UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FREDERICO ARTUR LIMBERGER

Real-Time Detection of Planar Regions in Unorganized Point Clouds

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Prof. Dr. Manuel Menezes de Oliveira Neto
Advisor

Porto Alegre, April 2014

CIP – CATALOGING-IN-PUBLICATION

Artur Limberger, Frederico

Real-Time Detection of Planar Regions in Unorganized Point Clouds / Frederico Artur Limberger. – Porto Alegre: PPGC da UFRGS, 2014.

66 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2014. Advisor: Manuel Menezes de Oliveira Neto.

1. Plane detection. 2. Hough transform. 3. Unorganized point clouds. I. Oliveira Neto, Manuel Menezes de. II. Title.

ACKNOWLEDGEMENTS

I would like to thank specially my master advisor, Manuel Menezes de Oliveira Neto. I have learned many things since I became his student. He instructed me how to write a paper, how to search literature and how to build a presentation, briefly how to do top quality research. I am also grateful to Leandro Augusto Frata Fernandes, Cláudio Rosito Jung and João Luiz Dihl Comba for spending time reading this thesis and providing useful suggestions about it. I would like to thank also my professors from UFRGS and UFSM that provide me fundamentals and instruments to develop this thesis. My thanks also to the National Council for Scientific and Technological Development (CNPq) for their financial support.

During the period of two years, many friends were essential to my growth. I am very thankful to all my colleagues and friends which shared my journey to become a master: Vinícius C. Azevedo, Rosália G. Schneider, Bernardo Henz, André G. Pereira, Victor C. Schetinger, Cristiano Reis, André L. B. da Rosa, Matheus L. Krueger, Daniele Fernandes, Joyce Bastos, Cleóbulo S. Neto, Victor A. Oliveira, Lucyana Vulcão, Jorge Carvalho, Leandro Coutinho, Silvia Nunes, Hernandi Filho, Tales Miranda, Jerônimo G. Grandi, Vitor A. Jorge, Wagner Schmitt, Roger A. Leite, Gerson Groth, Augusto L. P. Nunes.

Last but not the least important, I owe more than thanks to my family members for their unconditional love and support. My parents, Marta Miriam Bracht Limberger and Décio José Limberger, my elder sister, Mabel Thais Limberger, my uncles and my cousins. I would like to thanks also the support of those who contributed in some way during this period.

“Far better it is to dare mighty things, to win glorious triumphs, even though checkered by failure, than to take rank with those poor spirits who neither enjoy much nor suffer much, because they live in the gray twilight that knows neither victory nor defeat.”

— Theodore Roosevelt, Strenuous Life

ABSTRACT

Automatic detection of planar regions in point clouds is an important step for many graphics, image processing, and computer vision applications. While laser scanners and digital photography have allowed us to capture increasingly larger datasets, previous techniques are computationally expensive, being unable to achieve real-time performance for datasets containing tens of thousands of points, even when detection is performed in a non-deterministic way. We present a deterministic technique for plane detection in unorganized point clouds whose cost is $O(n \log n)$ in the number of input samples. It is based on an efficient Hough-transform voting scheme and works by clustering approximately co-planar points and by casting votes for these clusters on a spherical accumulator using a trivariate Gaussian kernel. A comparison with competing techniques shows that our approach is considerably faster and scales significantly better than previous ones, being the first practical solution for deterministic plane detection in large unorganized point clouds.

Keywords: Plane detection, hough transform, unorganized point clouds.

Detecção em Tempo Real de Regiões Planares em Nuvens de Pontos Não Estruturadas

RESUMO

Detecção automática de regiões planares em nuvens de pontos é um importante passo para muitas aplicações gráficas, de processamento de imagens e de visão computacional. Enquanto a disponibilidade de digitalizadores a laser e a fotografia digital tem nos permitido capturar nuvens de pontos cada vez maiores, técnicas anteriores para detecção de planos são computacionalmente caras, sendo incapazes de alcançar desempenho em tempo real para conjunto de dados contendo dezenas de milhares de pontos, mesmo quando a detecção é feita de um modo não determinístico. Apresentamos uma abordagem determinística para detecção de planos em nuvens de pontos não estruturadas que apresenta complexidade computacional $O(n \log n)$ no número de amostras de entrada. Ela é baseada em um método eficiente de votação para a transformada de Hough. Nossa estratégia agrupa conjuntos de pontos aproximadamente coplanares e deposita votos para estes conjuntos em um acumulador esférico, utilizando núcleos Gaussianos trivariados. Uma comparação com as técnicas concorrentes mostra que nossa abordagem é consideravelmente mais rápida e escala significativamente melhor que as técnicas anteriores, sendo a primeira solução prática para detecção determinística de planos em nuvens de pontos grandes e não estruturadas.

Palavras-chave: Detecção de Planos, Transformada de Hough, Nuvens de Pontos.

LIST OF FIGURES

Figure 1.1: Example of plane detection using our technique. 16

Figure 2.1: Examples of registration errors. 21
Figure 2.2: Resulting accumulator comparing the standard voting versus the KHT voting scheme. 22
Figure 2.3: Detection of planes generated by the SHT. 23
Figure 2.4: Detection of planes generated by the PHT. 23
Figure 2.5: Detection of planes generated by the APHT. 24
Figure 2.6: Detection of planes generated by the PPHT 25
Figure 2.7: Detection of planes generated by the RHT. 25
Figure 2.8: A comparison between the *accumulator cube* and *accumulator ball*. . 26
Figure 2.9: The detection of spurious planes, depending on the used parameters. . 27
Figure 2.10: Real example of the threshold problem in RANSAC. 28
Figure 2.11: Detection of planes generated by RANSAC. 29

Figure 3.1: Adaptive octree refinement and sample clustering 36
Figure 3.2: Cluster ellipsoid defined by the eigenvalues and eigenvectors 37
Figure 3.3: A discrete representation of the 3-D spherical accumulator 39
Figure 3.4: The spherical accumulator and its representation in memory 40
Figure 3.5: Different uncertainties in the θ and ϕ dimensions, according to the position in the accumulator 42

Figure 4.1: Accuracy tests performed in the Box dataset, showing the point cloud, 3D accumulator, 3D unfolded accumulator and the planes detected by our approach 46
Figure 4.2: Point clouds and detected planes after rotating the point cloud by 20, 40, 60 and 80 degrees around the x-axis. 46
Figure 4.3: Test performed in a downsampled version of the cube. 47
Figure 4.4: Plane detection comparison of the Museum and Computer datasets using our technique, RHT and RANSAC. 49
Figure 4.5: Examples of plane detection produced by 3D KHT. 50

LIST OF TABLES

Table 4.1: Data on the experiments performed with our technique. 47
Table 4.2: Performance comparison of our approach (3D KHT) against RHT and RANSAC for various datasets. 48

TABLE OF CONTENTS

1 INTRODUCTION 15
1.1 Main Idea 15
1.2 Contributions 16
1.3 Thesis Statement 16
1.4 Structure of this Thesis 17

2 RELATED WORK 19
2.1 Point Clouds 19
2.1.1 Point Cloud Acquisition Methods 19
2.1.2 Acquisition and Registration Errors 20
2.2 Hough Transform 21
2.2.1 Standard Hough Transform 21
2.2.2 Other Hough Transform Variants 26
2.3 Random Sample Consensus 26
2.4 Surface Growing 28
2.5 Tensor Voting 29
2.6 Linear Subspace Learning 30
2.7 Summary 30

3 EFFICIENT PLANE DETECTION IN POINT CLOUDS 33
3.1 Overview 33
3.2 Clustering of Approximately Coplanar Samples 34
3.3 Computing Gaussian Trivariate Kernels for Cluster Voting 36
3.4 Cluster Voting using 3D Gaussian Distributions 38
3.5 The Spherical Accumulator 39
3.6 Peak Detection 41
3.7 Algorithm Complexity 42
3.8 Space Complexity 43
3.9 Summary 43

4 RESULTS 45
4.1 Accuracy Tests 45
4.2 Comparison with Other Methods 47
4.3 Limitations 49
4.4 Summary 49

5 CONCLUSIONS AND FUTURE WORK 51

REFERENCES 53

APPENDIX A DERIVING THE JACOBIAN MATRIX 59

APPENDIX B DETECÇÃO EM TEMPO REAL DE REGIÕES PLANARES EM NUVENS DE PONTOS NÃO ESTRUTURADAS 61

B.1 Trabalhos Relacionados 62
B.1.1 Transformada de Hough 62
B.1.2 Random Sample Consensus 63
B.1.3 Surface Growing 63
B.1.4 Tensor Voting 63
B.1.5 Linear Subspace Learning 63
B.2 Detecção Eficiente de Planos em Nuvens de Pontos 64
B.2.1 Agrupando Amostras Aproximadamente Coplanares 64
B.2.2 Calculando Núcleos Trivariados para Votação 64
B.2.3 Votando para Clusters Utilizando Distribuições Gaussianas 64
B.2.4 Detecção de Picos 65
B.3 Resultados 65
B.4 Conclusão 66

