Figure 7.18 illustrates an example of

Algorithm 16 Computation of reward score for function $R(s, g_{obj})$

Input: Object graph g_{obj}, structure graph $\mathcal{G}$ of input point cloud, segment s to match with object graphs in a, list $scores_{obj}$ of values (scr, g_{obj}, $segs$, θ, $scale$) where $g_{obj} \in G_{obj}$ and $segs \subseteq P_{se}$, θ and $scale$ store the rotation and scale parameters for the score, rotation increments rot, empty list of segments P_{se}, maximum depth l for transition tree tt_s at s.
$\theta = 360$ / rot
tt_s = CreateTransitionTreeAt(s,l)
for all $g_{obj} \in a$ **do**
    $chosenAnchors$ = PatternMatch(g_{obj}, s)
    for all $anchor \in chosenAnchors$ **do**
        for i=0 to rot **do**
            $P_{se} \leftarrow^{add} s$
            $score_{best} = 0$;
            $rot_{best} = 0$;
            $segs_{best} = \emptyset$
            while $MoreSegsToConsider$ **do**
                g_s = CreateGrid(s, $anchor$, θ*i, P_{se})
                $score_{crt}$ = Score(g_{obj}, g_s)
                if $score_{crt} \geq score_{best}$ **then**
                    $score_{best} = score_{crt}$
                    $rot_{best} = rot$
                    $segs_{best} = P_{se}$
                    $anchor_{best} = anchor$
                else
                    $P_{se} \leftarrow^{remove} n$
                end if
                $n = tt_s.ChooseNextSegment$
                $P_{se} \leftarrow^{add} n$
            end while
        end for
        $scores_{obj} \leftarrow^{add} (score_{best}, g_{obj}, anchor_{best}, P_{se}, rot_{best}, scale)$
    end for
end for

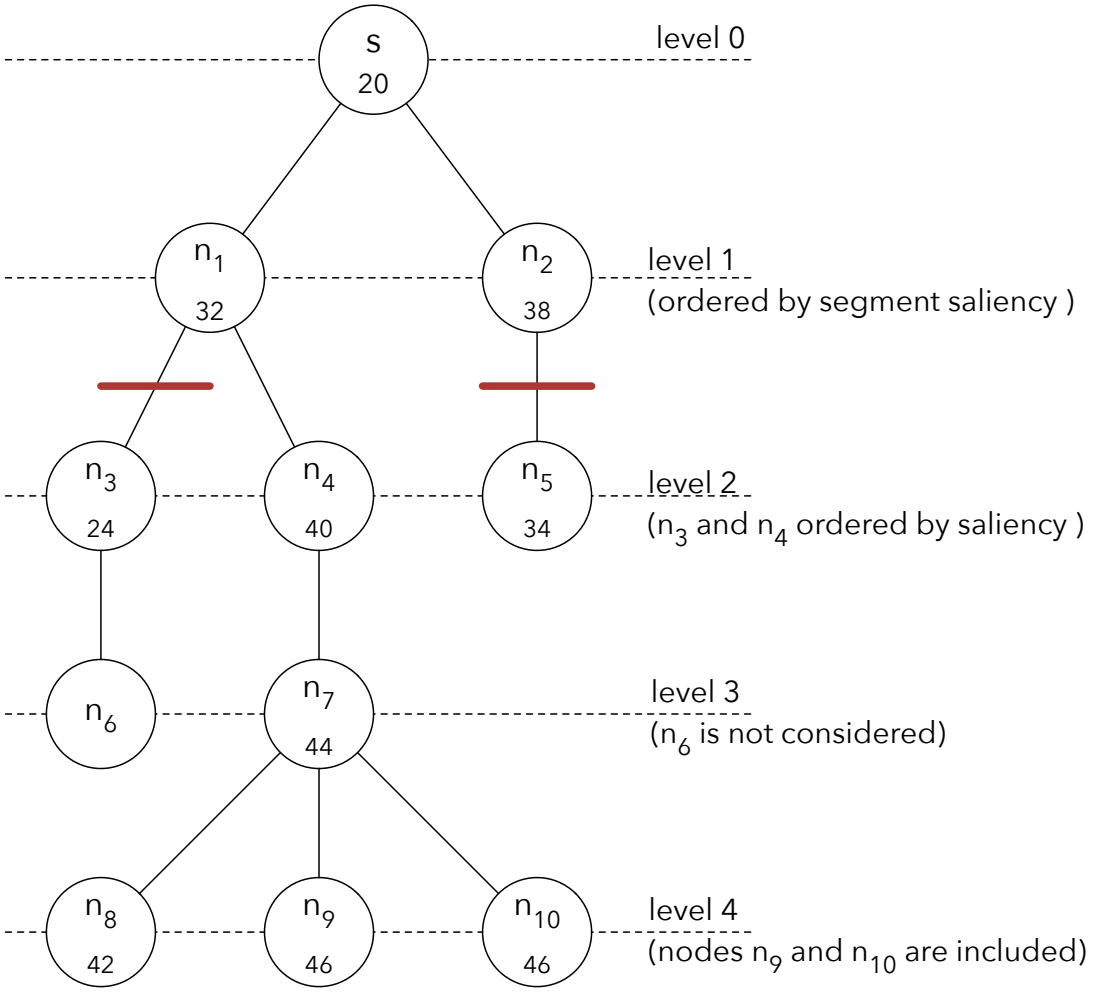


Figure 7.18: Segments in P_{se} are determined by traversing the transition tree computed at s. All child nodes are ordered by segment saliency.

a transition tree with sample scores included in each node. At level 0, the grid containing only segment s has a score of 20. The function $ChooseNextSegment$ (line 23 in Algorithm 16) implements a breadth first traversal search for the next segment to consider. If the score obtained is less then the current best score that path is abandoned and the segment removed from P_{test}. Since not all combinations of segments in the tree are tested, child node order is important. Segment saliency is used for ordering child nodes with the intention of including the more important segments earlier. In this example, the next segment at level 1 is n_1, followed by n_2. In both cases the score improves and therefore both segments are retained. At level 2, when P_{test} is $\{s, n_1, n_2, n_3\}$, the score is lower and therefore segment n_3 is removed from the set. Note that segment n_6 is also not considered since this is only reachable from n_3. The final composition of P_{test}

is $\{s, n_1, n_2, n_4, n_7, n_8, n_9, n_{10}\}$.

Exhaustively comparing every segment in the structure graph of the input point cloud with all anchors of all object graphs is impractical and would clearly impact the scalability of CoFFrS when increasing the number of objects graphs. For this purpose only a subset of object graphs is tested as outlined in line 20 of Algorithm 14. Moreover, the solution tree generation process does not simultaneously check multiple segments from the point cloud structure graph against different object graphs, but rather first sorts them (line 18 of Algorithm 14), and then chooses the best matching pair (s,g_{obj}) for the next s in the sorted list. An alternative solution could consider all segments s simultaneously and choose the best matching pairs, but this clearly would have an impact of the running time of the process. In general, the segment sorting mechanism used, which is based on the saliency of segments, gives very good results.

Some objects, for instance a box-like cabinet with closed drawers, may result in a good labelling of the segments making up the object, but result in a wrong pose (in this case 90° rotations), given that different poses are possible for the same set of segments. Similarly, the back and seat of a chair where only the back and the seat surface patches are sampled may be swapped. In the case of tables, if only the surface is sampled, with no leg segments present, CoFFrS takes the assumption that the legs are oriented towards the closest boundary segment, which is always the floor of the room in the examples used.

7.5 Results

In this section, a prototype implementation of CoFFrS is evaluated on a number of different scenarios consisting of raw point clouds from previous literature and a number of newly captured indoor environments. The evaluation of CoFFrS is qualitative in design and illustrates how the system performs when no scene specific information is available. In all the examples, the set of object graphs G_{obj} contains two chairs, an armchair, two tables, a couch, a cabinet and a plant pot with a cylindrical base (see Figure 7.19).

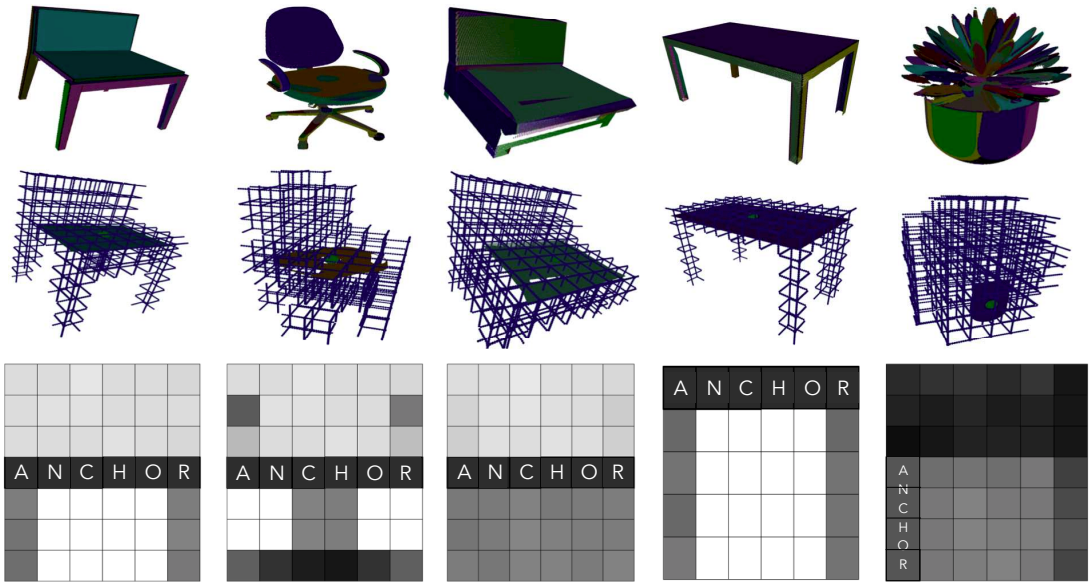


Figure 7.19: Top row shows sub-set of objects trained; middle column shows voxel grid approximating object shapes around one anchor segment and third row illustrates a density map of the grid parallel to the anchor.

Three query graphs are defined and used to search for room boundaries, shelving units and stairs in a scene. The boundaries query graph, shown in Figure 7.20, is applied to extract boundary segments in all examples. Since, no upward direction is assumed for scenes, the query graph searches through the entire structure graph and extracts those segments which have the rest of the segments contained within one of its two half-planes. Since segments are already labelled as *boundary*, *central*, or *centralboundary*, this check is only carried out on *boundary* segments. Given a segment which is identified as boundary (state s_0 in the query graph), the rest of the structure graph is visited in order to locate other segments which fit within the same plane parameters. These segments are grouped together as one of the boundaries (e.g. floor) of the room. The same query graph is applied

until no more segments in the structure graph are found which satisfy the initial properties (at node S_0) of the query. Note that following the first iteration of the query, re-application of the query still considers the previously labelled segments when checking for segments falling on either of the two half-planes.

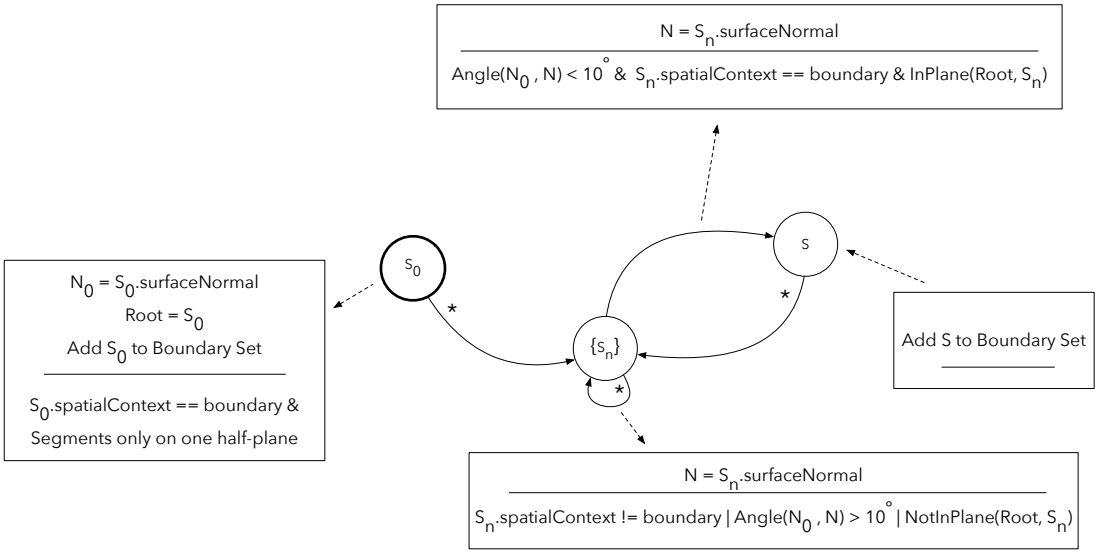


Figure 7.20: The boundary query graph is used to establish the boundary surfaces of a room, which could consists of multiple planar disconnected patches. Node S above is used to add to the list those segments which are compliant with the properties of the root node S_0.

The second query graph is used to determine whether shelving units are located in a scene. The shelving query graph, shown in Figure 7.21, is applied to each $*{\cdot}planar$ segment in the structure graph and determines whether for a particular segment, the root at S_0, there exist other parallel segments which are located within an extended OBB computed around it. The extended OBB, takes the OBB of the root and extends it marginally along the first and second eigenvectors, and considerably to include the whole scene, along the third (smallest variance) eigenvector. Some additional notation to query graphs is introduced in these definitions, namely the $*$ symbol on transitions, to denote that all transitions in the structure graph connected to the source node are followed, resulting in query graph nodes containing a set of segments rather than just one. This is denoted by the symbols { and } in the node. An iterator on these nodes then evaluates outgoing transitions on each of these segments. A shelving unit is established when a minimum number of parallel segments is found (transition between S and S_F). In this case all the segments contained in the extended OBB

of the root are included in the shelving set and labelled as one unit. Note that the sides and anything located on the shelves is included.