1 INTRODUCTION

Automatic plane detection in point clouds is a key component in many graphics, image processing, and computer vision applications. These include, among others, model reconstruction for reverse engineering (VOSSELMAN; DIJKMAN, 2001; KAUCIC; HARTLEY; DANO, 2001; TARSHA-KURDI; LANDES; GRUSSENMEYER, 2007; HUANG; BRENNER; SESTER, 2011; FUCHS; KEDEM; USELTON, 1977), camera calibration (TRIGGS, 1998), object recognition (ROTHWELL et al., 1995; PETERNELL; STEINER, 2004), augmented reality (SIMON; FITZGIBBON; ZISSERMAN, 2000; CHEKHLOV et al., 2007), and segmentation (BIOSCA; LERMA, 2008; NING et al., 2009). The recent popularization of laser scanners has led to an increasingly growth in the sizes of the available datasets, and point clouds containing tens of millions of samples are now commonplace. Software applications like *SynthExport* (HAUSNER, 2010) and *Photosynth* (PHOTOSYNTH, 2008) also allow us to extract point clouds from large collections of digital images. Unfortunately, existing techniques for detecting planar regions in point clouds are computationally expensive and do not scale well with the size of the datasets. For performance improvement, they often exploit non-deterministic strategies, such as working on a randomly-selected sub-set of the original samples. While this can reduce execution time, these techniques are still unable to achieve real-time performance even on datasets containing just tens of thousands of points. More importantly, their results depend on the selected sample sub-sets and, therefore, there is no guarantee that all relevant planes will be detected, or that such results will be consistent across multiple executions.

We present an efficient technique to perform deterministic plane detection in unorganized point clouds whose cost is $O(n \log n)$ in the number of input samples. Our approach scales well with the size of the datasets, is robust to the presence of noise, and handles point clouds with different characteristics in terms of dimensions and sampling distributions. While the actual running times depend on specific features of the dataset (*e.g.*, the number of planar regions), our technique is several orders of magnitude faster than previous ones. For instance, it processes an entire point cloud with 20-million samples (Bremen dataset (PHOTOSYNTH, 2008)) in just 2.105 seconds on a typical PC. In contrast, RANSAC takes more than 2 hours to process the same dataset, while the randomized Hough transform takes 42.82 seconds to process only 10% of the samples.

1.1 Main Idea

Our technique is based on a robust and fast algorithm to segment point clouds into approximately planar patches, even in the presence of noise or irregularly distributed samples. For this, we use a subdivision procedure to refine an octree and cluster groups of approximately coplanar samples. We use the identified clusters to obtain an efficient

Hough-transform voting scheme by casting votes for each of these clusters (instead of for individual samples) on a spherical accumulator. For voting, we use a Gaussian kernel centered at the cluster's best fitting plane, which takes into account the cluster's variances. In this sense, our approach extends the kernel-based voting scheme proposed by Fernandes and Oliveira (2008) using a trivariate Gaussian distribution defined over spherical coordinates (θ, ϕ, ρ). While, at first, plane detection in unorganized point clouds might seem as an immediate extension of line detection in images, the lack of explicit neighborhood information among samples imposes significant challenges, requiring new clustering and accumulation-management strategies.

Figure 1.1: Example of plane detection using our technique. (left) Museum dataset: point cloud consisting of 179,744 samples obtained from a set of photographs using SynthExport and Photosynth. (right) Planes automatically detected by our technique in just 0.025 seconds on a 3.4 GHz PC. They were resized to better represent the original model.

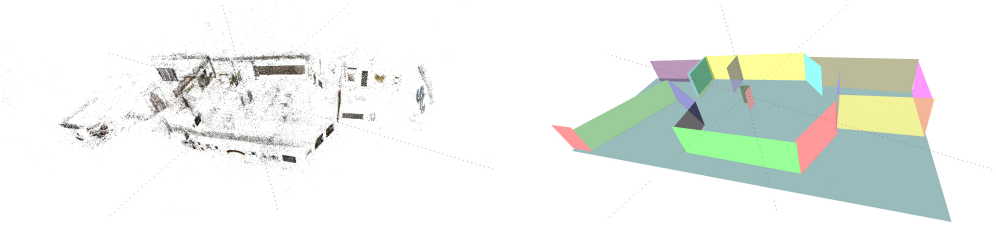


Figure 1.1 shows an example of planar regions detected using our technique. The point cloud shown on the left consists of 179,744 samples obtained from a set of photographs taken inside a museum. The samples were extracted using SynthExport (HAUSNER, 2010) and Photosynth (2008). The image on the right shows the planes detected by our technique in just 0.025 seconds on a 3.4 GHz PC, and illustrates the effectiveness of our approach.

1.2 Contributions

This thesis includes the following contributions:

- An $O(n \log n)$ deterministic Hough-transform-based technique for detecting planar regions in unorganized point clouds (Chapter 3). Our solution is robust to noise, and to sampling distributions. It is a few orders of magnitude faster and scales significantly better than existing approaches. A software implementation of our technique handles datasets with up to 10^5 points in real time on a typical PC;

- A fast Hough-transform voting strategy for plane detection (Section 3.4). Our solution uses a robust segmentation strategy to identify clusters of approximately coplanar samples. Votes are cast for clusters as opposed to for individual samples, greatly accelerating the detection process.

1.3 Thesis Statement

It is possible to increase the performance of planar detection systems to run in real-time by using a fast voting scheme. Our key observation is that the bottleneck for most plane detection techniques is the voting process, which is performed for an excessive number of possible solutions. By reducing the amount of voting, the plane detection can

be greatly accelerated. This is achieved by considering only clustered regions of points that present a plane-like spatial distribution as candidates for voting.

1.4 Structure of this Thesis

The remaining of this thesis is arranged as follows: Chapter 2 discusses related works; Chapter 3 introduces our method. Specifically, in Section 3.2 we present our clustering strategy to subdivide a point cloud; Section 3.3 describes the process for calculating the trivariate Gaussian kernels; Section 3.4 shows the cluster voting using 3D Gaussian distributions; the spherical accumulator and the peak detection procedures are addressed by Section 3.5 and Section 3.6 respectively. The results are shown in Chapter 4 with running times and comparisons. Finally, Chapter 5 summarizes the thesis and discusses directions for future exploration.

2 RELATED WORK

This Chapter discusses techniques which have been proposed for detecting planar regions in point clouds. Firstly, we introduce the key concepts and methods for point cloud registration. Afterwards, the most popular techniques to detect planes in point clouds are reviewed. They are mainly divided in five categories: *Hough transform*, *random sample consensus*, *surface growing*, *tensor voting* and *linear subspace learning*. Finally, we discuss the key issues of all techniques and how they relate to our approach.

2.1 Point Clouds

A point cloud is a set of samples represented by data points in a spatial coordinate system. They usually result from a sampling process over a continuous domain and represent some form of physical shape or object. This section addresses different kinds of point clouds and registration methods as a basis for understanding their nature and problems that may hamper detection of planes.

2.1.1 Point Cloud Acquisition Methods

There are different ways to sample points from a physical space into a point cloud, which are called acquisition methods. They usually differ by the nature of the sensor used and its application. The most common acquisition methods are **laser scanning**, **structured light**, **multi-view stereo** and **synth export** (HAUSNER, 2010).

Laser scanning methods control the steering of moving laser beams, measuring the distance at every pointing direction. LIDAR is a remote sensing scanner that uses this principle to register high resolution maps from the surface of the Earth. This has applications in many areas, such as geography, geology, archeology, and remote sensing.

Structured light is a technique that measures three-dimensional shapes by analyzing projected light patterns in objects using a camera system. A very well-known structured light capture device is the Microsoft Kinect (ZHANG, 2012). It uses an encoded infrared pattern and machine learning algorithms to interpret the scene providing depth maps.

Multi-view stereo techniques use a collection of pictures captured from different known positions to compute a 3D representation of the scene. It works by computing images correspondences by estimating pairwise disparity. There is a large number of techniques (SEITZ et al., 2006) that use this principle to reconstruct three-dimensional scenes from a collection of images.

Synth export is a tool that exports point clouds from synths on Photosynth (PHOTOSYNTH, 2008). Synths are images that are grouped based on camera parameters and geolocation, and they can be exported as a cloud of colored points (in 3-D space). There-

fore, given enouch images, these points can be used to represent (fairly) a 3d model of a photographed scene.

Some registration techniques may store more than only three-dimensional points, such as colors, connectivity, and normals. For our purpose, this information is not relevant since our goal is performing plane detection of datasets of any kind. This way, other point-cloud information will not be considered as clues to detect planes; only 3D points are used in our plane detection algorithm.

2.1.2 Acquisition and Registration Errors

Ideal plane detection algorithms are not sensitive to point-cloud acquisition or registration errors. Hereafter we explain some usual problems which point clouds may have by capturing or registering points incorrectly:

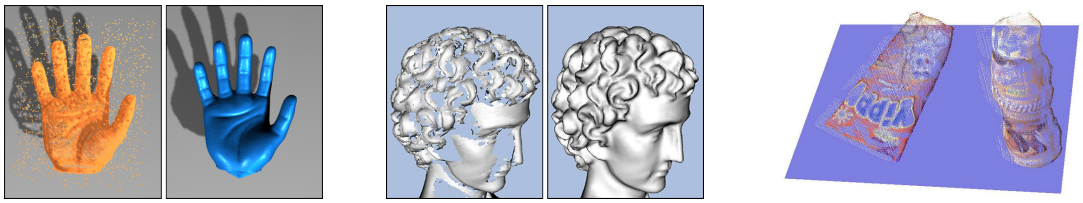
- **Noise** is an usual problem found in surface acquisition (Figure 2.1(a)). It can be classified as *high-order noise* or *low-order noise*. In laser scanners, for example, high-order noise usually appears when a scanner beam reaches the boundaries of objects. On these situations, part of the beam may touch the object while the other part goes on. The final calculation places the point at a wrong position making this an *outlier*. On the other hand, low-order noise is generated by low accuracy of scanners which fail on the precise sampling calculation.

- **Holes**: Once the surface properties hinder or restrict the scanners accessibility, certain regions may be lost, leading to incomplete reconstruction of objects and inserting holes into scanned models (Figure 2.1(b)). Holes can also be caused by specular or reflective surfaces, as well as by black surfaces (which do not reflect light). There are many algorithms concerned with filling holes of point clouds, since some methods cannot work with incomplete point models. We notice, however, that our algorithm works seamlessly in the presence of holes.

- **Overlapping samples**: When capturing all faces of an object, conventional scanners may register some faces from distinct positions. Thus, samples are acquired more than once from different perspectives. This may cause misalignment, i.e. when structures are shifted by each acquisition (Figure 2.1(c)). Later calibrations are necessary to overcome this problem, positioning samples on its right positions.

Plane detection algorithms can be implemented in such a way that they are not sensitive to these acquisition or registration errors. These problems always hamper the detection of shapes. How to treat them is out of the scope of this work. For further information on acquisition or registration error treatment, the reader may refer to (CHALMOVIANSKY; JUTTLER, 2003; MITRA; NGUYEN, 2003; LIU, 2006; WANG; OLIVEIRA, 2007).

Figure 2.1: Examples of registration errors. (a) Model of a hand with noise (left) and the model reconstructed by Kolluri et al.(right). (b) Model with holes of a head (left) and the model reconstructed by Lanman et al. (right). (c) Partially overlapped point cloud.



(a) (b) (c)

Kolluri; Shewchuck; O'Brien (2004), Lanman et al.(2006), Richtsfeld; Vincze (2009)

2.2 Hough Transform

The Hough transform (HT) (HOUGH, 1962; DUDA; HART, 1972) is a feature-detection technique. For any given input sample, it casts a vote for each instance of the feature one wants to detect that could possibly contain that sample. The votes are accumulated over all samples, and the detected features correspond to the ones with most votes. The time and space complexity of the algorithm both depend on the discretization used for the accumulator, whose dimensionality varies with the number of parameters used to describe the features to be detected. For instance, plane detection requires a 3D accumulator to represent the three parameters that characterize a plane.

2.2.1 Standard Hough Transform

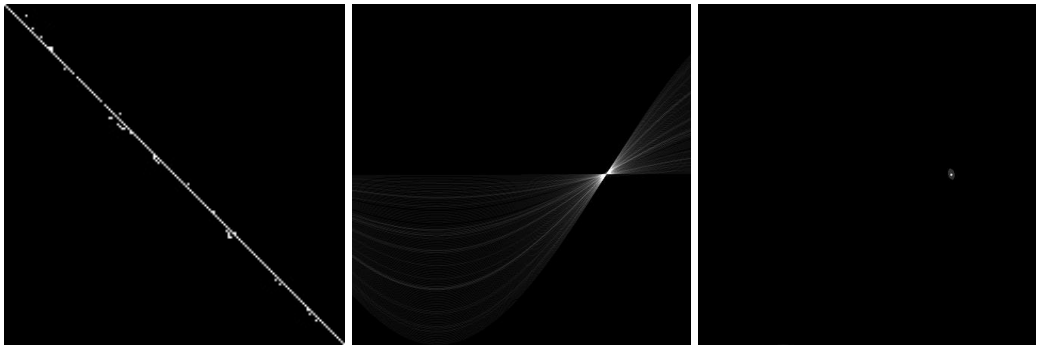
The Hough transform was introduced by Paul Hough (1962) for the detection of lines in images. Today, the universally used version of the HT is the *standard Hough transform* (SHT) proposed by Duda and Hart (1972), which replaced the slope-intercept with an angle-radius parameterization based on the normal equation of the line (Equation 2.1):

$$\rho = x\ cos(\theta) + y\ sin(\theta). \tag{2.1}$$

Here, x and y are the coordinates of a sample pixel, ρ is the distance from a line (passing through the pixel) to the origin of image's coordinate system, and θ is the angle between the normal of the line and the x-axis.

An important algorithm to the line detection field is the Kernel-based Hough transform (KHT) (FERNANDES; OLIVEIRA, 2008), which replaces the extensive voting procedure by an efficient voting scheme, casting votes only for close bins around the best-fitting lines thus producing a much cleaner voting map. For this, they cluster approximately collinear pixels and then they cast votes using an oriented elliptical-Gaussian kernel which models the uncertainty associated with the best-fitting line. For the input image in Figure 2.2 (a), (b) shows the resulting accumulator for the standard voting scheme compared to the KHT voting scheme (c).

Figure 2.2: For a given input image (a) the resulting accumulator for the standard voting (b) versus the Kernel-based Hough transform voting scheme (c). Note that the accumulator produced by the KHT voting is much cleaner than the accumulator produced by the standard voting scheme.



(a) (b) (c)

The Kernel-based Hough transform is able to achieve real-time performance even for relatively large images (approximately 2 MP), depending on the number of edge pixels. Unfortunately, its extension to the three-dimensional space is not straightforward since the subdivision procedure, which clusters the feature points, takes advantage of the pixel connection on an image.

The Standard Hough transform can be naturally extended to 3D by adding another slope parameter (Equation 2.2). Differently from lines, the 3D version of the SHT supports plane detection in the (θ, ϕ, ρ) Hough Space.

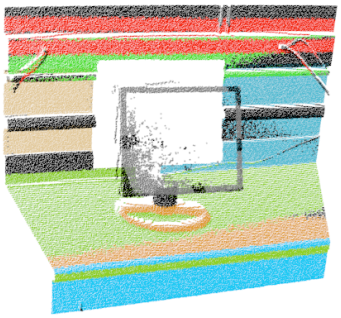
$$\rho = x\ cos(\theta)sin(\phi) + y\ sin(\phi)sin(\theta) + z\ cos(\phi). \tag{2.2}$$

In Equation 2.2, x, y and z are the Cartesian coordinates of the samples, $\theta \in [0°, 360°)$ and $\phi \in [0°, 180°]$ are the polar coordinates of the plane's normal vector, and $\rho \in \mathbb{R}_{\geq 0}$ is the distance from the plane to the origin of the coordinate system.

The Standard Hough transform for plane detection uses Equation 2.2 and iterates over each sample in the point cloud casting votes in the accumulator for all possible planes passing through that sample. More specifically, for given x, y and z coordinates, it iterates over all combinations of θ and ϕ, computing the value of the parameter ρ (Equation 2.2) and then casting a vote at the corresponding accumulator cell (or bin). To make the computation feasible, one needs to discretize the θ and ϕ parameter values (defining angular steps). Thus, the computational cost of the SHT is $O(|P|N_\theta N_\phi)$, where $|P|$ is the number of points in the point cloud P, and N_θ and N_ϕ are the number of bins in the discretization of the θ and ϕ angles, respectively. In the Figure 2.3 we show a plane-detection example in the *Computer* dataset with $68,852$ samples generated by SHT. The meaning of each parameter used in the following images is elucidated in (BORRMANN; DORIA, 2011).

Given the high computational cost of the SHT, many techniques have been proposed to accelerate its voting procedure. Common to most of these techniques is the focus on reducing the execution time by using a subset of the points in P, as opposed to designing new algorithms that effectively reduce the asymptotic cost of the voting process. The following part of this section reviews these techniques and shows examples of plane detection in the same point cloud (*Computer*). Note that all the following techniques of

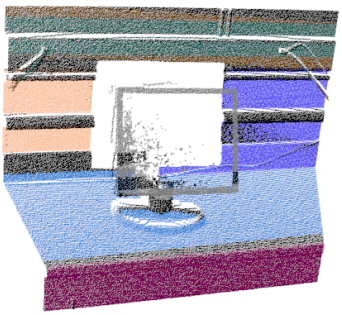
Figure 2.3: Example of plane detection generated by the Standard Hough transform technique with randomly colored planes. The technique detected 11 planes in approximately one minute. Parameters used in this example: *RhoNum* $= 100$, *ThetaNum* $= 60$, *PhiNum* $= 30$ and *PlaneRatio* $= 0.5$.



this Section are highly parameter depended. The detection may be compromised if one criterion is poorly estimated.

The *Probabilistic Hough transform* (PHT) (KIRYATI; ELDAR; BRUCKSTEIN, 1991) randomly selects m points ($m < |P|$) and uses them, instead of the entire point cloud, for voting. Since m is a percentage of $|P|$, the asymptotic cost is still $O(|P|N_\theta N_\phi)$. The PHT needs to find an optimal value for m to achieve good results. Small values tend to cause some relevant planes not to be detected, while large values do not result in significant reduction in execution time. As opposed to the SHT, the PHT is not deterministic. An example showing the detection of planar structures generated by this approach can be seen in Figure 2.4.