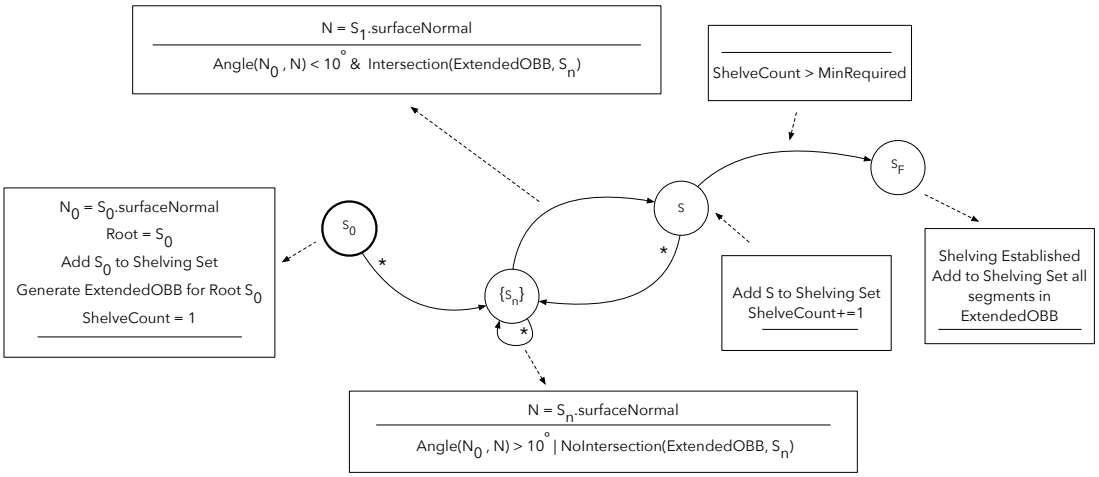


Figure 7.21: This shelving query graph is used to establish shelving units in a room, which could consists of multiple planar disconnected patches over each other.

A third query graph is used to establish whether a stairs structure is present in a scene. This query is more complex than the shelving query in that the pattern required is harder to search for. The stairs query graph, shown in Figure 7.22, is applied to each **·planar* segment of the scene structure graph and determines whether for that particular segments there exist other parallel segments adjacent and slightly above or below to its OBB. The direction for the above and below values is set to the third eigenvector (lowest variance) of the segment and the distance is set to 20% of the total scene OBB along the same direction. If at least one adjacent segment is found (transition between S and S_F), this is used to update the extended OBB which is used to determine adjacent segments. If no additional adjacent segments are found (transition between S and S_E), the query checks if the total number of stair steps is greater than a minimum required which is user specified and accordingly labels the segments as a stairs structure. The minimum number of steps is set to 3 segments.

The three query graphs are always applied in the same order; namely stairs, shelving and finally boundaries. Whereas the first two queries might not label any of the segments, the boundaries query graph always returns a set of segments representing boundaries, even if these are not present in the scene (e.g. 7.33)

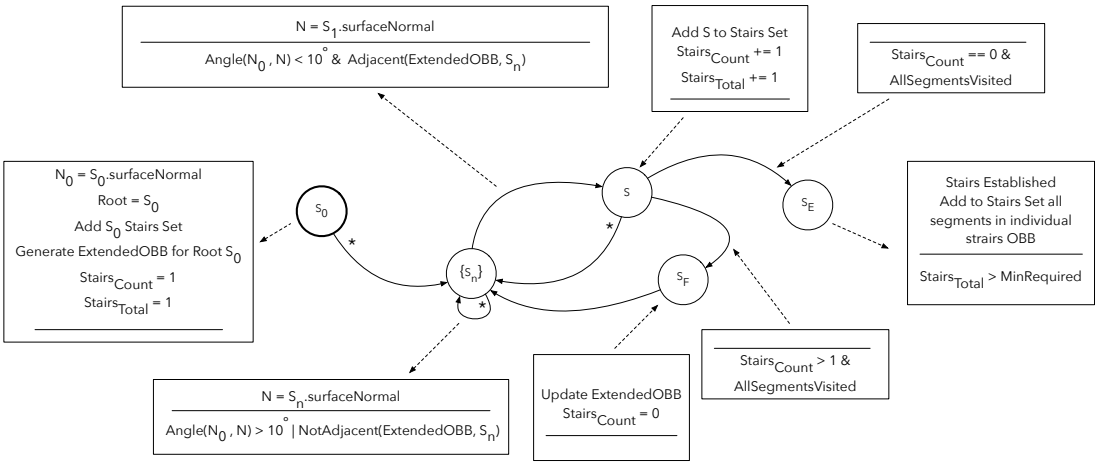


Figure 7.22: This stairs query graph is used to determine the presence of a stairs pattern in the point cloud.

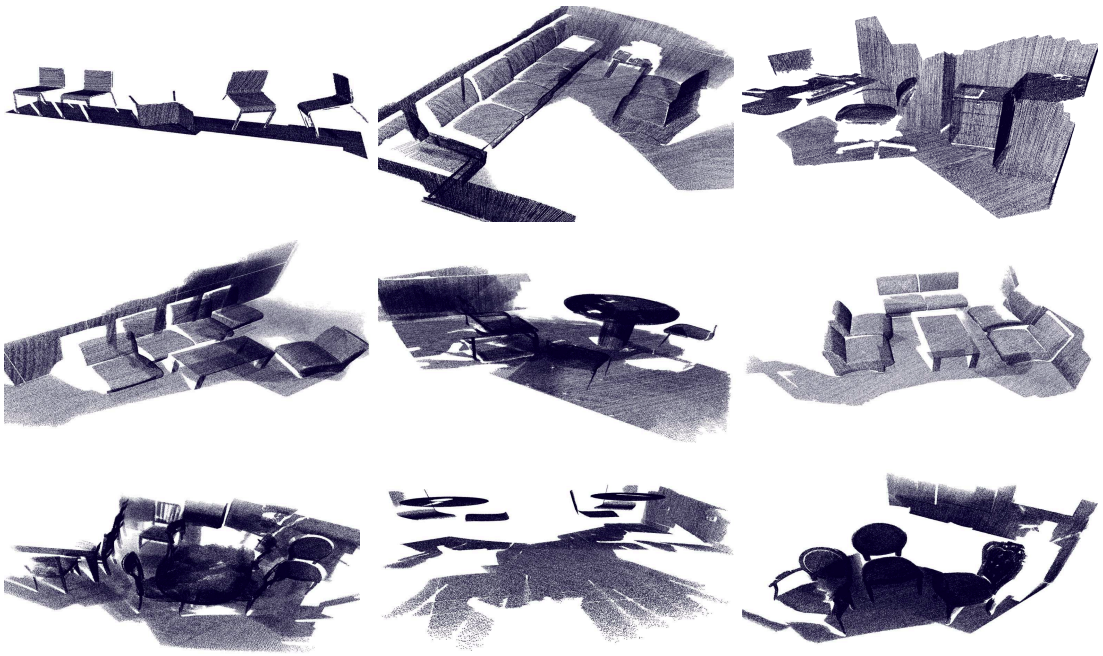


Figure 7.23: Point clouds of indoor scenes used by Nan *et al.* (2012)

7.5.1 Indoor Scenes from Nan *et al.* (2012)

In what follows, Algorithm 14 is applied on point clouds from Nan *et al.* (2012) and the resulting solutions are discussed. Figure 7.23 illustrates the variety in quality of these points clouds, with many of the objects partially occluded. Shelving and stairs structures are not detected in any of these examples. On the contrary boundaries are always extracted and are generally correct.

The first scene represents 5 chairs in different poses and as opposed to the method presented by Nan *et al.* (2012), CoFFrS can easily detect similar objects in a different pose to the one used for training. Figure 7.24 shows the models matched with the raw point cloud with additional details about the segmentation process illustrated in Figure 7.11.

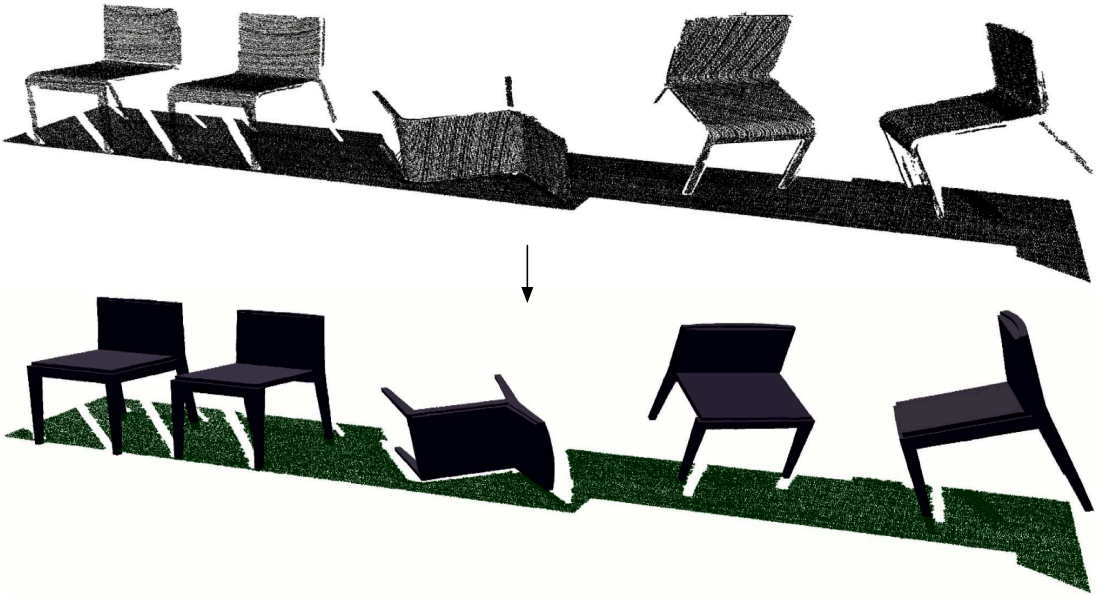


Figure 7.24: All chairs are correctly matched to the chair object graph. The floor segment is first extracted using the boundaries query graph.

The second scene consists of a number of couches, next to each other, a table and surrounding floor and walls. Figure 7.25 illustrates the matching order for this scene following the extraction of the boundary segments. Couches are all correctly matched expect for one, segment three, since the segmentation process groups together the back of two couches into one as illustrated in Figure 7.27 (bottom left). In this case, the grid matching process elongates the couch. Segment 8 shows an instance where the planar segment perfectly matches the second anchor segment of a table in both spatial context and point coverage and is thus correctly detected when matching the grids. Note however, that the size of the table is not correctly determined since there are only a few small segments from the top of the table.

The third scene consists of a desk, office chair and a number of cabinets. Figure 7.26 illustrates matches between the office chair, a table and three filing cabinets. In the case of the cabinets, a third cabinet (the largest) is erroneously matched to part of the wall which, due to their position, are not picked up by

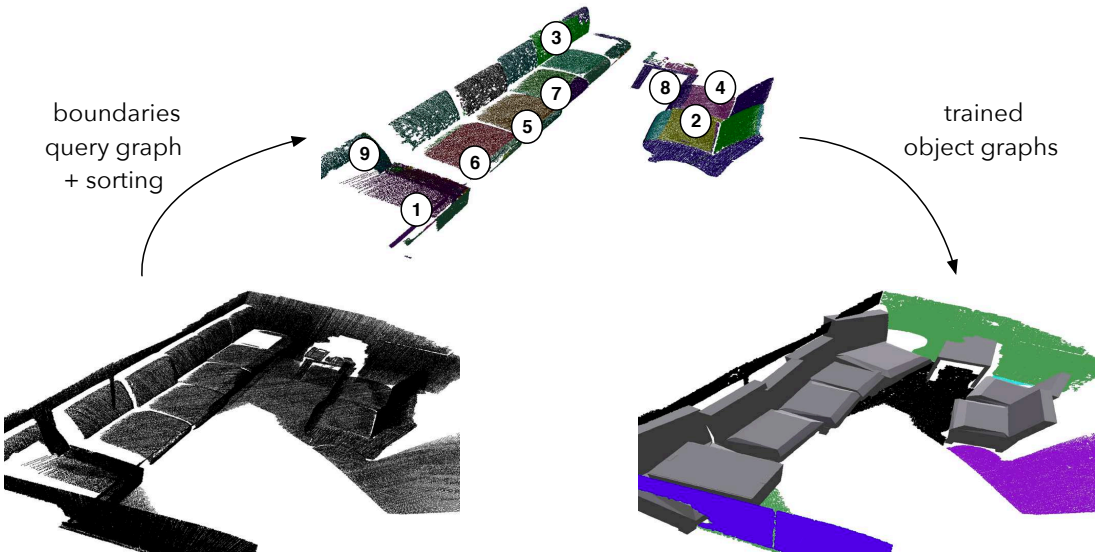


Figure 7.25: Planar segments after applying boundary query graph, numbers indicate segment order used for fitting seven couches, one box (segment 1) and one table (segment 8).

the boundary query graph. The office chair is correctly identified and obtains a higher score when grid matching because of the segments representing the arm rests as can be seen in Figure 7.27 (top left).

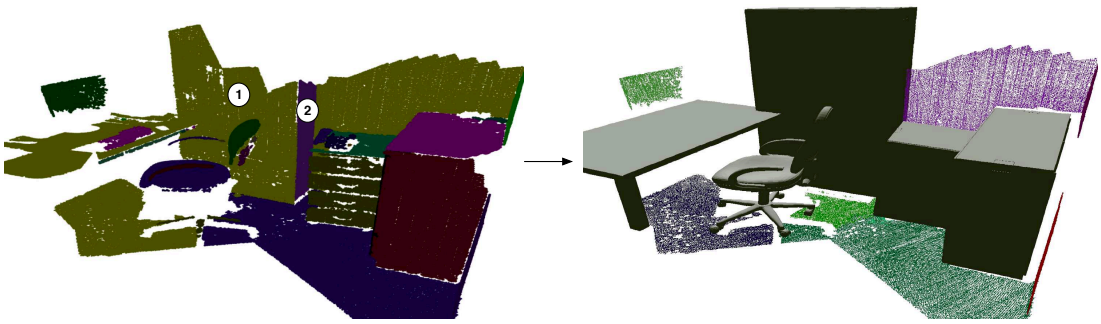


Figure 7.26: A segment forming part of the monitor is incorrectly labelled as boundary, and two segments (1 and 2) are incorrectly labelled as a cabinet. Chair, two cabinets and table are labelled correctly. Table is automatically oriented towards the floor.

The fourth scene consists of 5 couches, a table and room boundaries. The table is easily matched with the table object graph. Note that in this case, the visualised table model is automatically scaled to reflect the height of the table in the point cloud. Segments are matched in the order shown, even though in this example a different order would have resulted in the same solution. The computed voxel grid surrounding segment 2, is visualised from the side of the

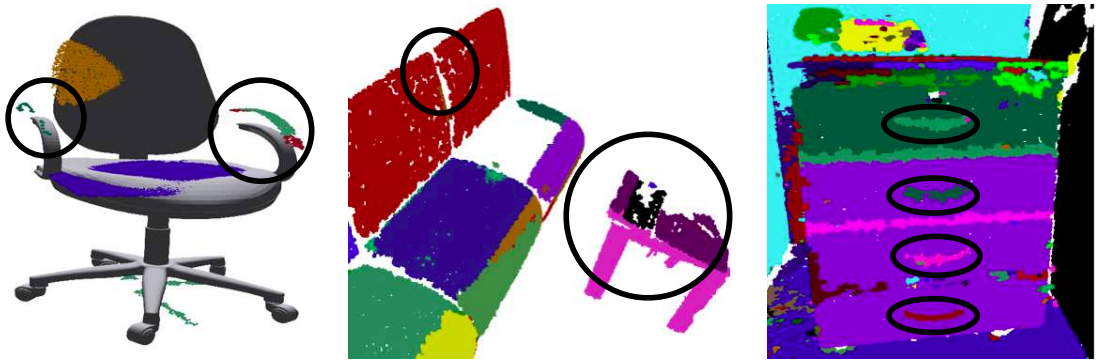


Figure 7.27: Left: Model mesh and point cloud segments super imposed showing arm rests in point cloud matching to arm rests in trained office chair improving grid match score. Middle: Over segmentation groups together the backs of two couches. Right: Segments representing the handles on the drawers of the cabinet could be used to orient model correctly.

couch to highlight the intersection between the voxel grids of the trained model and segments in the point cloud.