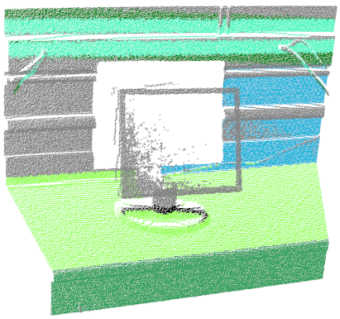
Figure 2.4: Example of plane detection generated by the Probabilistic Hough transform technique with randomly colored planes. The technique detected 9 planes in approximately 0.3 seconds. Parameters used in this example: *RhoNum* $= 100$, *ThetaNum* $= 60$, *PhiNum* $= 30$, *PlaneRatio* $= 0.5$ and *MinSizeAllPoints* $= 5$.



Finding the optimal value for m is not a simple task, as it depends on many characteristics of the point cloud. To overcome this difficulty, the *Adaptive Probabilistic Hough*

transform (APHT) (YLä-JääSKI; KIRYATI, 1994) monitors the accumulator during the voting procedure. As stable structures emerge, they are stored in a list of potential maximum cells and only this list needs to be monitored. Since the process is adaptive, there is no need for an initial m value. The algorithm ends when the list of potential peaks becomes stable (*i.e.*, when it keeps all largest peaks after an update phase). APHT is sensitive to noise, as the choice of the points is probabilistic and may lead to the detection of spurious planes, not present in the dataset. Its asymptotic cost is the same as SHT's. Figure 2.5 shows an example of plane detection generated by APHT.

Figure 2.5: Example of plane detection generated by the Adaptive Probabilistic Hough transform technique with randomly colored planes. The technique detected 6 planes in approximately one second. Parameters used in this example: *RhoNum* $= 100$, *ThetaNum* $= 60$, *PhiNum* $= 30$, *MinSizeAllPoints* $= 5$, *MinPlanarity* $= 0.3$, *PointDist* $= 0.05$ and *MaxPointPlaneDist* $= 0.02$.



The *Progressive Probabilistic Hough transform* (PPHT) (MATAS; GALAMBOS; KITTLER, 1998) tries to avoid the influence of random noise by only detecting structures whose number of votes exceeds a threshold defined as a percentage of the total number of votes. Once a structure has been detected, the votes from all samples that support it are removed from the accumulator. This is done to filter the accumulation that results from random noise. Thereafter, the points which lie on the shape are removed from the point set and the process restart for the remaining points. Like the previous techniques, PPHT is non-deterministic and its asymptotic cost is the same as SHT's. An example of plane detection generated by PPHT can be seen in Figure 2.6.

The Randomized Hough transform (RHT) (XU; OJA; KULTANEN, 1990) reduces the SHT's vote-processing time by exploiting the fact that a plane can be defined by three non-collinear points. The technique randomly selects groups of three non-collinear points and casts a single vote to the accumulator cell corresponding to the plane. This strategy drastically reduces the voting cost, making it $O(|P|)$. Unfortunately, the technique is non-deterministic and does not scale well with the size of the point cloud (due to the relatively large hidden asymptotic constant). Among all previous HT-based techniques for plane detection, RHT is by far the fastest one, being the universally used version to deal with tridimensional data considering other HT variants.

The accumulator is a decisive component of the Hough-transform methods, because it will be used for storing the votes. In order to detect planes in three-dimensional space it is necessary to use a special accumulator which better describes it, independently of the

Figure 2.6: Example of plane detection generated by the Progressive Probabilistic Hough transform technique with randomly colored planes. The technique detected 10 planes in approximately 30 seconds. Parameters used in this example: *RhoNum* = 100, *ThetaNum* = 60, *PhiNum* = 30, *MinSizeAllPoints* = 5, *MinPlanarity* = 0.3, *PointDist* = 0.05 and *MaxPointPlaneDist* = 0.02.

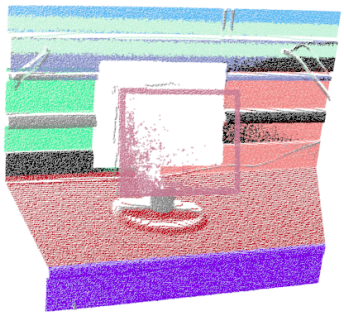
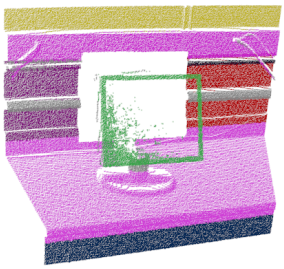


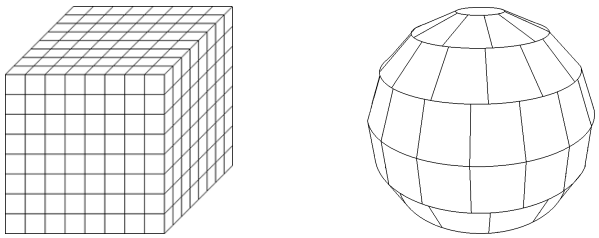
Figure 2.7: Example of plane detection generated by the Randomized Hough transform technique. The technique detected 7 planes in 0.121 seconds. Parameters used in this example: *RhoNum* = 100, *ThetaNum* = 60, *PhiNum* = 30, *MaxDist* = 0.1, *MinDist* = 0.01 and *AccumulatorMax* = 100



particularly Hough-based approach used. Different types of accumulators were tested by Borrmann et al. (2011) who have shown the advantages of using a spherical accumulator, called *accumulator ball*, with the purpose of having the same patch size for each cell. This allows each cell to have the same portion size of the space, decreasing the total number of cells that a regular 3D accumulator has while still using the same discretization. A comparison between a simple 3D accumulator (cube) and the accumulator proposed by Borrmann et al. (ball) can be seen in Figure 2.8. The accumulator ball also prevents the detection of spurious planes, helps in finding local maxima (peaks) and has a small storage space.

A detailed comparison between the techniques explained above (SHT, PHT, APHT, PPHT, RHT) and different types of accumulators can be found at (BORRMANN et al., 2011). In the following part of this Section we will focus in the remaining Hough transform variants.

Figure 2.8: A comparison between the *accumulator cube* (left) and *accumulator ball* (right) proposed by Borrmann et al. (2011). Note that both accumulators have the same discretization in ϕ, i.e., 8 cells over 180 degrees from the north pole to the south pole (the angle was sampled every 22.5 degrees). However, the *accumulator ball* has fewer cells and each one represents a different planar angle, which does not happen in the simple 3D accumulator. While the *accumulator cube* has 64 cells in the top, representing a single angle a plane can take ($\phi = 0$), the *accumulator ball* has only one.



2.2.2 Other Hough Transform Variants

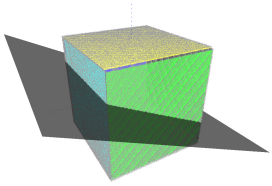
Vosselman et al. (2004) proposed a two-step procedure for the Hough transform, exploiting the connectivity of point clouds acquired with laser scanners to calculate the normal vectors of the points. This way, each sample casts a single vote. This approach is not as fast as the RTH. Moreover, it is not applicable to unorganized point clouds. Bauer and Polthier use the Radon transform (continuous form of the HT) to detect planes on a structured or unstructured grid. They use a subdivided icosahedron, using a Hamiltonian path over the edge graph, to represent the parameter space in order to search for all connected components and to compute their respective masses. This technique requires the use of a grid and its performance is similar to the SHT. More recently, Ogundana et al. (2011) used an optimized model for tridimensional sparse matrix to accumulate votes. Furthermore they propose a robust peak detection algorithm using connected component labeling and weighted average. Ogundana et al. also showed a Hough transform optimization to detect parallel planes, replacing the default accumulator by an one-dimensional array, since the planes have the same orientation. Nguyen et al. (NGUYEN et al., 2013) estimate normal vectors in range images (based on their neighbors) and map such vectors to a sphere (a Gauss map) to define plane orientations. Optimization is then used to segment patches of coplanar samples in the range image. The authors demonstrated their technique on simple box-like and polyhedral shapes.

2.3 Random Sample Consensus

Another important class of algorithms for performing plane detection is the Random Sample Consensus (RANSAC) (FISCHLER; BOLLES, 1981). RANSAC is a widely used technique, being reliable even in the presence of a high proportion of outliers. It can also be generalizable to solve other problems, such as cylinder detection in unorganized point clouds (CHAPERON; GOULETTE, 2001), ellipse detection for calibrating central catadioptric cameras (DUAN; WANG; GUO, 2010), and to estimate complex curves (*e.g.*, splines). RANSAC performs plane detection by randomly choosing three points, calculating the plane defined by them, and counting how many points (in the point cloud) lie on this plane within a tolerance threshold. The number of points found is called the *score*

of the plane. The algorithm stops when it reaches stability, based on a low probability of finding a plane with higher score than the previous ones. RANSAC's computational cost for detecting a single plane is then given by $(I\,(E + |P|\,F)) = O(I\,|P|)$, where I is the number of iterations required to detect a plane, E is the cost of estimating a plane from three points, and F is the cost of checking whether a point lies on a plane. While being robust to noise, RANSAC's random nature makes it non-deterministic. Depending on the choice of its parameter values, the algorithm may detect planes that do not represent the original dataset. This is illustrated in Figure 2.9 for a point cloud representing the faces of a cube. The gray plane was detected by RANSAC and represents one of many planes that could be detected by the algorithm.

Figure 2.9: A point cloud representing the faces of a cube, shown in color. Depending on the used parameters, RANSAC may detected (spurious) planes, such as the one shown in gray, which do not represent the original dataset.



RANSAC must optimize three main parameters to produce a good output: the threshold value (t_r) for determining when a subset fits a model, the number of iterations (k_r) performed by the algorithm and the minimum number of expected data required to fit a model (e_r). Hereafter we present a deeper explanation about these three parameters.

The **distance threshold** is the maximum orthogonal distance which a point will be considered an inlier to the model. This parameter is usually chosen empirically. If the error distribution is known, the distance threshold can be optimized to select inliers with high probability. Low threshold values will yield an insufficient number of inliers for the best-fit, while high threshold values may select incorrect points that are outside the best-fit, hindering the quality of the plane detection. The problem generated by misestimating distance thresholds in a real example can be seen in Figure 2.10.

The number of iterations are usually not constant and not specified directly. They depend on the probability p_r that the algorithm will produce a useful result in its next iteration. Let $0 \leq w_r \leq 1$ be the inlier point probability, which normally assume conservative values depending on the model being fitted. For a subset of n points, the probability that all points are inliers is $(w_r)^n$, while $1 - (w_r)^n$ is the probability that at least one of the n points is an outlier. For all iterations, the probability that the algorithm selects all sets with at least one outlier is $1 - p_r$. This way,

$$1 - p_r = (1 - (w_r)^n)^{k_r}. \tag{2.3}$$

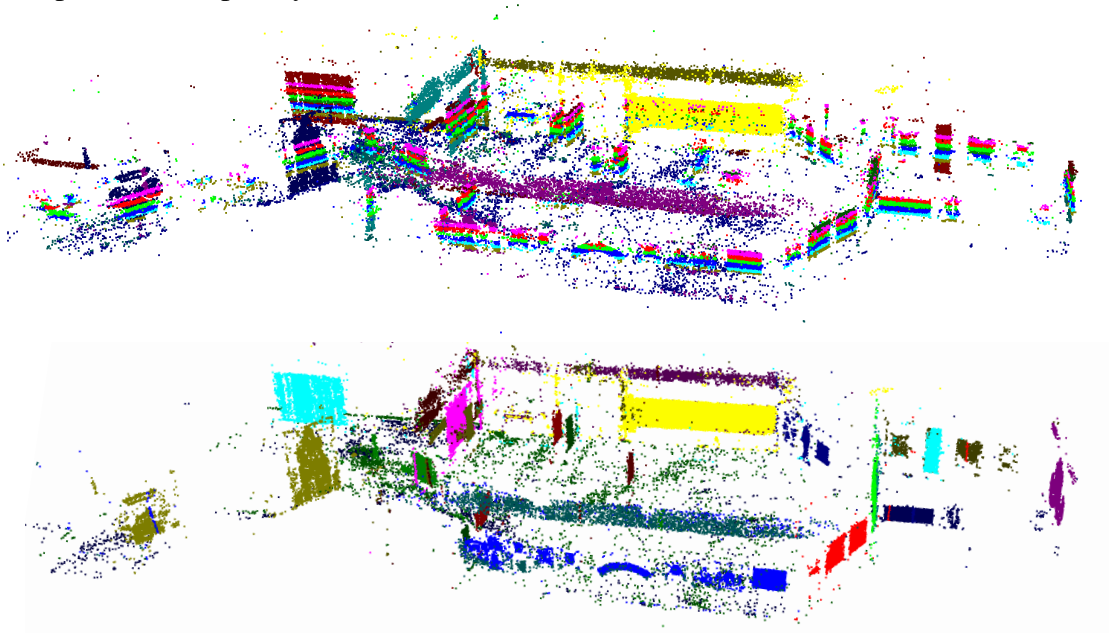
Isolating the number of iterations k_r we have

$$k_r = \frac{\log(1 - p_r)}{\log(1 - (w_r)^n)} \tag{2.4}$$

which give us the number of iterations that the algorithm must run to find a good solution. The probability p_r is usually chosen to be 0.99.

The minimum expected number of points defines how many inliers the algorithm needs to define a model. A good approximation to this parameter is given by multiplying

Figure 2.10: Real example of the threshold problem in RANSAC. High threshold ($t_r = 0.06$) (top) - many solutions which cross the entire model are considered good ones. Optimized threshold ($t_r = 0.02$) (bottom). Note that the thresholds are very close and produce completely different results.



the total number of points on the data by the inlier point probability w_r. A plane-detection example generated by RANSAC can be seen in Figure 2.11.

Schnabel et al. (2007) introduced an optimization to RANSAC using an octree to establish spatial proximity among samples. In their approach, point selection is based on spatial proximity, and the score function only tests a local subset of the samples. Since spatial proximity does not guarantee coplanarity, the technique needs to estimate normal vectors for samples, and the shapes have to be properly sampled. While this approach can significantly accelerate RANSAC, it inherits RANSAC's limitations, and the performance reported by the authors is still far from real time for datasets containing a few hundred thousand samples.

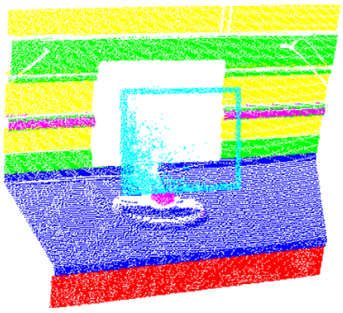
2.4 Surface Growing

The third class of techniques used to identify planes in point clouds is surface growing (FISCHLER; BOLLES, 1986; BESL; JAIN, 1988; CHEN, 1989; POPPINGA et al., 2008) – the 3D analogue of region growing in images. These techniques perform a local search to identify and expand regions with the same range of characteristics. Surface growing methods require information about the neighbors of each sample, not being directly applicable to unorganized point clouds, which lack explicit connectivity information.

The growing procedure can be based on one or more of these criteria:

- **Proximity of points**: To evaluate the possibility of a point be part of the growing region, it must be near other point already on the region. The proximity is calculated depending on the point cloud characteristics, such as bounding box, number of points and level of noise.

Figure 2.11: Example of plane detection generated by RANSAC. It took approximately 0.4 seconds to detect 7 planes. The parameters used in this example were: $t_r = 0.02$, $p_r = 0.99$ and $e_r = 2000$.



- **Global parameters**: The global parameters criteria depend on the distance from the growing region's fitted surface to the new candidate point. The surface must be fitted without the new point, while maintaining the its orthogonal distance from the surface below a specified threshold.

- **Local characteristics**: The local characteristics criteria are related with the local surface normal. Both the surface normal and the local surface normal must be similar, in order to add the evaluated point to the growing region.

Once the chosen criteria are satisfied, the region continues growing until it cannot expand anymore. Since it expands surfaces' growing areas locally, Surface-Growing-based techniques may be very time consuming when considering big (10^4 samples) to huge (10^6 samples) point clouds.

Recently, Deschaud and Goulette (2010) proposed an algorithm to detect planes in unorganized point clouds using filtered normals and voxel growing. Their approach assigns a normal vector to each point through an improved normal-estimation procedure and uses a voxel-growing algorithm based on these normal vectors. It is, however, considerably slower than existing techniques.

2.5 Tensor Voting

Tensor voting (TV) is a framework which retrieves, at the same time, all salient structures from a dataset (MEDIONI; LEE; TANG, 2000). It is founded on two components: *tensor calculus* for representation, and *non-linear voting* for data communication. TV can detect structures at any dimensionality, while still being robust against noise and preserving discontinuities as the HT. However, since tensor voting is naturally multidimensional, it cannot be directly applied to the detection of predefined types of data in an efficient way.

2.6 Linear Subspace Learning

The purpose of linear subspace learning (LSL) algorithms is to find the possible linear or affine subspaces which accommodate as many objects from a dataset as possible. These include, among others, techniques based on principal component analysis (PCA) (VIDAL; MA; SASTRY, 2003), linear discriminant analysis (LDA) (MCLACHLAN, 2004), general averaged divergence analysis (GADA) (TAO et al., 2007), and locality preserving projections (LPP) (HE; NIYOGI, 2004). A more complete review about LSL algorithms can be found in (VIDAL, 2011). In contrast to dimensionality reduction techniques, which can solve a class of more generic problems, our aim is to find planes in the three-dimensional space in an efficient manner.

2.7 Summary

In this Chapter we discussed the most relevant research related to our work. We pointed that even though registration errors can hamper the detection of planes, the computational cost to correct this is too high, so it is not taken into account.