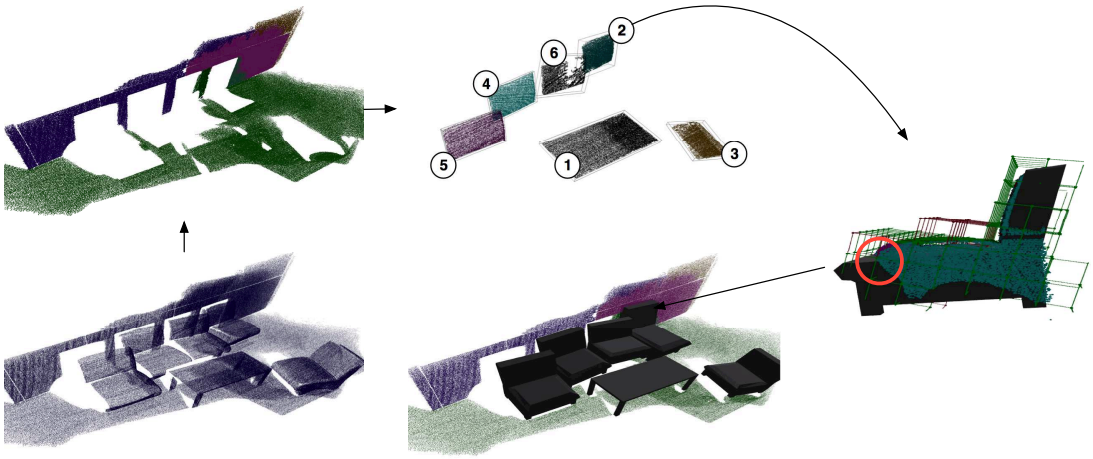


Figure 7.28: After extracting boundary segments, remaining segments are ordered and matched against object graphs. On the right hand side, the matching grid is seen overlaid on the model and segments.

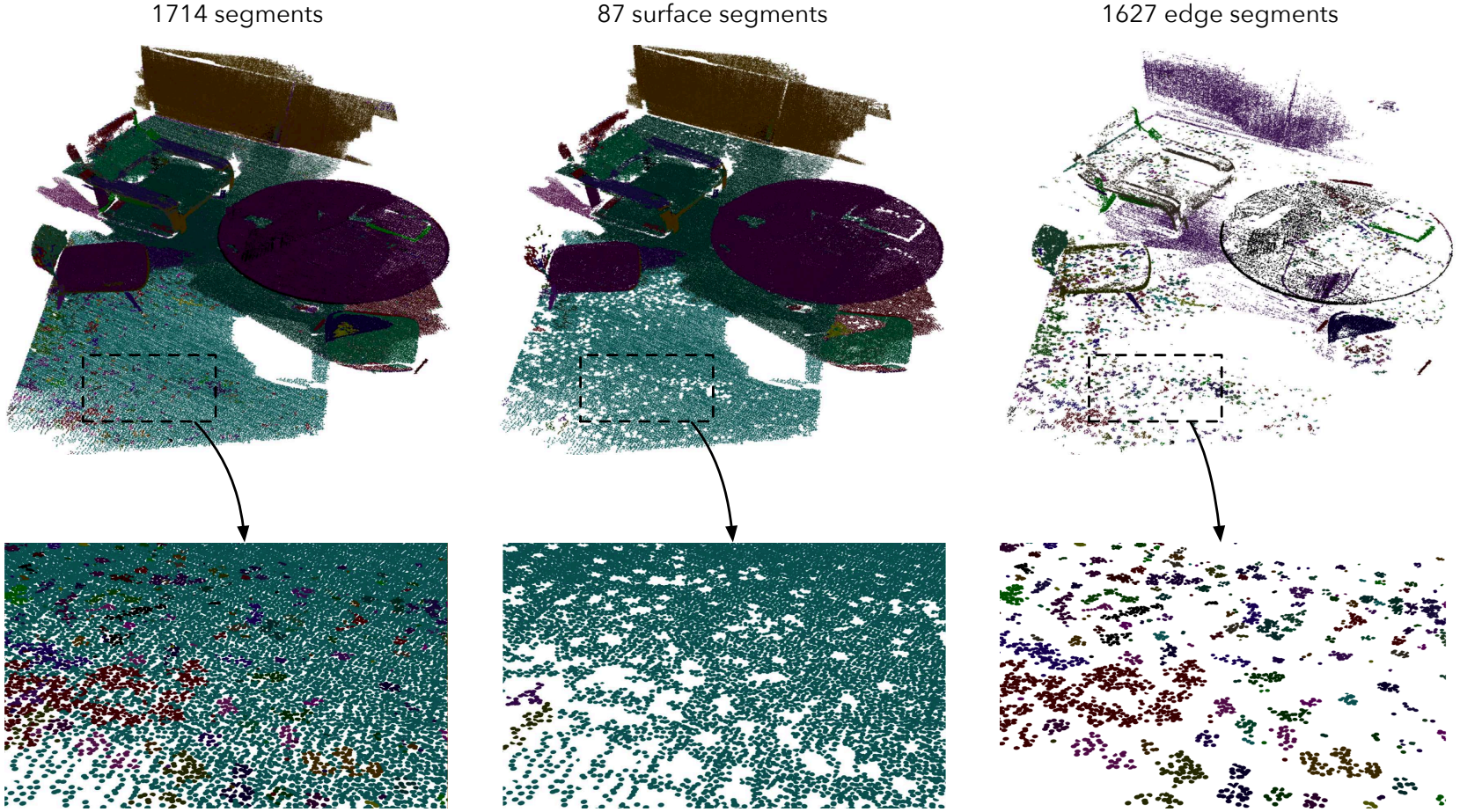


Figure 7.29: Scene 5 is partitioned into 1714 segments, 87 of which are surface segments and 1627 are edge segments. Close-ups on the floor show how many pockets of edge segments are found on the floor which probably consists of a carpet given the amount of noise. Part of the wall also contains considerable noise which is grouped together in one *edge·planar* segment.

The fifth scene consists of a table with rounded table-top and central stand, an armchair and two highly occluded chairs. The point cloud is partitioned into 1714 segments, 87 of which are surface segments and 1627 are edge segments. Figure 7.29 illustrates close-ups on the floor which show how many pockets of edge segments are located on the floor, which probably consists of a carpet given the amount of noise. Part of the wall also contains considerable noise which is grouped together in one *edge·planar* segment. Since both tables in G_{obj} have four legs at the corners of a rectangular table, an new object graph representing this particular table is synthesised and added to the set. Figure 7.30 illustrates the point cloud used to train the object graph representing a round table-top table together with the three anchor segments and the voxel grid computed around the first anchor segment.

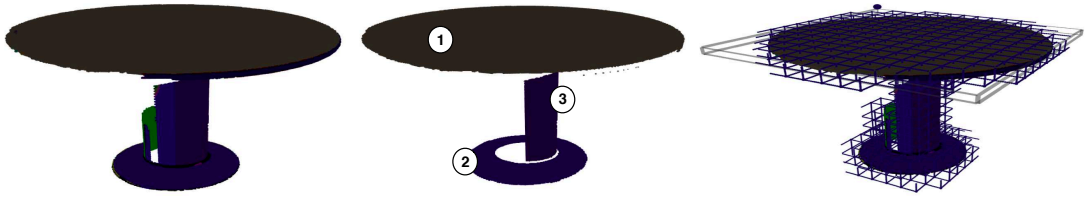


Figure 7.30: Point cloud of trained round table-top table; left hand side showing segments, middle showing the three automatically chosen anchor segments, and right hand side showing the grid computed around the first anchor segment.

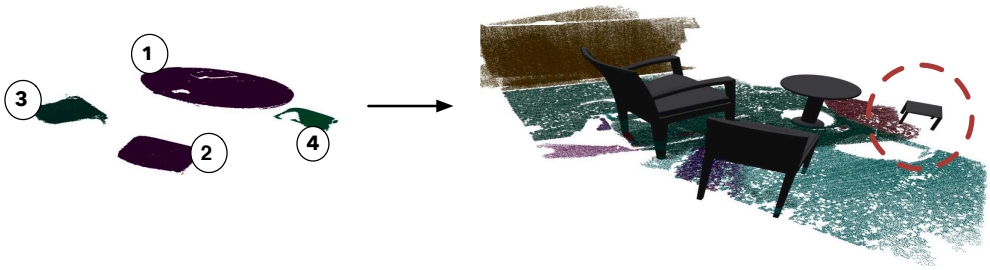


Figure 7.31: Round table object descriptor is included in the search and correctly matched to the scene. Note correct matching of armchair and incorrect matching of table to chair which lacks samples from the back.

Figure 7.31 shows the segment object mappings produced by the scene understanding process. The chair (segment 4) on the right hand side of the scene in incorrectly labelled as a small table, similar to a stool, since the point cloud does not include samples from the back of the chair. In this case the table obtained the highest grid matching score amongst the object graphs. The armchair object

graph scores slightly higher than the chair object graph for segment 3 and is matched to surrounding segments including the arms of the chair.

The sixth scene consists of a number of couches around a coffee table. This scene highlights a problem resulting from the heuristic used for the boundaries query graph, which in this case labels the backs of two couches as boundaries, given that there is nothing behind them (segments 6 and 7 in Figure 7.32). This leads to the incorrect labelling of segments surrounding them which make up the rest of the couches. Moreover, the seats of these two couches are not densely sampled and therefore cannot match anchors in the relevant object graphs. The best scoring match for these segments ended up being the cabinet. In the future work section, a solution to minimise these situations and in this specific case avoid it is described. The remaining couches and coffee table, rendered with matching voxel grids in Figure 7.32, are easily matched.

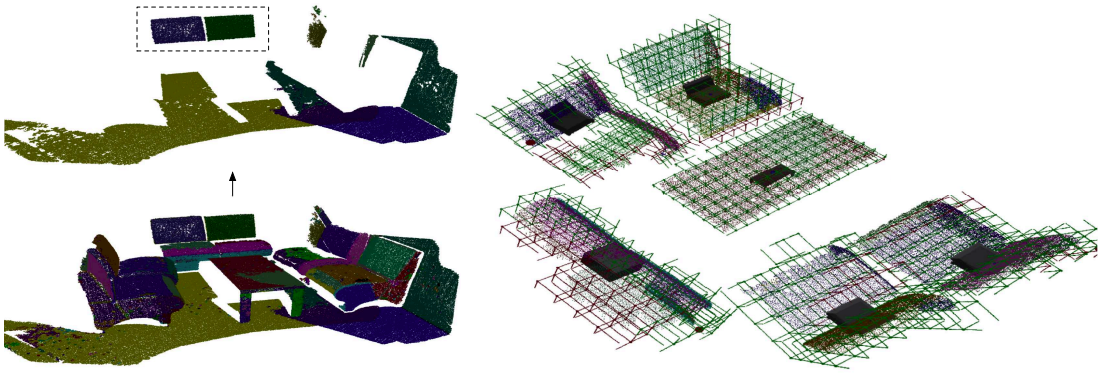


Figure 7.32: After extracting boundary segments, remaining segments are ordered and matched against object graphs. On the right hand side, the matching grid is seen overlaid on the model and segments. Note problem resulting from incorrectly including back seats to boundary.

Figure 7.33 illustrates a scene with two chairs and two tables. The low table includes two drawers whereas the larger table is not sufficiently sampled, probably because of its material. Since G_{obj} only contains generic objects, the low table is erroneously matched with the cabinet, with a higher score obtained during the matching process. Note that the table segment (2) which is matched against anchor segments in G_{obj} consists of the side, including the two drawers, on the table. After the cabinet, the second best option is a table tilted on one side. Both chairs are correctly matched to the generic chair.

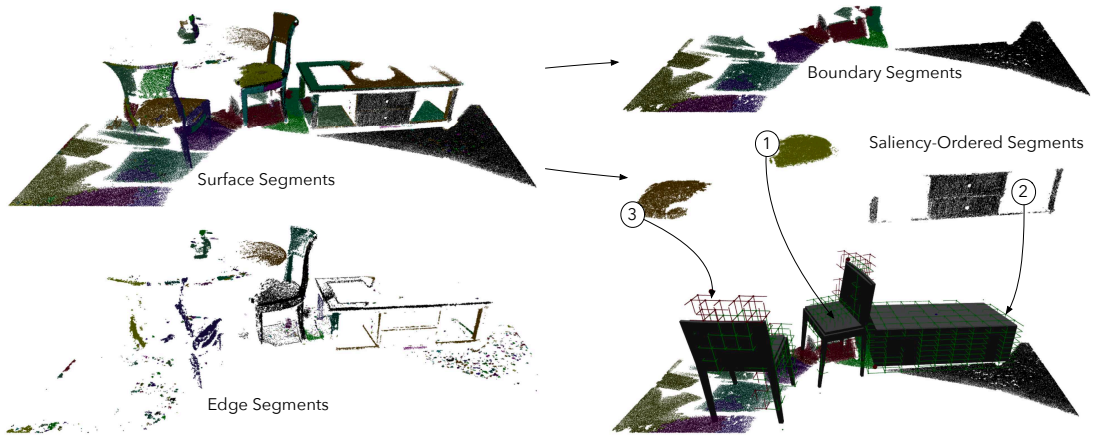


Figure 7.33: Scene consisting of two chairs and a low table with drawers in the central part. Whereas for both chairs sufficient segments are present to establish a correct match, the table is fitted to a cabinet which is appropriately scaled to fit the size of the table.

7.5.2 Additional Indoor Scenes

A number of additional scenes of indoor environments, scanned using both the Asus Xtion and Structure Sensor scanners, are used to evaluate CoFFrS on more complex scenarios which include stairs and shelving. Moreover, the newly scanned point clouds include couches which are longer along the horizontal axis to the one which is trained, include plants in pots and chairs not in an upright pose. As opposed to Nan *et al.* (2012) the point density of the data sets is much lower.

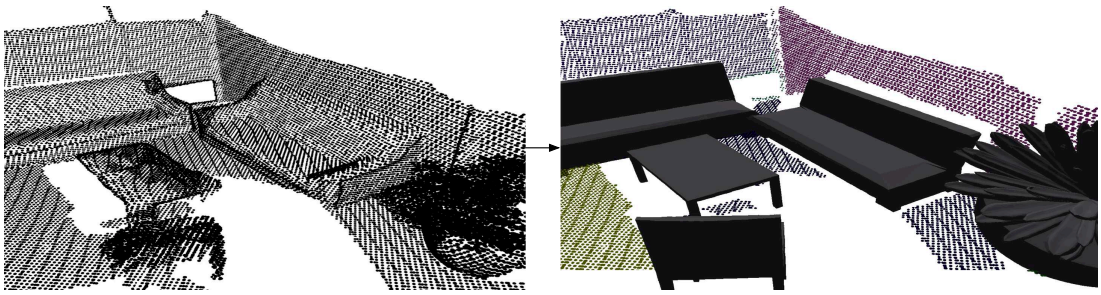


Figure 7.34: A low density point cloud scanned using the Asus Xtion sensor. The couch object graph is scaled to fit the two couches. Table and chair are easily fitted as enough samples are available for both, as well as the pot which has one of its sides closely sampled.

The first scene consists of two couches, a coffee table, a chair and a plant pot in a room. Figure 7.34 illustrates the low density raw point cloud and mappings

produced. The one-seat coach in G_{obj} is scaled during the grid fitting process to fit the segments making up the two two-seat couches in the scene. The plant pot model fits well with the segments in the structure graph representing the cylindrical sides of the pot. The segments resulting from the execution of the boundaries query graph are shown with the fitted models in Figure 7.34.

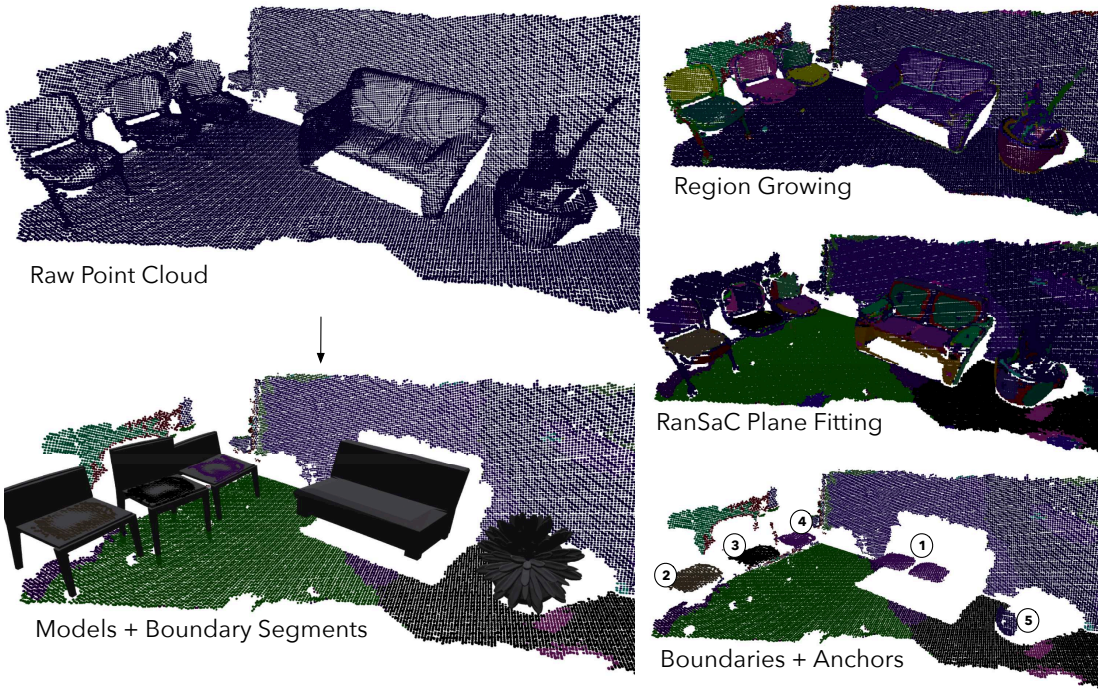


Figure 7.35: The region growing process generates a segment representing the coach, which RanSaC plane fitting breaks into planar segments adequate for CoFFrS to match with anchors in G_{obj}.

Figure 7.35 illustrates a scene consisting of a couch, three chairs and a plant. The region growing process of PaRSe produces the segments shown in the top-right image. One of these surface segments represents the entire couch, which is then partitioned into planar segments during the RanSaC plane fitting process. One of these *surface·planar* segments (1) is the seat of the couch which is used to match the couch object graph.

The third example consists of an office scene including shelving cabinets and chairs in a variety of poses and locations. Figure 7.36 illustrates the various steps carried out by CoFFrS with the top-right image showing the raw point cloud, and arrows pointing to two images showing edge (under) and surface (right) segments. Query graphs to determine boundaries and identify shelving units correctly determine the segments making up the boundaries and the shelving

cabinet. The middle row right image illustrates five segments along a specific direction (shelves surface normal) which are established by the shelving query graph. The rest of the segments are ordered according to their saliency values staring from the table top (1), and finishing with the seat of the toppled chair (6). Figure 7.37 shown a photograph of the shelving cabinet which is cluttered with objects and files, but which still has parts of the shelves visible. Points acquired from these parts are sufficient for PaRSe to generate multiple *surface·planar* segments, and subsequently for CoFFrS to consider this set of segments as a shelving unit.

A fourth example, consisting of a very noisy point cloud with tilted chairs is shown in Figure 7.38. It illustrates the steps carried out by CoFFrS, in determining the objects in the room and also highlights some ambiguity problems which can result in the process. Whereas two of the chairs are correctly fitted, the one at the corner is mapped to the couch object graph. This happens because the two segments representing the legs of the chair (bottom right corner) are not well aligned with the legs of the chair used in G_{obj} and therefore do not contribute to improve the chair fitting score. A fourth chair (segment 5) is also fitted to a table model since no scale information is used. The table is correctly fitted, however the orientation is not exactly the same as in reality, since only parts of the table-top are sampled.

Figure 7.39 illustrates a scene consisting of multiple shelving units in addition to some chairs and a table. As shown in the figure (top right), four separate sets of segments are returned by the query. The minimum number of shelves required to describe a unit is set to four. All segments within the extended OBBs of these segment groups are included in each shelving unit group resulting in one large unit due to their adjacent positions. After extracting the segments related to shelving, boundary segments are removed and the rest of the planar segments are ordered with respect to saliency.

Figure 7.40 illustrates a scene consisting of a flight of stairs. 127 planar segments are extracted from the 31K point cloud, 11 of which represent steps which are identified by the stairs query graph. The middle row illustrates a segment extracted by the region growing process, which is then further partitioned into 12 planar segments. Three of these in addition to another 4 (bottom row) form up a pattern which is identified by the stairs query graph. Another set of steps is extracted separately, since the query only checks for adjacent planar segments along one direction. Moreover, in this case the two sets are separated by a region

on the stairs which is not sampled. Whereas the chair standing on the stairs is correctly identified, the rails and additional segments along the stairs are excluded from the matching process with object graphs. This is done by the stairs query graph with generates a bounding volume along the boundaries of the steps which excludes any segments which fall within. This heuristic is used to identify walls and rails which are typically found close to stairs.

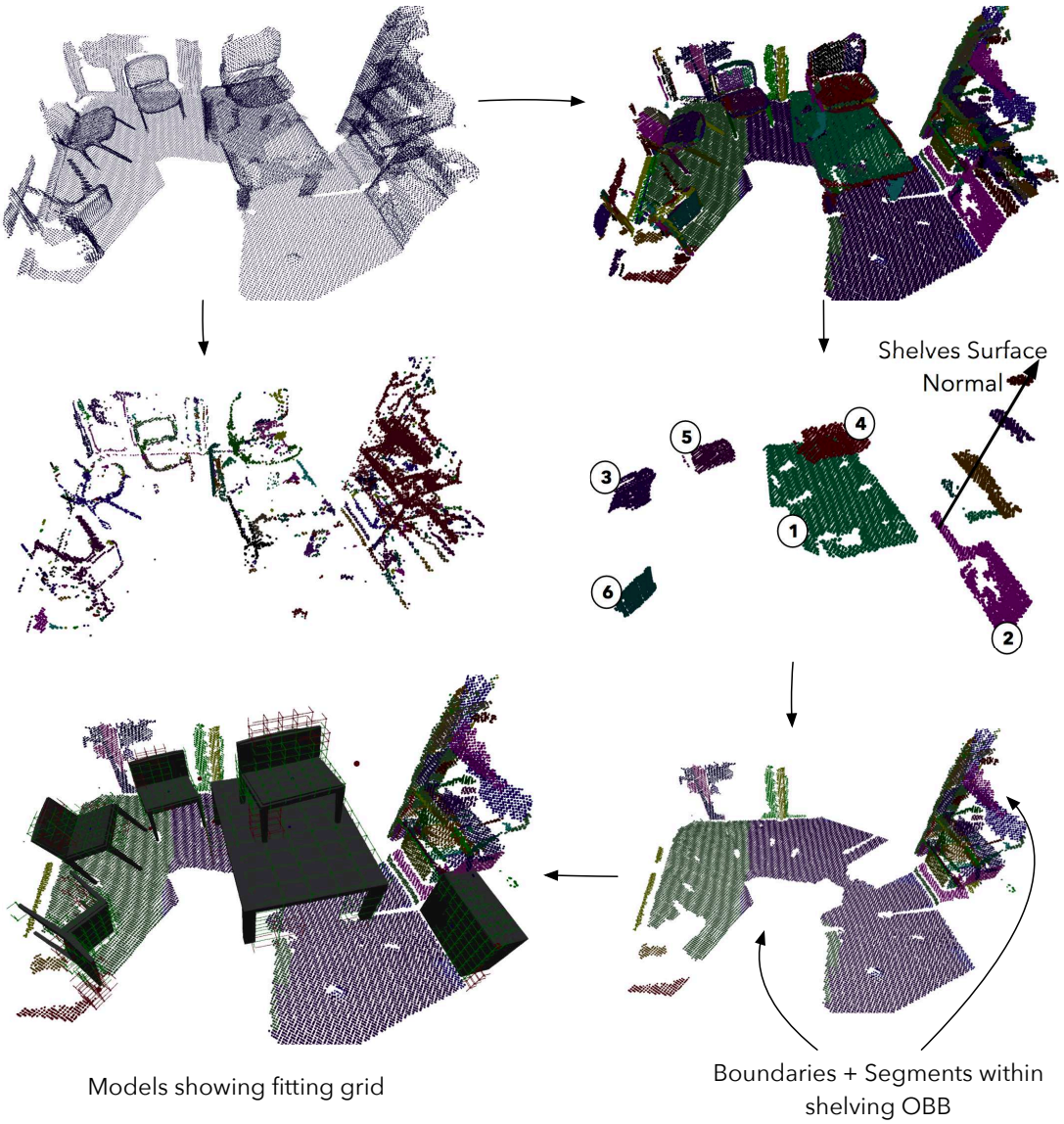


Figure 7.36: An office with chairs in a variety of poses and distances from the floor, a table, a cabinet and a cluttered shelving unit. Boundary and shelving query graphs first extract boundary and shelving unit segments. The rest of the segments are sorted (as indicated by numbers) by saliency and models fitted using anchor segments and grid matching.

Figure 7.37: Shelving cabinet cluttered with files, boxes and books. A shelving query graph first identified parallel segments with overlapping OBBs, then adds segments falling within the OBB of these segments.

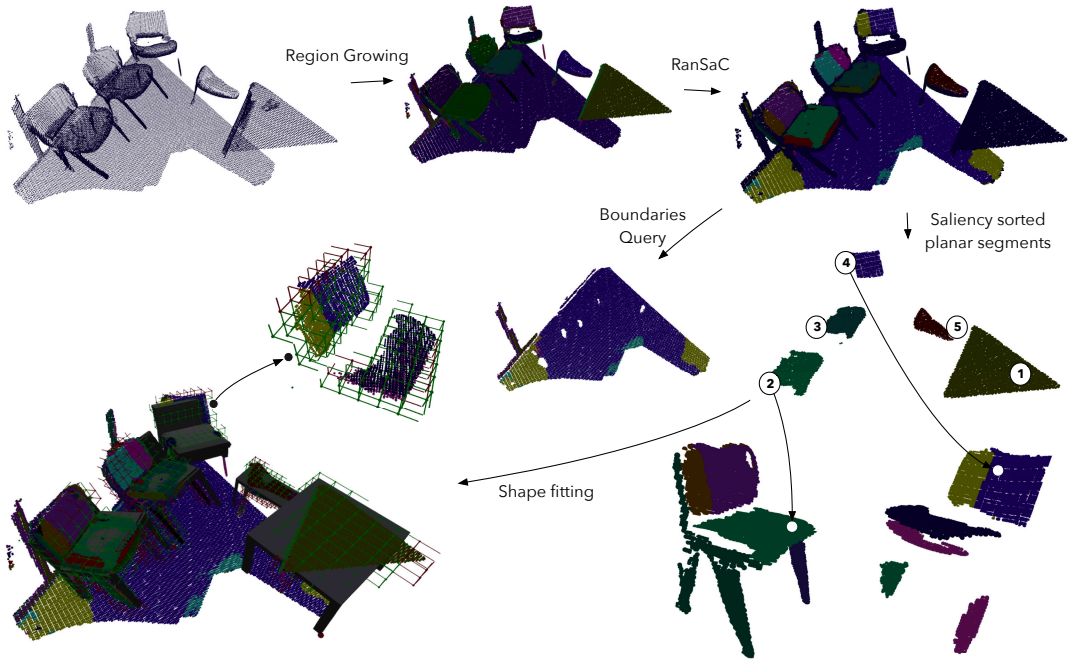


Figure 7.38: Chairs and tables in an office; two chairs are correctly recognised whereas a third tilted chair in the corner is recognised as a couch and a fourth as a table. In the latter case there is only one leg and no back info. In the former there is insufficient leg information and the back and seat of the chair are at a wider angle, which is closer to the trained couch.