Furthermore, we reviewed the five most important plane detection approaches: Hough transform, which perform a voting procedure on possible planes to decide the most fitting; random sample consensus, which search for larger amounts of coplanar points with a certain degree of error; surface growing, which start from a seed and grow to regions with similar characteristics; tensor voting, which detects salient structures in multidimensional data; and linear subspace learning, which reduces the dimensionality of the data to find possible solutions. HT-based techniques have received special attention as they are mostly related to this work.

Generally speaking, the referenced papers reflect the state of the art in point-cloud plane detection. Since most reported approaches are very dependent on parameters, and/or slow for large datasets, the main motivation for this work is to improve on existing techniques for plane detection to provide an efficient and easy-to-use solution. In this way, by greatly reducing the needed computational costs we make possible a new range of applications that require real-time execution.

This work proposes detection of planes in unorganized point clouds combining and enhancing some previous ideas. The main inspiration of this work is the Kernel-based Hough transform of Fernandes and Oliveira (2008), which performs real-time line detection in images using a fast voting scheme. Another important concept used was the accumulator ball, proposed by Borrmann et al. (2011) which improves the three-dimensional perception of space, and thus improving the quality of results. Both the voting scheme and the accumulator design were modified to better suit our purposes. The most significant differences from the previous works are:

- An optimized way to handle the access of neighboring cells in the accumulator, used to cast votes around the best-fitting plane. In (BORRMANN et al., 2011) they are not worried about accessing the neighborhood of a cell, since only one vote is cast per plane.

- A new subdivision strategy that extends the one used by (FERNANDES; OLIVEIRA, 2008) for three-dimensional space, while being efficient for large datasets and still producing good results. This adaptation is not straightforward, as it is harder to deal with neighborhood in the three dimensional space than in the two dimensional space.

- Real-time performance for datasets up to 10^5 points. None of the reported approaches are able to detect planes in real-time in datasets of this magnitude.
- A deterministic approach that can generate consistent result in dynamic data. Most current approaches are probabilistic, producing different outputs event for the same data.
- A new approach for storing votes in the accumulator using floating points numbers instead of integers, that is more accurate and robust.
- An $O(n \log n)$ approach in number of samples which does not require connectivity information nor information about sample normals.

3 EFFICIENT PLANE DETECTION IN POINT CLOUDS

This chapter presents the details of our solution for detecting planes in point clouds. It discusses the faced challenges and how we solved them. We will first present an overview of the entire process and after this we will explain each part in details. At the same time, we focus on how to implement some algorithms, needed for better understanding.

3.1 Overview

The main contribution of this work is provide an algorithm capable of detecting planes in real-time for sufficiently large point clouds (10^5). It has been inspired by the efficient Kernel-based Hough transform (KHT) for line detection introduced by Fernandes and Oliveira (2008). However, when dealing with unorganized point clouds, the lack of explicit neighborhood information among samples (which is available for images) requires new and efficient clustering and accumulation-management strategies. In a nutshell, our technique performs a fast and robust octree-based segmentation of approximately coplanar clusters of samples. We then use the identified clusters to perform a Hough-transform voting procedure where votes are cast by clusters (as opposed to by individual samples) on a spherical accumulator. For voting, we use a trivariate Gaussian distribution (kernel) defined over spherical coordinates (θ, ϕ, ρ) and centered at each cluster's best fitting plane. Peak detection is then performed on the resulting accumulator identifying peaks of votes, according to their importance, which better represent the point-cloud planes. Thus, we choose to organize this chapter into each part of this process: *Clustering*, *Voting* and *Peak Detection*. Algorithm 1 summarizes the technique.

Algorithm 1 Plane Detection (3-D KHT) Algorithm

Require: P {point cloud}

1: $nodes \leftarrow Clustering(P)$ {*cluster approximately coplanar samples* }
2: **for** each n in $nodes$ **do**
3: $q \leftarrow$ kernel(n) {*kernel estimation*}
4: $accumulator \leftarrow accumulator + voting(q)$ {*voting*}
5: **end for**{*peak detection*}
6: sort $accumulator\ cells$ by voting importance in descending order
7: iterate over $accumulator$ detecting cells not adjacent to already inspected ones {*peak detection*}

Our goal is to be able to process any kind of point cloud with the same assurance. The optimizations proposed in this thesis for the three-dimensional HT allow a software implementation to operates in real-time for sufficiently large point clouds (10^5) of any

kind. The time performance of our method is smaller to the ones obtained by others techniques, while producing results at least as good as theirs. In the next sections we explain how this is done.

3.2 Clustering of Approximately Coplanar Samples

Clustering of approximately coplanar samples is key to our technique as it optimizes the voting procedure, which is the Hough transform's bottleneck. For unorganized point clouds, no neighborhood information among samples is available. For efficiency, we perform clustering by spatial subdivision. For this, we have compared the advantages of using kd-trees and octrees. Kd-trees provide little control over the dimensions of the nodes. According to our experience, subdividing the kd-tree cells using the centroid of the samples tends to lead to thin cells that do not capture the shapes of the planes in the dataset. In contrast, all nodes at a given level of an octree have $1/8$ of the size of its parent node and better capture the structure of the planes. Moreover, the costs of creating and manipulating a kd-tree are higher than for an octree. For these reasons, we have chosen an octree as spatial-subdivision data structure. It has proven to be a good choice both in terms of efficiency and quality of the results.

The clustering procedure starts with a root node that includes the entire point cloud, which is then recursively subdivided to refine the octree. Except when the entire point cloud is just a plane, searching for planes in the initial level(s) of the octree often lends to less effective computations.

Thus, the procedure only checks for approximate coplanarity among samples after a certain level of the octree has been reached, thus minimizing processing time. According to our experience, starting checking for approximately sample coplanarity around level 4 of the octree provides a good compromise between computational performance and robustness, and produces good segmentation in practice. The more detailed the point cloud, the more subdivisions are required, as nodes must be small enough to contain only approximately coplanar samples. If no further subdivision is required for an octree node, that branch stops and the node is stored as a cluster. If, on the other hand, the number of samples inside the octree node is smaller than a threshold, the node is marked as not containing a cluster.

The procedure for clustering approximately coplanar samples is presented in Algorithm 2. It uses descriptive statistics to analyze the data and calculate the variances associated with the point-cloud distribution. For this, principal component analysis (PCA) is used. Since the eigenvalues of the covariance matrix associated to the set of samples inside an octree node represent the proportions of the variances of the sample distribution inside that cell, they can be used to filter out clusters that could not represent planes. In order to check whether a set of samples is approximately coplanar, two conditions are verified: the cluster *thickness* and its degree of *isotropy* (the technique should avoid detecting lines and thin elongated clusters as planes). Thus, let Λ and V represent, respectively, the eigenvalues and the eigenvectors of a cluster's covariance matrix Σ. These eigenvalues are non-negative and we sort them in ascending order so that $\lambda_i \leq \lambda_{i+1}$. A test for approximate sample coplanarity can be obtained by checking if $(\lambda_2 > s_\alpha \lambda_1)$ and $(\lambda_3 < s_\beta \lambda_2)$, where s_α and s_β are scaling factors defining relative tolerances for the acceptable amount off-plane displacement (*i.e.*, noise) and degree of sample anisotropy on the cluster. According to our experience, $s_\alpha = 25$ and $s_\beta = 6$ produce good results and have been used for all examples shown in the paper. The recursive subdivision performed by Algorithm 2

stops when the current octree cell is considered to contain approximate coplanar samples (*i.e.*, $n_{coplanar} = true$, line 9) or the cell contains less than a minimum number of samples (s_{ms}, in line 2). Small number of samples tend to provide less reliable estimates for the variances of the samples. In our experience, $s_{ms} = 30$ provides a good threshold for large point clouds.

Algorithm 2 Clustering

Require: n {current node of the octree }
s {settings to cluster data: s_{mp}, s_{level}, s_{α}, s_{β} }

Symbols

$n_{samples}$ {samples in the current octree node }
n_{level} {level of the current octree node}
$n_{coplanar}$ {are the samples in current node approximately coplanar?}
$n_{children}$ {children of the current octree node}
s_{ms} {minimum number of samples required in a cluster}
s_{level} {first octree level for checking for approximate coplanarity}
s_{α} {relative tolerance associated with plane thickness }
s_{β} {relative tolerance associated with plane isotropy }