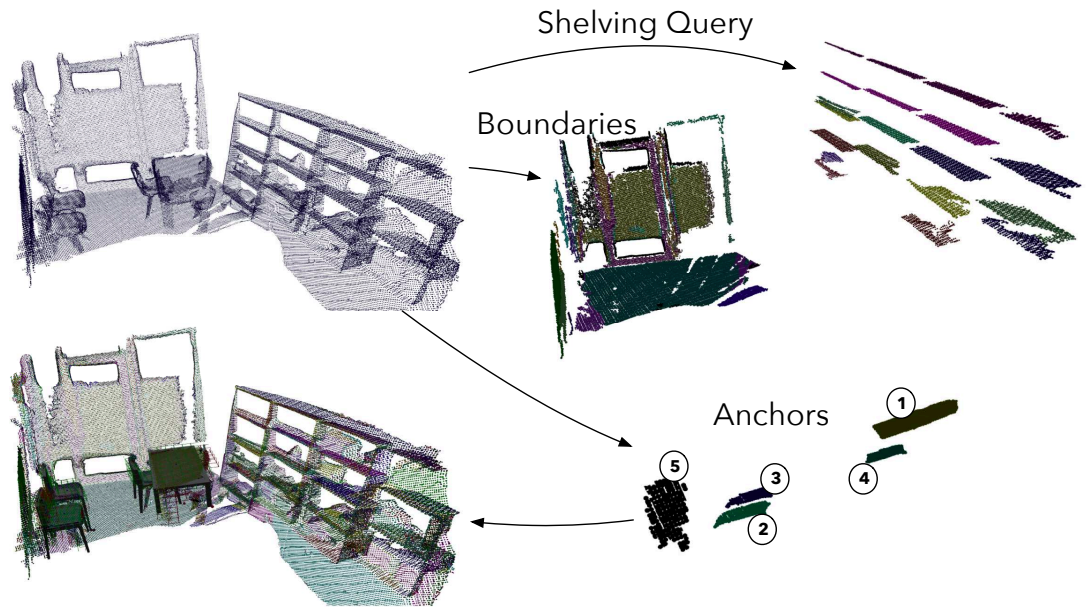


Figure 7.39: The shelving query graph returns four sets of segments each representing a shelving cabinet. In addition, a table and three chairs are correctly identified together with boundary segments.

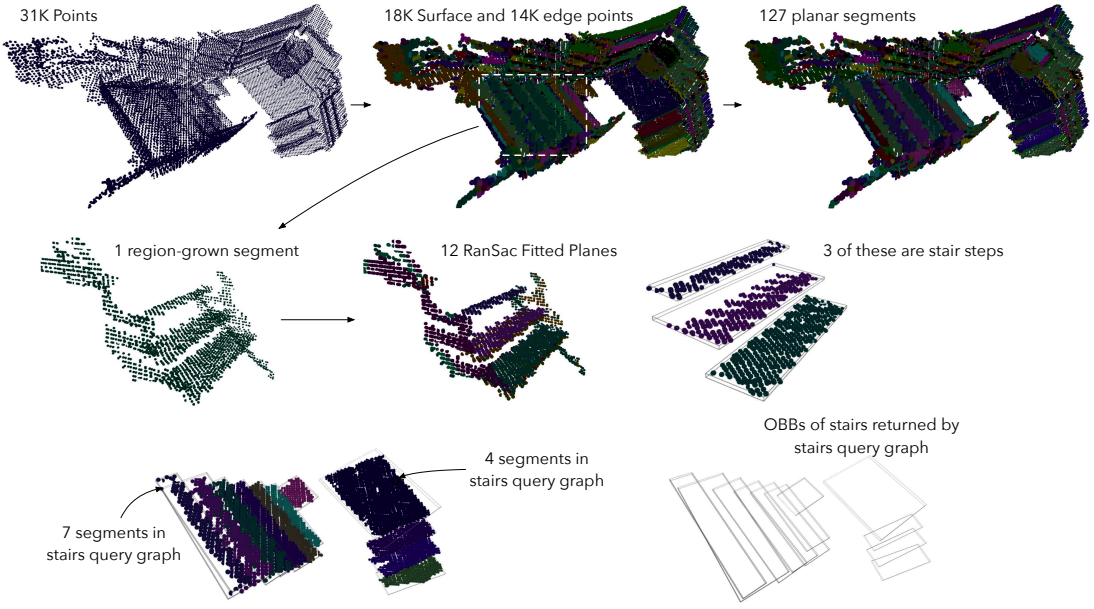


Figure 7.40: CoFFrS is applied on a low quality point cloud represetning a scene with a flight of stairs in order to evaluate the stairs query graph. Two sets of steps are returned by the query, with directions orthogonal to each other. Note how (middle row) given the quality of the point cloud, region growing produces segments which included multiple steps, which RanSaC plane fitting further segments to produce the required planar segment primitives. The third row shows the two sets of segments and the OBB of each step.

7.6 Discussion

CoFFrS proposes a scene understanding approach which does not depend on prior scene-specific information. Clearly, in order to populate the set G_{obj} of object graphs, the system needs to know, generically, which objects can be found in the scene and then use adequate 3D models to synthesize generic graph-based descriptors building on the set partitions output by PaRSe. In the case studies used, typical office furniture including chairs, tables and couches is used (Figure 7.19). Contrary to other indoor scene understanding approaches (Nan *et al.*, 2012; Kim *et al.*, 2012; Mattausch *et al.*, 2014), CoFFrS does not require scene parameters such as scale, upward direction and floor location. This is possible via a search mechanism which matches anchor segments in previously trained object graphs to saliency-ordered planar segments in the structure graph of a scene. Whereas in many cases CoFFrS demonstrates that it is possible to interpret a scene without prior scene parameters, severe occlusion and noise may pose a limitation. In general, at least one matching anchor segment is required to be mostly visible, even if with holes, in addition to some supporting segments around it to correctly establish a mapping between segments in the scene structure graph and anchors in G_{obj}. For instance, Figure 7.41 shows a scene where two chairs have only their back visible. Whereas PaRSe does a good job at clustering these points as separate segments, currently, these segments are not properly matched since there are no other connected segments which match to the chair object graph. The $PatternMatch$ function (Algorithm 14, line 20), which selects the subset of G_{obj} to be checked, can be modified to include information about the segment pose of backs of chairs, then just one planar segment in the scene could be enough for correct identification.

CoFFrS includes the possibility of searching for specific patterns in a scene prior to object matching, akin to using an unsupervised approach. These patterns are encoded as query graphs, introduced in PaRSe and further extended in CoFFrS, which are applied to extract boundaries, shelves and stairs. In all cases, instances of these structures will vary across scenes and are therefore difficult to encode in a scene descriptor such as the one used by Nan *et al.* (2012). Once more, the lack of prior scene parameters, limits the robustness of these methods. For instance, if four chairs are perfectly aligned one after the other, the backs of the chairs would form the pattern required to establish a shelving unit. Similarly, chairs can be arranged in such a manner to resemble a flight of

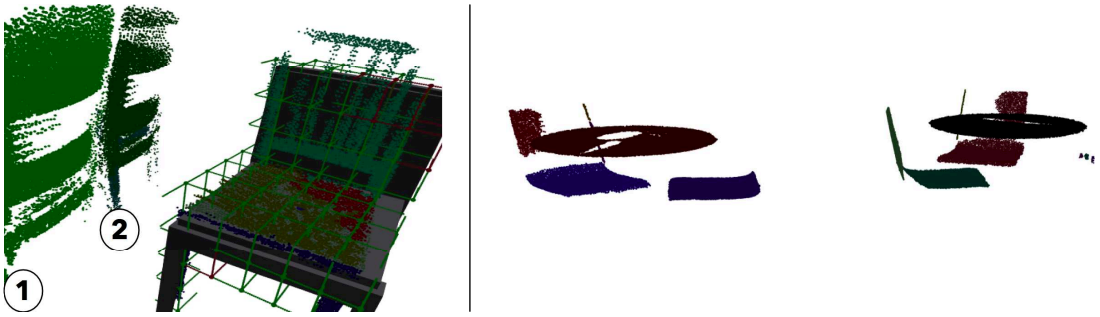


Figure 7.41: Segments 1 and 2 lack the support of other chair components which are required to identify both chairs, given that no prior pose assumptions about the objects in G_{obj} are made. Could easily be interpreted as one or two tables. A third chair in the scene is shown with mesh and grid superimposed on the segment points. Similarly, the segments in the scene on the right do not have sufficient supporting structure to correctly map segments to objects.

stairs. If prior information about the possible world scenarios is utilised during the application of these queries, then these exceptions can quickly be discounted. Nevertheless, since heuristics are used to describe these structures, certain ambiguities, although minimal, can always occur. If scene parameters are available or can be established during the acquisition or searching process, CoFFrS can take advantage of these constraints to prune the search space and improve the interpretation of the scene. For instance, a future implementation may look into using location-based RFID tags to guide the selection of object graphs returned by the *PatternMatch* function. For this to be possible, the acquisition phase needs to gather this information and include it with the input to CoFFrS. Additionally, if at any point of the scene understanding process, a specific object is identified (e.g. a table), the segments close to that object could be searched for within that context by using techniques similar to those proposed by Fisher *et al.* (2011). Moreover, previously established relationships between objects can be used in cases of extensive occlusion and noise.

Whereas the boundaries query graph used in the examples is generally sufficient to determine the walls and floor, it makes the assumption that these are represented by segments lying near the boundaries of the scene. Evidently, this is not always the case, for instance with multiple-room environments and scenes with no walls such the one shown in Figure 7.26. Since, there can be no guaranteed as to how much of the scene including the walls is sampled or occluded, further research in the use of more advanced heuristics is necessary in order to develop query graphs which more robustly identify these important structures.

Similarly, both the shelving and stairs query graphs use a user-set parameter which determines the minimum number of repetitions (shelves or steps) required to determine the presence of the structure. The value of four is used in the examples presented, however with shelving units or flights of stairs having less shelves and steps, the query graph would not have been able to detect the structures.

The current implementation of structure graph searches is very inefficient in that each query is iteratively applied (Algorithm14, lines 7-16) to each segment in the scene structure graph. Further work is required in order to provide a mechanism which merges the query graphs into one, which is then used to map segments to structures in a single iteration. A histogram based on segment properties such as size and orientation, similar to Mattausch *et al.* (2014), may also be helpful in accelerating the application of these query graphs by avoiding full traversals of the scene structure graph.

Voxel grids in object graph descriptors are used to coarsely describe the shape of an object around three automatically established anchor segments. Whereas this representation has achieved good results, object graphs can be further enhanced to improve both descriptive power and matching performance. In the examples used, G_{obj} consists of a small number of objects given that the generic descriptor is designed to fit similar objects (e.g. different chairs in Figure 7.41). However, if the number of object graphs increases substantially matching performance would become a primary concern. Whereas, run-time context may play an important role, object graphs may also contribute towards improving matching performance by embellishing them with the capability of describing a class of objects rather than a single object. Splitting the current uniform voxel grid in two, each representing the half-plane above and below the anchor segment, would probably be required in order to increase the compatibility of an object graph with a larger number of objects.

Not all objects can be reasonably encoded using one type of object descriptor. In particular, object graphs used by CoFFrS are not appropriate for complex objects which do not exhibit dominant planar patches. Similarly, other scene understanding techniques have not considered the inclusion of multiple descriptors during the searching process. The segmentation results returned by PaRSe, position CoFFrS as a good basis for the inclusion and testing of this possibility.

7.7 Summary

This chapter has presented CoFFrS, which extends PaRSe to provide a framework for context-free scene understanding. Context is not used in the training of object and structure descriptors, despite its importance in pruning solution search spaces. This novel approach has been demonstrated on point clouds, some taken from literature, some newly acquired. It is acknowledged that the lack of scene-specific parameters may limit the recognition efficiency of CoFFrS in the case of low quality point clouds. Nevertheless, CoFFrS pushes forward the boundaries of current solutions in its successful application to previously unsolved scenes. Specifically, given enough samples, objects in different poses are correctly matched and structures such as shelving, boundaries and stairs, which vary between scenes, are naturally included in the interpretation pipeline. This chapter has utilised PaRSe in establishing a novel scene understanding framework on which further research can be based.

CHAPTER 8

Conclusions and Future Work

The processing of point cloud data has become an increasingly important field of computer graphics. In part, this is due to the development of novel acquisition methods, which in recent years improved both in sampling rates and precision. Nevertheless, segmentation methods which help to manage the ever increasing complexity of acquired data sets has advanced at a slower pace, with the focus being mostly on producing ad hoc solutions using context-specific information. This is the case in many areas where a segmentation process is required, including indoor scene understanding. This work presents a different approach to the status quo: a general-purpose segmentation method and a context-free indoor scene understanding framework are proposed, which take as input raw point clouds and make no prior assumptions on the input data sets. This approach widens the applicability of point cloud data and benefits the fields and applications wherein they are employed.

8.1 Contributions

The overall contribution of this thesis is a step forward in widening the applicability of point cloud data via the design of context-free point cloud processing algorithms. PaRSe contributes a segmentation pipeline to facilitate the processing of raw point clouds. As was demonstrated, the general purpose segmentation process proposed in this work can be successfully applied to a wide variety of tasks to produce segments which are meaningful within their field of application. This is especially true in the field of scene understanding from point clouds, where PaRSe has contributed towards the creation of an alternative approach to indoor scene understanding, which had predominantly depended on context and

prior scene information. In this regard, CoFFrS contributes a context-free scene understanding framework for point clouds representing indoor scenes. PaRSe is also extended to cope with massive point clouds which do not entirely fit in main memory via a novel out-of-core method which accelerates k-NN computations utilised by a variety of point cloud post-processing algorithms. Chapters 3 and 4 contribute literature reviews on segmentation and indoor scene understanding techniques from point clouds. This work is intended to open the door to further investigation into alternative general purpose segmentation and context-free scene understanding algorithms, as well as improvements on those presented herein.

8.1.1 Plane-fitting and region-growing Segmentation (PaRSe)

In Chapter 5, a general purpose point cloud segmentation method has been introduced which outputs a structure graph with nodes representing segment primitives. The transition relation of the output structure graph describes adjacency between these segments and is incrementally built during the segmentation process. In order to increase its applicability, rather than choosing between a region-growing or a shape fitting process, PaRSe, proposes a segmentation pipeline combining point labelling, region-growing and shape fitting. When applied to raw point clouds, especially if considerable sample noise is present, region-growing processes tend to suffer from over-segmentation problems since neither local curvature nor surface normals can be computed reliably. PaRSe uses an initial binary labelling of points, based on local surface smoothness, to drive the region-growing process and is, therefore, less susceptible to noise. Plane fitting using the RanSaC paradigm is then applied on the resulting regions. In order to provide for a context-free segmentation process, only plane fitting is carried out. To bolster automated reasoning about the input point cloud, a query mechanism is proposed which searches for sub-graphs by matching patterns defined as a query graph. Results were demonstrated on a variety of point clouds showing how the resulting segments correspond to meaningful parts in the input. Query graphs were used to extract columns, temple apses and trees in different case studies. A LiDaR acquired point cloud of the Maltese archipelago is partitioned into segments representing streets, house roofs and agricultural fields. In all cases, no prior context specific information about the input point clouds is used.

List of Contributions

- The design of a general purpose segmentation pipeline applicable to a wide variety of tasks in different fields.
- A proof of concept implementation of a point cloud query mechanism which can be tailored to automate different post-processing tasks.
- PaRSe was demonstrated on a variety of raw point clouds, showing how the resulting segment primitives correspond to meaningful parts of the input.

8.1.2 Fast Scalable Out-of-Core k-NN Searches

In Chapter 6, a novel out-of-core k-NN search method is presented. Many point cloud post-processing algorithms rely on the computation of point neighbourhoods which can take a considerable amount of time depending on the size of the point clouds and value of k. In order to avoid overheads related to disk I/O, out-of-core techniques are required when the point cloud does not entirely fit in device memory. To counter these situations, the implementation of PaRSe incorporates a novel external memory algorithm which uses a hybrid of spatial subdivision techniques for out-of-core fast k-NN searches. The hybrid approach exploits the spatial locality of point clusters in the data and loads them in system memory on demand by taking advantage of paged virtual memory in modern operating systems. Results demonstrate that processor utilisation is maximised while keeping I/O overheads to a minimum. The method is evaluated on point cloud sizes ranging from 50 thousand to 333 million points on machines with 1Gb, 2Gb, 4Gb and 8Gb of system memory. On a 1Gb machine, 100 thousand neighbourhoods/s are computed on a point cloud consisting of 166 million points.

List of Contributions

- An fast scalable out-of-core k-NN method suitable for devices with limited amounts of system memory.
- A comparison with standard CPU based in-memory k-NN methods.

8.1.3 Context-Free Framework for Scene Understanding (CoFFrS)

CoFFrS, introduced in Chapter 7, is a novel scene understanding framework suited for indoor scenes which does not require prior scene-specific information during the identification process. CoFFrS uses structure and query graphs introduced in PaRSe to search for patterns in an input point cloud and adds object graphs to encode segment compositions of specific objects. These object descriptors are independent of object pose and do not assume a fixed upright position as is the case with many indoor scene understanding methods. Following a training process which encodes object graphs for a small set of generic indoor office objects, a search process first applies query graphs to identify varying structures such as room boundaries, shelving and stairs. This is then followed by a search for scene segments matching anchor segments in the object graphs. Connectivity patterns between segments are used to prune the matching process, with the planar segments produced by PaRSe critical to the success of CoFFrS. Results show that CoFFrS is a viable alternative to indoor scene understanding methods which are based on prior scene information and provides a firm foundation for context-free methods in this field. CoFFrS has been evaluated on scenes used in previous literature, and on new ones to demonstrate the benefits of combining a process which searches for specific segment patterns and a supervised process which searches for previously trained objects in low quality point clouds. CoFFrS addresses indoor scene understanding scenarios which were previously unsolved.

List of Contributions

- A novel pose invariant object descriptor, the object graph, based on the segmentation results of PaRSe.
- A context-free scene understanding framework based on a supervised process in order to correctly interpret scenes consisting of varying structures (e.g. shelves) and previously trained objects (e.g. chairs).
- An implementation of CoFFrS which is used to evaluate the validity of the approach on a variety of point clouds.

8.2 Synopsis

An ideal segmentation process partitions a raw point cloud into subsets of points each representing meaningful entities for a particular task. In many instances, for example Tarsha-Kurdi *et al.* (2007), Moosmann *et al.* (2009) and Golovinskiy & Funkhouser (2009) prior knowledge of a scene is utilised in order for the segmentation process to obtain good results. These requirements considerably decrease the adaptability of these algorithms to address problems in different settings other than the ones for which they are designed. PaRSe takes a different approach by only assuming the availability of position information. More general purpose region-growing methods make use of local point properties including surface normal and curvature, to expand regions from seeds which conform with user-set boundary criteria. The set partitions produced with these methods, however, are not amenable to further processing in cases where more complex objects are built from the composition of these segments. Vosselman *et al.* (2004) and Pu *et al.* (2006) address this problem by growing segments which adhere to plane primitives extracted from seed points, resulting in point cloud set partitions whose elements represent points falling on (or near) each plane. The application of this method on generic point clouds, which might not include large smooth planar surfaces is not ideal. PaRSe addresses this problem by including a plane fitting process over regions with the same point type. Point type, $surface$ or $edge$, is determined via a local surface smoothness metric computed using the k-NN of each point, and is primarily used to distinguish between points falling on the edges of an object, from the rest. This allows for the creation of a hierarchy of segments which can either represent complex surfaces or else regions which are decomposed into a set of planar segments. In the former case these segments typically represent important entities within the input, and in the latter, planar segments can be used to reason about or search for specific patterns in a scene.

Methods targeting specific scenarios have been proposed which assume the data can fit a particular shape. For example, Chaperon *et al.* (2001) extracts cylinder primitives from point clouds acquired from industrial scenes, and Abuzaina *et al.* (2013) and Camurri *et al.* (2014) fit sphere primitives to point clouds representing spherical objects like balls and apples using the 3D Hough transform. In these cases, the applicability of these methods to generic point clouds is limited. Schnabel *et al.* (2007) uses RanSaC shape fitting to produce a set partition of the input point cloud whose elements map to a variety of shape primitives. Seg-

mentation proceeds by extracting locally close support points and determining which shape best fits nearby points. Whereas the method does a very good job of approximating the point cloud by a collection of shapes, it is only evaluated on high-quality dense point clouds. On the contrary PaRSe is applied on a wide variety of point clouds from different fields and in many cases produces set partitions whose planar segments closely match the input point cloud. As opposed to Schnabel *et al.* (2007) rather than applying RanSaC shape fitting over points within a user-set distance, PaRSe applies plane fitting within the regions produced by the region-growing process. This creates a two-level hierarchy of segments which can be tailored to address a wide variety of tasks. PaRSe is an intuitive and robust general purpose segmentation method which can easily be integrated within existing point cloud post-processing pipelines. In particular, PaRSe has been extended to offer out-of-core processing capabilities, thus extending its applicability to point clouds which do not entirely fit in main memory.

Table 8.1 provides a feature comparison of CoFFrS to related work. The methods presented by Anguelov *et al.* (2005), Rušu *et al.* (2008), Zhao *et al.* (2010), Koppula *et al.* (2011), Adan & Huber (2011), Shao *et al.* (2012), Anand *et al.* (2012) and Song & Xiao (2014) all require range images as their input. In particular Zhao *et al.* (2010); Shao *et al.* (2012) use camera parameters to determine distances to specific objects and Koppula *et al.* (2011); Anand *et al.* (2012) require colour information from these images to carry out segmentation. In contrast, CoFFrS does away with these constraints and can be applied on raw point clouds acquired using SLAM methods (Bailey & Durrant-Whyte, 2006) and which only contain position information. In a similar fashion to Nan *et al.* (2012), a process searches for meaningful objects in a scene by accumulating surface segments (patches) with a high classification likelihood referred to as anchors. Identification of scene objects relies on the correct extraction and association of these segments with anchor segments in trained object descriptors. This allows for a pose-invariant matching of objects, since these are now aligned with the basis of the anchors rather than with a global scene upright position as required by Nan *et al.* (2012) and Kim *et al.* (2012). All indoor scene understanding techniques are context-sensitive with the exception of Shao *et al.* (2012) which however requires user input in order to properly generate range image segments representing individual objects. The unsupervised method proposed by Mattausch *et al.* (2014) extracts segment patterns in a raw point cloud but still makes assumptions on distances between the ground and table surfaces. So

even though unsupervised, their method still depends on important scene specific information in order to extract interesting patterns. Moreover, their method is only evaluated on high-quality dense point clouds which contribute towards the correct extraction of planar segments and do not associate a label to the clusters of planar segments extracted. CoFFrS integrates a similar approach to the interpretation process of a scene by augmenting the supervised approach using object graphs with searches for specific patterns in a scene using PaRSe query graphs in order to determine the presence of structures like shelving which vary between different scene but still adhere to a specific pattern. As opposed to Mattausch *et al.* (2014), PaRSe has been evaluated on low-quality point clouds acquired using commodity hardware and demonstrably manages to identify patterns in scenes such as shelving units and stairs. In particular, CoFFrS is a first attempt at reasoning about multi-level indoor scenes through the application of query graphs introduced in PaRSe.

Prior contextual information about a scene can be taken advantage of when providing a solution for a specific task. This however also restricts the adaptability of a solution to different tasks. In removing context from the design of segmentation and scene understanding methods from raw point clouds, PaRSe and CoFFrS propose an approach which widens their applicability to a variety of fields. In the overall, rather than engineering a solution given a specific set of input point clouds, both methods have shown how a general purpose approach can be successfully applied, and compare favourably with other methods which are limited to a specific context.

8.3 Impact

The work presented in this thesis impacts a number of areas that employ point cloud processing. PaRSe enhances current point cloud processing pipelines in fields such as urban planning, architecture and manufacturing by automating the extraction of specific parts of a scene. A variety of user-friendly GUI tools may be built, based on PaRSe structure graphs, to facilitate the management of point clouds representing complex sites. Such tools could include graph operations to further refine the initial automatic approximation provided by PaRSe and also allow for the embedding of additional semantics into segment nodes, for instance, including photographs from parts of the site. CoFFrS would benefit architects and

Method	Approach	Labelling	Quality	Inp. Format	Context	Obj. Pose Inv.	Exp. Range
Rušu *et al.* (2008)	Unsupervised	✓	High	Range Maps	Sensitive	×	Coarse
Koppula *et al.* (2011)	Supervised	✓	Low	Range Maps	Sensitive	×	Medium
Adan & Huber (2011)	Supervised	✓	Low	Range Maps	Sensitive	×	Medium
Nan *et al.* (2012)	Supervised	✓	Low	Point Cloud	Sensitive	×	Medium
Shao *et al.* (2012)	Supervised/Interactive	✓	Low	Range Maps	Free	✓	Medium
Anand *et al.* (2012)	Supervised	✓	Low	Range Maps	Sensitive	×	Medium
Karpathy *et al.* (2013)	Supervised	✓	High	Point Cloud	Sensitive	×	High
Song & Xiao (2014)	Supervised	✓	High	Range Maps	Sensitive	×	Large
Kim *et al.* (2012)	Supervised	✓	Low	Point cloud	Sensitive	×	Medium
Mattausch *et al.* (2014)	Unsupervised	×	High	Point cloud	Sensitive	✓	Large
Spina *et al.* (2014)	Supervised/Pattern Search	✓	Low	Point Cloud	Free	✓	Medium

Table 8.1: Feature comparison of indoor scene understanding systems.

interior designers by automatically producing CAD models from point clouds of existing places. Point cloud visualisation is an important activity in many fields, where PaRSe could be used in rendering algorithms to guide segment specific tessellation and level-of-detail.

PaRSe and CoFFrS also benefit point cloud acquisition methods by providing real-time feedback while scanning and directing the acquisition device towards areas which are not sufficiently sampled. The entertainment industry, amongst others, has recently started looking at the use of point cloud data in augmented reality (AR) applications. Immersion into these systems is only possible if the application is able to determine the objects surrounding the person. CoFFrS positions itself as a viable solution to accomplish this task. With the advances seen in both 3D scanners and drones, future site acquisition systems will be looking into methods for integrating these two technologies together. Both PaRSe and CoFFrS would impact the development of such systems by providing a first step into context-free mechanisms which would enable reasoning about the acquired data while mapping the environment. Moreover, the out-of-core methods implemented in PaRSe would be suitable for devices mounted on drones, which typically have limited amounts of working memory.

8.4 Limitations and Future Work

This section outlines some limitations of the work presented in this thesis together with possible avenues for future work, which could address these limitations.

A general purpose segmentation algorithm tries to maintain a balance between over and under-segmentation. Since over-segmentation is generally preferred to under-segmentation, joining segments is easier than splitting them, PaRSe currently favours over-segmentation. PaRSe generally suffers from over-segmentation when the surfaces sampled are very rough, with the creation of many small surface segments interspersed with edge segments, or vice-versa. This could be avoided by looking into a mechanism which dynamically adapts the number of neighbours used during the region-growing phase, and also by factoring in surface normal in the growing criteria. Alternatively, mechanisms to switch the type of individual points based on the types of neighbouring points can be considered. The straddling between over and under-segmentation is governed by a number of parameters in PaRSe which are currently user specified. Future

work would investigate the possibility of automatically setting these values, for instance, by formulating segmentation as an optimisation problem.

Given the context-free nature of CoFFrS, the mechanism for detecting boundary segments (walls, floor and ceiling) of an indoor environment is currently not very robust. For instance, if a wall does not clearly exist, then segments belonging to other objects might be labelled as boundaries. Further work is required for the development of a query graph which can encapsulate richer semantics and therefore can be used to more robustly determine the presence or absence of a boundary, whilst maintaining the system context-free.

A limitation of CoFFrS is the reliance on the saliency score of planar segments. If these segments are heavily occluded and cluttered, then the system can easily fail in selecting them as anchors. Future work would investigate alternatives to the planar segment sorting mechanism which is currently based on number of points and coverage, and determine the extent to which this affects the scene interpretation performance of CoFFrS.

The field of indoor scene understanding from point clouds currently lacks a proper evaluation framework. Future work would look into establishing such a framework, which could include algorithms for procedurally generating and sampling, in a physically correct manner, virtual scenes. Such a framework would create a common base on which different techniques for indoor scene understanding can be compared.

For very large point clouds, structure graphs could easily grow into the thousands of segments. With the visualisation of segments currently adopting a simple colour per segment approach, this leads to situations where points from adjacent segments are rendered using the same colour, giving the impression these are from the same segment. Further research is required to address this visualisation challenge. Similarly, more work is necessary in order to establish adequate structure graph presentation layouts which can be easily manipulated by a user.

Point cloud segmentation methods, including PaRSe, are viewed as a post-processing task carried out after acquisition. However, a variety of benefits may be obtained if the segmentation and scene understanding processes are interleaved with scene acquisition. For instance, including the possibility of positioning and focusing the scanner on areas which, based on some quality criteria, have not been properly sampled. In future work, the structure graph produced by PaRSe could be used to determine optimal scanning positions and trajectories. For instance, during an site acquisition session, a user might decide that newly acquired

points falling within the OBB of specified segments are not added to the point cloud (e.g. to discard point samples from trees). In the case of CoFFrS, real-time feedback to the acquisition module, can be used to direct the scanner to surfaces which could help disambiguate between similar objects. Future work will look at a GPU based object graph and grid matching implementations, which would considerably contribute towards the goal of achieving an online scene understanding process. In many cases, scene understanding ambiguity is a result of a poor quality point cloud, which can be improved if real-time feedback is used to guide the acquisition stage. Improvements to PaRSe and CoFFrS along this direction would pave the way to the creation of an automated mobile acquisition system.

8.5 Final Remarks

The many advances in acquisition hardware meant to capture the world around us in 3D have resulted in the popularisation and wider utility of point cloud data. The work in this thesis contributes to the body of knowledge pertaining to point cloud processing by presenting generic and context-free segmentation and scene understanding methods. PaRSe provides a segmentation method to accelerate post-processing tasks which can be applied to a variety of tasks in different fields. A novel out-of-core method for the computation of the k-NN of a point is included in PaRSe to cater for very large point clouds when these are processed on machines with limited main memory. CoFFrS extends the data structures proposed in PaRSe to coarsely describe indoor scene objects and presents a scene understanding framework which incorporates searching for patterns representing varying structures and trained objects descriptors. This thesis has addressed a number of important research challenges faced in point cloud segmentation and indoor scene understanding, providing a firm foundation from which future research can build.