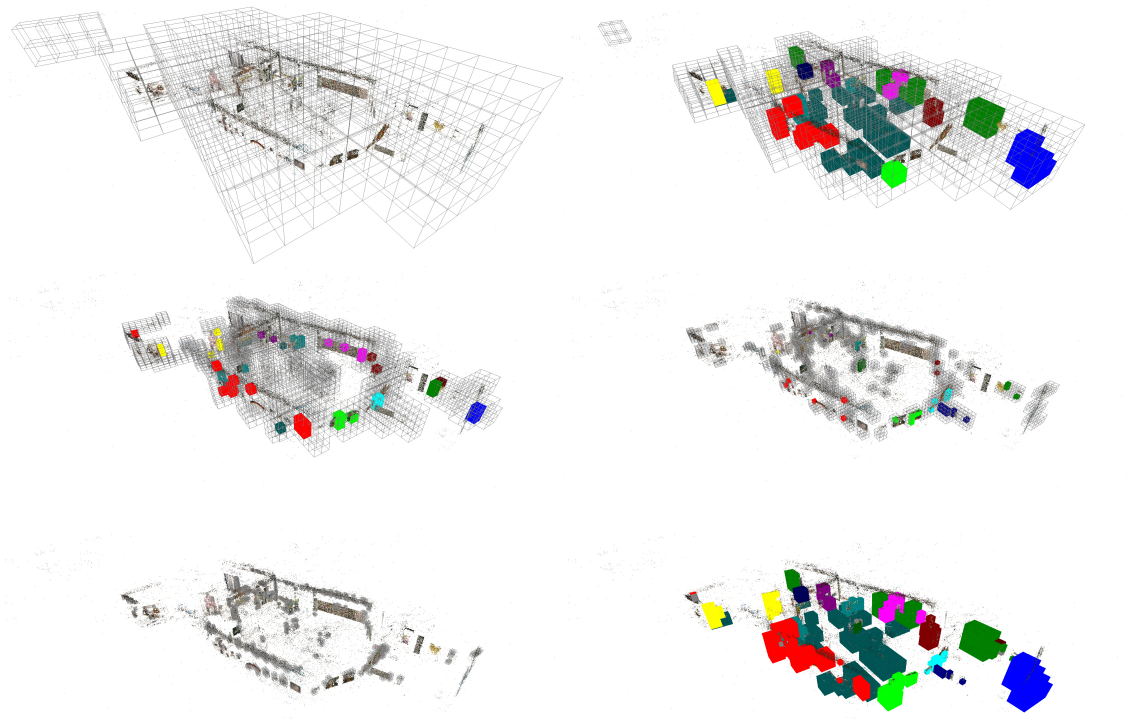
1: $n_{coplanar} \leftarrow false$
2: **if** $size(n_{samples}) < s_{ms}$ **then**
3: **return**
4: **end if**{octree subdivision step}
5: **if** $n_{level} > s_{level}$ **then**
6: $\Sigma_{(x,y,z)} \leftarrow cov(n_{samples})$ {covariance matrix in (x, y, z) space}
7: $(V_{xyz}, \Lambda_{xyz}) \leftarrow eigen(\Sigma_{(x,y,z)})$ {eigen-decomposition}
{approximate coplanarity test }
8: **if** $(\lambda_2 > s_{\alpha}\lambda_1)$ and $(\lambda_3 < s_{\beta}\lambda_2)$ **then**
9: $n_{coplanar} \leftarrow true$
10: **return**
11: **end if**
12: **end if**
13: $n_{children} \leftarrow children(n)$ {initialize the node's eight children}
14: **for** each p in $n_{samples}$ **do**
15: put p in respective child node
16: **end for**
17: **for** each c in $n_{children}$ **do**
18: call Clustering(c,s_{mp}, s_{level}, s_{α}, s_{β}) {recursive call for node c}
19: **end for**

Once an octree cell is considered to contain an approximately coplanar sample cluster, least-squares is used for plane fitting (SHAKARJI, 1998) after discarding samples at a distance bigger than $\tau/10$ from the plane passing by the centroid of the cluster and whose normal is given by the eigenvector with smallest eigenvalue of Σ. τ is the current octree-node edge length.

Figure 3.1 illustrates the adaptive octree refinement and sample clustering for the Museum dataset using Algorithm 2. From top to bottom, left to right, the first five images show the 6th, 7th, 8th, 9th, and 10th levels of the octree. The image at the bottom right shows the octree nodes containing approximately coplanar samples. Note that these nodes might be at different levels of the octree. A different color has been assigned to each plane,

even when they span different levels of the octree. This is possible by keeping track of the clusters who voted for the individual planes, and will be explained next.

Figure 3.1: Adaptive octree refinement and sample clustering for the Museum dataset using Algorithm 2. From top to bottom, left to right, the first five images show the 6th, 7th, 8th, 9th, and 10th levels of the octree. The image at the bottom right shows all nodes at different octree levels containing coplanar samples. Note that once a planar patch is found the subdivision stops for that branch. Each color represents one plane, whose reconstructions are shown in Figure 1.1.



3.3 Computing Gaussian Trivariate Kernels for Cluster Voting

Let a cluster of approximately coplanar samples stored in an octree node, with covariant matrix Σ, and centroid $\mu = (\mu_x, \mu_y, \mu_z)^T$ (Figure 3.2). Also let $V = \{\vec{v_1}, \vec{v_2}, \vec{v_3}\}$ be the unit eigenvectors of Σ and let $\Lambda = \{\lambda_1, \lambda_2, \lambda_3\}$ be their respective eigenvalues, so that $\lambda_i \leq \lambda_{i+1}$. The equation of the plane π passing though μ and with normal $\vec{n} = \vec{v_1} = (n_x, n_y, n_z)^T$ is given by:

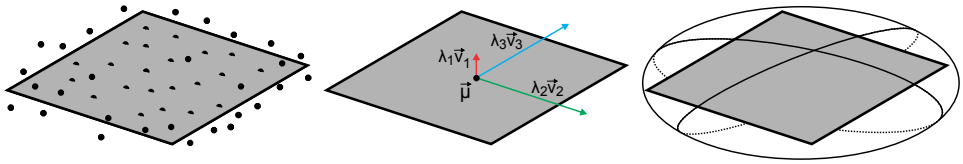
$$Ax + By + Cz + D = n_x x + n_y y + n_z z - (n_x \mu_x + n_y \mu_y + n_z \mu_z) = 0 \tag{3.1}$$

Using Equation 2.2, one can rewrite Equation 3.1 using spherical coordinates as:

$$\begin{aligned} \rho &= -D = \mu_x\, n_x + \mu_y\, n_y + \mu_z\, n_z = \sqrt{p_x^2 + p_y^2 + p_z^2}, \\ \theta &= \arctan 2(p_y, p_x), \qquad \phi = \arccos\left(\frac{p_z}{\rho}\right), \end{aligned} \tag{3.2}$$

$\rho \in \mathbb{R}_{\geq 0}$, $\theta \in [0°, 360°)$, $\phi \in [0°, 180°]$ and $\vec{p} = (p_x, p_y, p_z)^T = \rho\vec{n}$. For θ calculation, if the angle between $\vec{n}$ and $\vec{\mu}$ is bigger than $90°$, we point $\vec{n}$ to the opposite direction (*i.e.*, multiply it by -1). When voting in an accumulator indexed by (θ, ϕ, ρ), the vote distribution is based on the uncertainties associated to each cluster's best-fitting plane π (*i.e.*, the cluster's variances σ_ϕ^2, σ_θ^2, and σ_ρ^2). A cluster with small variances concentrates its votes in a small region of the accumulator, while a cluster with large variances spreads its votes over a large region, like in the KHT (FERNANDES; OLIVEIRA, 2008).

Figure 3.2: Samples approximating a planar region in a point cloud, shown with its best-fitting plane (left). Eigenvectors of the covariance matrix Σ associated to the sample distribution (center). Ellipsoid defined by the eigenvalues and eigenvectors of Σ (right).



Algorithm 3 Computing $\Sigma_{(\theta,\phi,\rho)}$ and the Gaussian kernel voting threshold

Require: $\Sigma_{(x,y,z)}$ {covariance matrix with respect to x, y and z coordinates}
1: $J \leftarrow Jacobian()$ {defined in Equation 3.4}
2: $\Sigma_{(\theta,\phi,\rho)} \leftarrow J\,\Sigma_{(x,y,z)}\,J^T$ {cov. matrix in (θ, σ, ρ) space from $\Sigma_{(x,y,z)}$}
3: $\sigma_\rho^2 \leftarrow \sigma_\rho^2 + \varepsilon$ {add a small value to avoid zero variance}
4: $\sigma_\theta^2 \leftarrow \sigma_\theta^2$
5: $\sigma_\phi^2 \leftarrow \sigma_\phi^2$
6: $(V_{\theta\phi\rho}, \Lambda_{\theta\phi\rho}) \leftarrow eigen(\Sigma_{(\theta,\phi,\rho)})$ {eigen-decomposition of $\Sigma_{(\theta,\phi,\rho)}$}
7: $\lambda_{\theta\phi\rho\_min} \leftarrow smallestEigenvalueIn(\Lambda_{\theta\phi\rho})$ {smallest eigenvalue}
8: $V_{\theta\phi\rho\_min} \leftarrow Eigenvector(\lambda_{\theta\phi\rho\_min})$ {eigenvector corresp. to $\lambda_{\theta\phi\rho\_min}$}
9: $std\_dev \leftarrow sqrt(\lambda_{\theta\phi\rho\_min})$ {standard deviation}
10: $g_{min} = Gaussian(V_{\theta\phi\rho\_min}\, 2\, std\_dev)$ {threshold value for voting}