Bibliography

Abuzaina, A., Nixon, M. S. & Carter, J. N. (2013). Sphere detection in kinect point clouds via the 3d hough transform, *Computer Analysis of Images and Patterns*, Springer, pp. 290–297.

Adan, A. & Huber, D. (2011). 3d reconstruction of interior wall surfaces under occlusion and clutter, *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2011 International Conference on*, IEEE, pp. 275–281.

Anand, A., Koppula, H. S., Joachims, T. & Saxena, A. (2012). Contextually guided semantic labeling and search for three-dimensional point clouds, *The International Journal of Robotics Research* p. 0278364912461538.

Anguelov, D., Taskarf, B., Chatalbashev, V., Koller, D., Gupta, D., Heitz, G. & Ng, A. (2005). Discriminative learning of markov random fields for segmentation of 3d scan data, *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, Vol. 2, IEEE, pp. 169–176.

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R. & Wu, A. Y. (1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions, *Journal of the ACM (JACM)* **45**(6): 891–923.

Asus, X. P. (2015). Xtion pro sensor, *URL [http://http://www. asus. co. jp/Multimedia/Motion_Sensor/Xtion_PRO_LIVE/]* .

Bab-Hadiashar, A. & Gheissari, N. (2006). Range image segmentation using surface selection criterion, *Image Processing, IEEE Transactions on* **15**(7).

Bailey, T. & Durrant-Whyte, H. (2006). Simultaneous localization and mapping (slam): Part ii, *IEEE Robotics & Automation Magazine* **13**(3): 108–117.

Bayer, R. (1972). Symmetric binary b-trees: Data structure and maintenance algorithms, *Acta informatica* **1**(4): 290–306.

Belongie, S., Malik, J. & Puzicha, J. (2002). Shape matching and object recognition using shape contexts, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **24**(4): 509–522.

Bernardini, F. & Rushmeier, H. (2002). The 3d model acquisition pipeline, *Computer graphics forum*, Vol. 21, Wiley Online Library, pp. 149–172.

Besl, P. J. & Jain, R. C. (1985). Three-dimensional object recognition, *ACM Computing Surveys (CSUR)* **17**(1): 75–145.

Biswas, J. & Veloso, M. (2012). Depth camera based indoor mobile robot localization and navigation, *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, IEEE, pp. 1697–1702.

Borrmann, D., Elseberg, J., Lingemann, K. & Nüchter, A. (2011). The 3d hough transform for plane detection in point clouds: A review and a new accumulator design, *3D Research* **2**(2): 1–13.

Boykov, Y. & Funka-Lea, G. (2006). Graph cuts and efficient nd image segmentation, *International journal of computer vision* **70**(2): 109–131.

Breiman, L. (2001). Random forests, *Machine learning* **45**(1): 5–32.

Bro, R., Acar, E. & Kolda, T. G. (2008). Resolving the sign ambiguity in the singular value decomposition, *Journal of Chemometrics* **22**(2): 135–140.

Budroni, A. & Böhm, J. (2009). Toward automatic reconstruction of interiors from laser data, *Proceedings of Virtual Reconstruction and Visualization of Complex Architectures (3D-Arch)* .

Bustos, B., Keim, D., Saupe, D. & Schreck, T. (2007). Content-based 3d object retrieval, *Computer Graphics and Applications, IEEE* **27**(4): 22–27.

Camurri, M., Vezzani, R. & Cucchiara, R. (2014). 3d hough transform for sphere recognition on point clouds, *Machine Vision and Applications* **25**(7): 1877–1891.

Chaperon, T., Goulette, F. & Laurgeau, C. (2001). Extracting cylinders in full 3d data using a random sampling method and the gaussian image., *VMV*, Vol. 1, Citeseer, pp. 35–42.

Chen, H. & Bhanu, B. (2007). 3d free-form object recognition in range images using local surface patches, *Pattern Recognition Letters* **28**(10): 1252–1262.

Chen, X., Golovinskiy, A. & Funkhouser, T. (2009). A benchmark for 3d mesh segmentation, *ACM Transactions on Graphics (TOG)*, Vol. 28, ACM, p. 73.

Chetverikov, D., Svirko, D., Stepanov, D. & Krsek, P. (2002). The trimmed iterative closest point algorithm, *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, Vol. 3, IEEE, pp. 545–548.

Chua, C. S. & Jarvis, R. (1997). Point signatures: A new representation for 3d object recognition, *International Journal of Computer Vision* **25**(1): 63–85.

Cignoni, P., Corsini, M. & Ranzuglia, G. (2008). Meshlab: an open-source 3d mesh processing system, *Ercim news* **73**: 45–46.

Cignoni, P. & Scopigno, R. (2008). Sampled 3d models for ch applications: A viable and enabling new medium or just a technological exercise?, *Journal on Computing and Cultural Heritage (JOCCH)* **1**(1): 2.

Clarkson, K. L. (1983). Fast algorithms for the all nearest neighbors problem, *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, IEEE, pp. 226–232.

Cohen-Steiner, D., Alliez, P. & Desbrun, M. (2004). Variational shape approximation, *ACM Transactions on Graphics (TOG)*, Vol. 23, ACM, pp. 905–914.

Daras, P. & Axenopoulos, A. (2010). A 3d shape retrieval framework supporting multimodal queries, *International Journal of Computer Vision* **89**(2-3): 229–247.

Delingette, H. (1999). General object reconstruction based on simplex meshes, *International Journal of Computer Vision* **32**(2): 111–146.

Denning, P. J. (1970). Virtual memory, *ACM Computing Surveys* **2**: 153–189.

Deschaud, J.-E. & Goulette, F. (2010). A fast and accurate plane detection algorithm for large noisy point clouds using filtered normals and voxel growing, *in* B. Fisher & C. Theobalt (eds), *3D Data Processing, Visualization and Transmission*, Eurographics Association, Paris, France.

Do Carmo, M. P. & Do Carmo, M. P. (1976). *Differential geometry of curves and surfaces*, Vol. 2, Prentice-Hall Englewood Cliffs.

Dorninger, P. & Nothegger, C. (2007). 3d segmentation of unstructured point clouds for building modelling, *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* **35**(3/W49A): 191–196.

Douillard, B., Underwood, J., Kuntz, N., Vlaskine, V., Quadros, A., Morton, P. & Frenkel, A. (2011). On the segmentation of 3d lidar point clouds, *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, pp. 2798–2805.

Elseberg, J., Borrmann, D. & Nuchter, A. (2011). Efficient processing of large 3d point clouds, *Information, Communication and Automation Technologies (ICAT), 2011 XXIII International Symposium on*, IEEE, pp. 1–7.

Falcidieno, B. (2004). Aim@ shape project presentation, *Shape Modeling Applications, 2004. Proceedings*, IEEE, p. 329.

Fan, T.-J., Medioni, G. & Nevatia, R. (1989). Recognizing 3-d objects using surface descriptions, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **11**(11): 1140–1157.

Faugeras, O. D. & Hebert, M. (1986). The representation, recognition, and locating of 3-d objects, *Int. J. Rob. Res.* **5**(3): 27–52.
URL: *http://dx.doi.org/10.1177/027836498600500302*

Fischler, M. A. & Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, *Communications of the ACM* **24**: 381–395.

Fisher, M., Savva, M. & Hanrahan, P. (2011). Characterizing structural relationships in scenes using graph kernels, *ACM Transactions on Graphics (TOG)*, Vol. 30, ACM, p. 34.

Friedman, J. H., Bentley, J. L. & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time, *ACM Transactions on Mathematical Software (TOMS)* **3**(3): 209–226.

Frome, A., Huber, D., Kolluri, R., Bülow, T. & Malik, J. (2004). Recognizing objects in range data using regional point descriptors, *Computer Vision-ECCV 2004*, Springer, pp. 224–237.

Frueh, C., Jain, S. & Zakhor, A. (2005). Data processing algorithms for generating textured 3d building facade meshes from laser scans and camera images, *International Journal of Computer Vision* **61**(2): 159–184.

Gao, J. & Yang, R. (2013). Online building segmentation from ground-based lidar data in urban scenes, *3DTV-Conference, 2013 International Conference on*, IEEE, pp. 49–55.

Golovinskiy, A. & Funkhouser, T. (2009). Min-cut based segmentation of point clouds, *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, IEEE, pp. 39–46.

Golovinskiy, A., Kim, V. G. & Funkhouser, T. (2009). Shape-based recognition of 3d point clouds in urban environments, *Computer Vision, 2009 IEEE 12th International Conference on*, IEEE, pp. 2154–2161.

Google (2014). Project tango @ONLINE.
URL: *https://www.google.com/atap/projecttango*

Gotardo, P. F., Bellon, O. R. P. & Silva, L. (2003). Range image segmentation by surface extraction using an improved robust estimator, *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, Vol. 2, IEEE, pp. II–33.

Graham, R. L. & Hell, P. (1985). On the history of the minimum spanning tree problem, *Annals of the History of Computing* **7**(1): 43–57.

Gross, M. & Pfister, H. (2011). *Point-based graphics*, Morgan Kaufmann.

Gumhold, S., Wang, X. & MacLeod, R. (2001). Feature extraction from point clouds, *Proceedings of 10th international meshing roundtable*, Vol. 2001, Citeseer.

Hähnel, D., Burgard, W. & Thrun, S. (2003). Learning compact 3d models of indoor and outdoor environments with a mobile robot, *Robotics and Autonomous Systems* **44**(1): 15–27.

Hearst, M. A., Dumais, S., Osman, E., Platt, J. & Scholkopf, B. (1998). Support vector machines, *Intelligent Systems and their Applications, IEEE* **13**(4): 18–28.

Hetzel, G., Leibe, B., Levi, P. & Schiele, B. (2001). 3d object recognition from range images using local feature histograms, *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, Vol. 2, IEEE, pp. II–394.

Hoover, A., Jean-Baptiste, G., Jiang, X., Flynn, P. J., Bunke, H., Goldgof, D. B., Bowyer, K., Eggert, D. W., Fitzgibbon, A. & Fisher, R. B. (1996). An experimental comparison of range image segmentation algorithms, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **18**(7): 673–689.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. & Stuetzle, W. (1992). *Surface reconstruction from unorganized points*, Vol. 26, ACM.

Hu, J., You, S. & Neumann, U. (2003). Approaches to large-scale urban modeling, *Computer Graphics and Applications, IEEE* **23**(6): 62–69.

Jaakkola, A., Hyyppä, J., Hyyppä, H. & Kukko, A. (2008). Retrieval algorithms for road surface modelling using laser-based mobile mapping, *Sensors* **8**(9): 5238–5249.

Jared, D. (2014). Structure offers 3d scanning right on your ipad @ONLINE. **URL:** *http://www.imore.com/ceslive-scan-your-world-structure-sensor*

Johnson, A. E. (1997). *Spin-images: a representation for 3-D surface matching*, PhD thesis, Citeseer.

Johnson, A. E. & Hebert, M. (1999). Using spin images for efficient object recognition in cluttered 3d scenes, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **21**(5): 433–449.

Juan, L. & Gwun, O. (2009). A comparison of sift, pca-sift and surf, *International Journal of Image Processing (IJIP)* **3**(4): 143–152.

Karpathy, A., Miller, S. & Fei-Fei, L. (2013). Object discovery in 3d scenes via shape analysis, *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, IEEE, pp. 2088–2095.

Kazhdan, M., Funkhouser, T. & Rusinkiewicz, S. (2003). Rotation invariant spherical harmonic representation of 3 d shape descriptors, *Symposium on geometry processing*, Vol. 6.

Kim, Y. M., Mitra, N. J., Yan, D.-M. & Guibas, L. (2012). Acquiring 3d indoor environments with variability and repetition, *ACM Transactions on Graphics (TOG)* **31**(6): 138.

Kindermann, R., Snell, J. L. *et al.* (1980). *Markov random fields and their applications*, Vol. 1, American Mathematical Society Providence, RI.

Koenderink, J. J. & van Doorn, A. J. (1992). Surface shape and curvature scales, *Image and vision computing* **10**(8): 557–564.

Kolb, A., Barth, E., Koch, R. & Larsen, R. (2009). Time-of-flight sensors in computer graphics, *Proc. Eurographics (State-of-the-Art Report)*, pp. 119–134.

Koppula, H. S., Anand, A., Joachims, T. & Saxena, A. (2011). Semantic labeling of 3d point clouds for indoor scenes, *Advances in Neural Information Processing Systems*, pp. 244–252.

Lafarge, F. & Alliez, P. (2013). Surface reconstruction through point set structuring, *Computer Graphics Forum*, Vol. 32, Wiley Online Library, pp. 225–234.

Lafferty, J., McCallum, A. & Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Lai, K., Bo, L., Ren, X. & Fox, D. (2011). A large-scale hierarchical multi-view rgb-d object dataset, *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, pp. 1817–1824.

Lalonde, J.-F., Vandapel, N., Huber, D. F. & Hebert, M. (2006). Natural terrain classification using three-dimensional ladar data for ground robot mobility, *Journal of field robotics* **23**(10): 839–861.

Lavoué, G., Vandeborre, J.-P., Benhabiles, H., Daoudi, M., Huebner, K., Mortara, M. & Spagnuolo, M. (2012). Shrec'12 track: 3d mesh segmentation,

Proceedings of the 5th Eurographics conference on 3D Object Retrieval, Eurographics Association, pp. 93–99.

Lee, D.-T. & Schachter, B. J. (1980). Two algorithms for constructing a delaunay triangulation, *International Journal of Computer & Information Sciences* **9**(3): 219–242.

Lew, M. S., Sebe, N., Djeraba, C. & Jain, R. (2006). Content-based multimedia information retrieval: State of the art and challenges, *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)* **2**(1): 1–19.

Li, B., Godil, A., Aono, M., Bai, X., Furuya, T., Li, L., López-Sastre, R., Johan, H., Ohbuchi, R., Redondo-Cabrera, C. *et al.* (2012). Shrec'12 track: generic 3d shape retrieval, *Proceedings of the 5th Eurographics conference on 3D Object Retrieval*, Eurographics Association, pp. 119–126.

Liaw, A. & Wiener, M. (2002). Classification and regression by random forest, *R news* **2**(3): 18–22.

Lin, H., Gao, J., Zhou, Y., Lu, G., Ye, M., Zhang, C., Liu, L. & Yang, R. (2013). Semantic decomposition and reconstruction of residential scenes from lidar data., *ACM Trans. Graph.* **32**(4): 66.

Lipson, H. & Kurman, M. (2013). *Fabricated: The new world of 3D printing*, John Wiley & Sons.

Lowe, D. G. (1999). Object recognition from local scale-invariant features, *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, Vol. 2, Ieee, pp. 1150–1157.

Malisiewicz, T., Gupta, A. & Efros, A. A. (2011). Ensemble of exemplar-svms for object detection and beyond, *Computer Vision (ICCV), 2011 IEEE International Conference on*, IEEE, pp. 89–96.

Maple, C. & Wang, Y. (2004). A three-dimensional object similarity test using graph matching, *Information Visualisation, 2004. IV 2004. Proceedings. Eighth International Conference on*, IEEE, pp. 363–369.

Marr, D. & Poggio, T. (1979). A computational theory of human stereo vision, *Proceedings of the Royal Society of London. Series B. Biological Sciences* **204**(1156): 301–328.

Mattausch, O., Panozzo, D., Mura, C., Sorkine-Hornung, O. & Pajarola, R. (2014). Object detection and classification from large-scale cluttered indoor scans, *Computer Graphics Forum*, Vol. 33, Wiley Online Library, pp. 11–21.

Mikolajczyk, K. & Schmid, C. (2005). A performance evaluation of local descriptors, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **27**(10): 1615–1630.

Moosmann, F., Pink, O. & Stiller, C. (2009). Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion, *Intelligent Vehicles Symposium, 2009 IEEE*, IEEE, pp. 215–220.

Muja, M. & Lowe, D. (2009a). Flann-fast library for approximate nearest neighbors user manual, *Computer Science Department, University of British Columbia, Vancouver, BC, Canada* .

Muja, M. & Lowe, D. G. (2009b). Fast approximate nearest neighbors with automatic algorithm configuration, *International Conference on Computer Vision Theory and Application VISSAPP'09)*, INSTICC Press, pp. 331–340.

Mura, C., Mattausch, O., Villanueva, A. J., Gobbetti, E. & Pajarola, R. (2013). Robust reconstruction of interior building structures with multiple rooms under clutter and occlusions, *Computer-Aided Design and Computer Graphics (CAD/Graphics), 2013 International Conference on*, IEEE, pp. 52–59.

Nan, L., Xie, K. & Sharf, A. (2012). A search-classify approach for cluttered indoor scene understanding, *ACM Transactions on Graphics (TOG)* **31**(6): 137.

Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohi, P., Shotton, J., Hodges, S. & Fitzgibbon, A. (2011). Kinectfusion: Real-time dense surface mapping and tracking, *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, IEEE, pp. 127–136.

Nguatem, W., Drauschke, M. & Mayer, H. (2014). Localization of windows and doors in 3d point clouds of facades, *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* **1**: 87–94.

Niklas, K. J. & Spatz, H.-C. (2006). Allometric theory and the mechanical stability of large trees: proof and conjecture, *American journal of botany* **93**(6): 824–828.

Ning, X., Zhang, X. & Wang, Y. (2009). Tree segmentation from scanned scene data, *Plant Growth Modeling, Simulation, Visualization and Applications (PMA), 2009 Third International Symposium on*, IEEE, pp. 360–367.

Nistér, D. (2005). Preemptive ransac for live structure and motion estimation, *Machine Vision and Applications* **16**(5): 321–329.

Novatnack, J. & Nishino, K. (2008). Scale-dependent/invariant local 3d shape descriptors for fully automatic registration of multiple sets of range images, *Computer Vision–ECCV 2008*, Springer, pp. 440–453.

Occipital, S. S. (2015). Occipital structure sensor, *URL [http://http://structure.io/]* .

Oehler, B., Stueckler, J., Welle, J., Schulz, D. & Behnke, S. (2011). Efficient multi-resolution plane segmentation of 3d point clouds, *Intelligent Robotics and Applications*, Springer, pp. 145–156.

Oppenheim, A. V., Schafer, R. W., Buck, J. R. *et al.* (1989). *Discrete-time signal processing*, Vol. 2, Prentice-hall Englewood Cliffs.

Pal, N. R. & Pal, S. K. (1993). A review on image segmentation techniques, *Pattern recognition* **26**(9): 1277–1294.

Pauling, F., Bosse, M. & Zlot, R. (2009). Automatic segmentation of 3d laser point clouds by ellipsoidal region growing, *Australasian Conference on Robotics and Automation*, p. 10.

Pauly, M., Gross, M. & Kobbelt, L. P. (2002). Efficient simplification of point-sampled surfaces, *Proceedings of the conference on Visualization'02*, IEEE Computer Society, pp. 163–170.

Pauly, M., Mitra, N. J. & Guibas, L. (2004). Uncertainty and variability in point cloud surface data, *Symposium on point-based graphics*, Vol. 9.

Petrov, M., Talapov, A., Robertson, T., Lebedev, A., Zhilyaev, A. & Polonskiy, L. (1998). Optical 3d digitizers: Bringing life to the virtual world, *IEEE Computer Graphics and Applications* **18**(3): 28–37.

Poppinga, J., Vaskevicius, N., Birk, A. & Pathak, K. (2008). Fast plane detection and polygonalization in noisy 3d range images, *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, IEEE, pp. 3378–3383.

Pu, S., Vosselman, G. *et al.* (2006). Automatic extraction of building features from terrestrial laser scanning, *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* **36**(5): 25–27.

Puterman, M. L. (2009). *Markov decision processes: discrete stochastic dynamic programming*, Vol. 414, John Wiley & Sons.

Ribo, M. & Brandner, M. (2005). State of the art on vision-based structured light systems for 3d measurements, *Robotic Sensors: Robotic and Sensor Environments, 2005. International Workshop on*, IEEE, pp. 2–6.

Rinaudo, F., Chiabrando, F., Nex, F. & Piatti, D. (2010). New instruments and technologies for cultural heritage survey: full integration between point clouds and digital photogrammetry, *Digital Heritage*, Springer, pp. 56–70.

Robbani & Vosselman (2006). Segmentation of point clouds using smoothness constraint, *Image Engineering and Vision Metrology*.

Roth, G. & Levine, M. D. (1993). Extracting geometric primitives, *CVGIP: Image Understanding* **58**(1): 1–22.

Rublee, E., Rabaud, V., Konolige, K. & Bradski, G. (2011). Orb: an efficient alternative to sift or surf, *Computer Vision (ICCV), 2011 IEEE International Conference on*, IEEE, pp. 2564–2571.

Ruiz-Correa, S., Shapiro, L. G. & Meila, M. (2003). A new paradigm for recognizing 3-d objects from range data, *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, IEEE, pp. 1126–1133.

Ruiz-Correa, S., Shapiro, L. G. & Melia, M. (2001). A new signature-based method for efficient 3-d object recognition, *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, Vol. 1, IEEE, pp. I–769.

Rusu, R. B., Blodow, N. & Beetz, M. (2009). Fast point feature histograms (fpfh) for 3d registration, *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, IEEE, pp. 3212–3217.

Rusu, R. B., Blodow, N., Marton, Z. C. & Beetz, M. (2008). Aligning point cloud views using persistent feature histograms, *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, IEEE, pp. 3384–3391.

Rusu, R. B., Bradski, G., Thibaux, R. & Hsu, J. (2010). Fast 3d recognition and pose using the viewpoint feature histogram, *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, IEEE, pp. 2155–2162.

Rusu, R. B. & Cousins, S. (2011). 3d is here: Point cloud library (pcl), *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, pp. 1–4.

Rusu, R. B., Marton, Z. C., Blodow, N. & Beetz, M. (2008). Learning informative point classes for the acquisition of object model maps, *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, IEEE, pp. 643–650.

Ruther, H. (2010a). Documenting africa's cultural heritage, *In Proceedings of the 11th International Symposium VAST. Virtual Reality, Archaeology and Cultural Heritage*.

Ruther, H. (2010b). Documenting africa's cultural heritage, *In Proceedings of the 11th International Symposium VAST. Virtual Reality, Archaeology and Cultural Heritage*.

Rušu, R., Marton, Z., Blodow, N., Dolha, M. & Beetz, M. (2008). Towards 3d point cloud based object maps for household environments, *Robotics and Autonomous Systems* **56**(11): 927–941.

Sankaranarayanan, J., Samet, H. & Varshney, A. (2007). A fast all nearest neighbor algorithm for applications involving large point-clouds, *Computers & Graphics* **31**(2): 157–174.

Schnabel, R., Degener, P. & Klein, R. (2009). Completion and reconstruction with primitive shapes, *Computer Graphics Forum*, Vol. 28, Wiley Online Library, pp. 503–512.

Schnabel, R., Wahl, R. & Klein, R. (2007). Efficient ransac for point-cloud shape detection, *Computer Graphics Forum*, Vol. 26, Wiley Online Library, pp. 214–226.

Schnabel, R., Wessel, R., Wahl, R. & Klein, R. (2008). Shape recognition in 3d point-clouds, *Proc. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision*, Vol. 2, Citeseer.

Scholkopf, B., Sung, K.-K., Burges, C. J., Girosi, F., Niyogi, P., Poggio, T. & Vapnik, V. (1997). Comparing support vector machines with gaussian kernels to radial basis function classifiers, *Signal Processing, IEEE Transactions on* **45**(11): 2758–2765.

Shafer, G. *et al.* (1976). *A mathematical theory of evidence*, Vol. 1, Princeton university press Princeton.

Shamir, A. (2008). A survey on mesh segmentation techniques, *Computer graphics forum*, Vol. 27, Wiley Online Library, pp. 1539–1556.

Shao, T., Xu, W., Zhou, K., Wang, J., Li, D. & Guo, B. (2012). An interactive approach to semantic modeling of indoor scenes with an rgbd camera, *ACM Transactions on Graphics (TOG)* **31**(6): 136.

Shi, J. & Malik, J. (2000). Normalized cuts and image segmentation, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **22**(8): 888–905.

Song, S. & Xiao, J. (2014). Sliding shapes for 3d object detection in depth images, *Computer Vision–ECCV 2014*, Springer, pp. 634–651.

Sonka, M., Hlavac, V. & Boyle, R. (2014). *Image processing, analysis, and machine vision*, Cengage Learning.

Spina, S., Debattista, K., Bugeja, K. & Chalmers, A. (2014). Scene segmentation and understanding for context-free point clouds, *Pacific Conference on Computer Graphics and Applications - Short Papers*, Eurographics Association, Seoul, Korea, pp. 7–12.

Stamos, I. & Allen, P. (2000). 3-d model construction using range and image data, *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, Vol. 1, IEEE, pp. 531–536.

Stein, F. & Medioni, G. (1992). Structural indexing: Efficient 3-d object recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14**(2): 125–145.

Stoer, M. & Wagner, F. (1997). A simple min-cut algorithm, *Journal of the ACM (JACM)* **44**(4): 585–591.

Sun, Y. & Abidi, M. A. (2001). Surface matching by 3d point's fingerprint, *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, Vol. 2, IEEE, pp. 263–269.

Sundstedt, V., Chalmers, A. & Martinez, P. (2004). High fidelity reconstruction of the ancient egyptian temple of kalabsha, *Proceedings of the 3rd international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, ACM, pp. 107–113.

Tang, P., Huber, D., Akinci, B., Lipman, R. & Lytle, A. (2010). Automatic reconstruction of as-built building information models from laser-scanned point clouds: A review of related techniques, *Automation in construction* **19**(7): 829–843.

Tangelder, J. W. & Veltkamp, R. C. (2008). A survey of content based 3d shape retrieval methods, *Multimedia tools and applications* **39**(3): 441–471.

Tarsha-Kurdi, F., Landes, T., Grussenmeyer, P. *et al.* (2007). Hough-transform and extended ransac algorithms for automatic detection of 3d building roof planes from lidar data, *International Archives of Photogrammetry, Remote Sensing and Spatial Information Systems* **36**: 407–412.

Thrun, S., Martin, C., Liu, Y., Hahnel, D., Emery-Montemerlo, R., Chakrabarti, D. & Burgard, W. (2004). A real-time expectation-maximization algorithm for acquiring multiplanar maps of indoor environments with mobile robots, *Robotics and Automation, IEEE Transactions on* **20**(3): 433–443.

Tisserand, N. & Burrus, N. (2015). Skanect scanning software, *URL [http://skanect.occipital.com/]* .

Tombari, F., Salti, S. & Di Stefano, L. (2010). Unique signatures of histograms for local surface description, *Computer Vision–ECCV 2010*, Springer, pp. 356–369.

Unnikrishnan, R. & Hebert, M. (2003). Robust extraction of multiple structures from non-uniformly sampled data, *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, Vol. 2, IEEE, pp. 1322–1329.

Vaidya, P. M. (1989). Ano (n logn) algorithm for the all-nearest-neighbors problem, *Discrete & Computational Geometry* **4**(1): 101–115.

Van De Geer, S. & Van De Geer, S. (2000). *Empirical Processes in M-estimation*, Vol. 105, Cambridge university press Cambridge.

Vosselman, G., Gorte, B. G., Sithole, G. & Rabbani, T. (2004). Recognising structure in laser scanner point clouds, *International archives of photogrammetry, remote sensing and spatial information sciences* **46**(8): 33–38.

Wang, Y., Weinacker, H. & Koch, B. (2008). A lidar point cloud based procedure for vertical canopy structure analysis and 3d single tree modelling in forest, *Sensors* **8**(6): 3938–3951.

Weyrich, T., Pauly, M., Keiser, R., Heinzle, S., Scandella, S. & Gross, M. (2004). Post-processing of scanned 3d surface data, *Proceedings of the First Eurographics conference on Point-Based Graphics*, Eurographics Association, pp. 85–94.

Witten, I. H., Frank, E., Trigg, L. E., Hall, M. A., Holmes, G. & Cunningham, S. J. (1999). Weka: Practical machine learning tools and techniques with java implementations.

Wu, Z. & Leahy, R. (1993). An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **15**(11): 1101–1113.

Xu, H., Gossett, N. & Chen, B. (2007). Knowledge and heuristic-based modeling of laser-scanned trees, *ACM Transactions on Graphics (TOG)* **26**(4): 19.

Yamany, S. M. & Farag, A. A. (2002). Surface signatures: an orientation independent free-form surface representation scheme for the purpose of objects registration and matching, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **24**(8): 1105–1120.

Yastikli, N. (2007). Documentation of cultural heritage using digital photogrammetry and laser scanning, *Journal of Cultural Heritage* **8**(4): 423–427.

Zhang, H., Fritts, J. E. & Goldman, S. A. (2008). Image segmentation evaluation: A survey of unsupervised methods, *computer vision and image understanding* **110**(2): 260–280.

Zhang, S. & Yau, S.-T. (2006). High-resolution, real-time 3d absolute coordinate measurement based on a phase-shifting method, *Optics Express* **14**(7): 2644–2649.

Zhang, Z. (2012). Microsoft kinect sensor and its effect, *MultiMedia, IEEE* **19**(2): 4–10.

Zhao, H., Liu, Y., Zhu, X., Zhao, Y. & Zha, H. (2010). Scene understanding in a large dynamic environment through a laser-based sensing, *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, IEEE, pp. 127–133.

Zhou, Y., Wang, D., Xie, X., Ren, Y., Li, G., Deng, Y. & Wang, Z. (2014). A fast and accurate segmentation method for ordered lidar point cloud of large-scale scenes, *Geoscience and Remote Sensing Letters, IEEE* **11**: 1981–1985.