The variances and covariances defined in the (θ, ϕ, ρ) space can be estimated from the covariance matrix $\Sigma_{(x,y,z)}$ defined in Euclidean space, using first-order uncertainty propagation analysis (PARRATT, 1961) as:

$$\Sigma_{(\theta,\phi,\rho)} = \begin{pmatrix} \sigma_\rho^2 & \sigma_{\rho\phi} & \sigma_{\rho\theta} \\ \sigma_{\rho\phi} & \sigma_\phi^2 & \sigma_{\phi\theta} \\ \sigma_{\rho\theta} & \sigma_{\phi\theta} & \sigma_\theta^2 \end{pmatrix} = J\Sigma_{(x,y,z)}J^T = J \begin{pmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_z^2 \end{pmatrix} J^T, \tag{3.3}$$

where J is the Jacobian matrix:

$$J = \begin{pmatrix} \frac{\partial\rho}{\partial p_x} & \frac{\partial\rho}{\partial p_y} & \frac{\partial\rho}{\partial p_z} \\ \frac{\partial\phi}{\partial p_x} & \frac{\partial\phi}{\partial p_y} & \frac{\partial\phi}{\partial p_z} \\ \frac{\partial\theta}{\partial p_x} & \frac{\partial\theta}{\partial p_y} & \frac{\partial\theta}{\partial p_z} \end{pmatrix} = \begin{pmatrix} n_x & n_y & n_z \\ \frac{p_x p_z}{\sqrt{w}\rho^2} & \frac{p_y p_z}{\sqrt{w}\rho^2} & -\frac{\sqrt{w}}{\rho^2} \\ \frac{-p_y}{w} & \frac{p_x}{w} & 0 \end{pmatrix}. \tag{3.4}$$

(p_x, p_y, p_z) are defined in (3.2), $\vec{n} = (n_x, n_y, n_z)^T$, and $w = p_x^2 + p_y^2$.

3.4 Cluster Voting using 3D Gaussian Distributions

Once we have computed the variances and covariances associated with θ, ϕ and ρ ($\Sigma_{(\theta,\phi,\rho)}$), the votes are cast in the spherical accumulator using a trivariate Gaussian distribution. For the multivariate non-degenerate case, *i.e.*, when the covariance matrix Σ is symmetric and positive definite, its probability density function is given by (TONG, 1990)

$$p(\mathbf{x}|\mu,\Sigma) = \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^t\Sigma^{-1}(\mathbf{x}-\mu)\right), \tag{3.5}$$

where $|\Sigma|$ is the determinant of Σ. Considering the trivariate case (*i.e.*, $k = 3$), letting $\vec{\delta} = \mathbf{x} - \mu$ be the displacement with respect to the center, and since the votes are already centered at the best-fitting parameters (θ, ϕ, ρ), this equation can be rewritten as

$$p(\vec{\delta},\Sigma) = \frac{1}{15.7496\,|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}\,\vec{\delta}^{\,t}\,\Sigma^{-1}\,\vec{\delta}\right). \tag{3.6}$$

Casting votes for a given accumulator cell requires two matrix-vector multiplications and one exponentiation, since the determinant of the covariance matrix and its inverse need to be calculated only once per cluster.

As planes become more horizontal (*i.e.*, when ϕ approaches 0 or 180 degrees) the variance relative to θ tend to become large, since at the poles the parameter θ does not affect the orientation of a plane. As a result, the amount of votes in individual accumulator cells near the poles tend to be smaller than in voted cells in other regions of the accumulator. Although this does not affect the correct detection of peaks around the poles, sorting the detected planes taking into account only the amount of votes would lead to always finding vertical planes before horizontal ones.

When estimating the importance of a cluster (and consequently the importance of its votes), one should consider other properties besides its number of samples. Aspects, such as area coverage should be given greater importance as sampling density varies with object distance to the scanner. Thus, we weight votes from a cluster by the relative size of its octree node with respect to the size of the octree root (in our system, all octree cells, including the root, are cubic cells). Votes are then weighted using

$$w_{cluster(i)} = w_a\left(\frac{node_{size}}{octree_{size}}\right) + w_d\left(\frac{|C_i|}{|P|}\right), \tag{3.7}$$

where $node_{size}$ and $octree_{size}$ are, respectively, the edge length of the octree node containing the cluster and the edge length of the root node. $|C_i|$ is the number of samples in the cluster i and $|P|$ is the total number of samples in the point cloud. w_a and w_d are the weights associated with *relative area* and *relative number of samples*, such that $w_a + w_d = 1$. According to our experience, spatial coverage should be favored over number of samples. While we have used $w_a = 0.75$ and $w_d = 0.25$ in all examples shown in the paper, we have also observed that $w_a = 1$ and $w_d = 0$ still produces good results in practice.

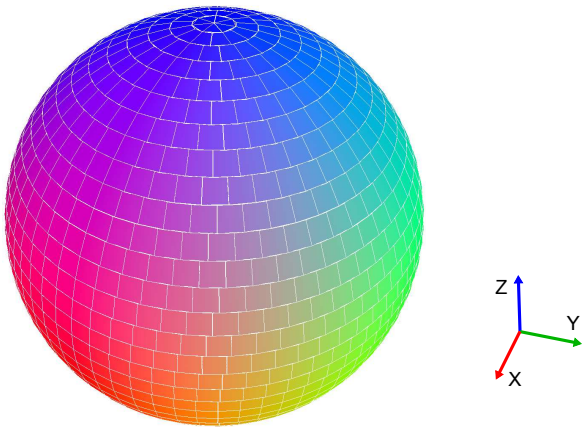
The voting procedure uses a spherical accumulator indexed by (θ, ϕ, ρ), which is described in Section 3.5. Starting at the center of the 3-D Gaussian kernel representing the position, orientation, and uncertainties of the best-fitting plane for a given cluster, the voting procedure iterates away from the kernel's center up to two standard deviations storing votes in the accumulator's cells. This provides a 95.4% assurance that the selected region of the parameter space receiving votes covers the true plane. Sampling is performed in

the accumulator at steps of $\Delta\theta$, $\Delta\phi$ and $\Delta\rho$. The number of votes that a cluster casts in a given accumulator cell a is obtained by multiplying the weight of the cluster's vote (Equation 3.7) by the evaluation of Equation 3.6 for the cell's $(\theta_a, \phi_a, \rho_a)$ parameter values (*i.e.*, for $\vec{\delta} = (\theta_a, \phi_a, \rho_a) - (\mu_\theta, \mu_\phi, \mu_\rho)$).

3.5 The Spherical Accumulator

An accumulator is a registry where intermediate results are stored, during the execution of an algorithm, to build an incremental solution. In this case, votes are stored in a 3-D accumulator to represent different combinations of θ, ϕ and ρ, where each combination (accumulator cell) represents a plane. While a 2-D array provides a good accumulator for the detection of lines in images, Borrmann et al. (2011) have demonstrated that the use of a 3-D accumulator array for plane detection may compromise the result of the HT. They have shown that since polar regions must have fewer cells than equatorial regions, the use of a full 3-D array may result in cells improperly receiving less votes than others. To overcome this problem, Borrmann et al. have proposed a spherical accumulator called the *accumulator ball*, whose illustration is provided in Figure 3.3.

Figure 3.3: A discrete representation of the 3-D spherical accumulator, showing the individual cells for a given value of the parameter ρ. The colors represent normal directions.



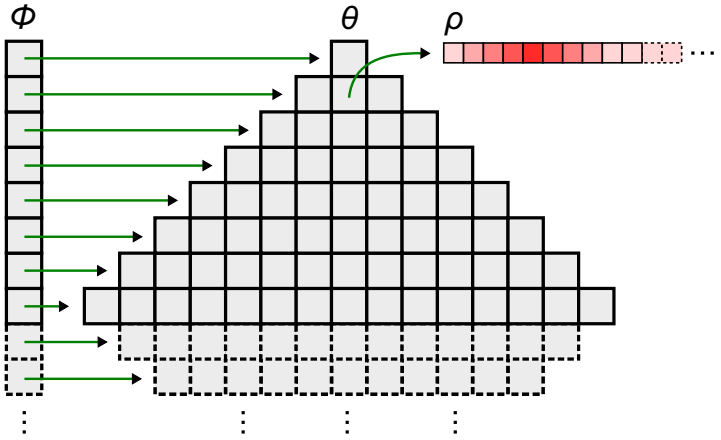
While Borrmann et al. cast votes in a conventional way (*i.e.*, they cast one vote for each possible plane passing through each sample in 3-D), we vote for each entire cluster at once. Thus, we also need to cast votes for cells adjacent to the one that represents the cluster's best-fitting plane. This is required to account for the uncertainty resulting from the variances in the cluster's sample positions. We use Borrmann et al.'s spherical accumulator, but normalize the azimuthal angle $\theta \in [0°, 360°)$ to $[0, 1)$, as its discretization varies with the elevation angle ϕ (see Figure 3.3). For any value of ϕ, the θ index for actually accessing the spherical accumulator is obtained as $\theta_{index} = round(\theta\, nc(\phi))$, where $nc(\phi)$ is the number of accumulator cells for the elevation angle ϕ.

For identifying adjacent cells, θ_{index} must be incremented/decremented using modular arithmetic. The ϕ_{index} must be between 0 and the size of the array (ϕ_{max}), which varies with the accumulator discretization. If incrementing/decrementing ϕ_{index} should exceed its lower or upper bounds, the sign of its increment is reversed to guarantee the wrapping around the sphere. Finally, ρ_{index} must be between 0 and ρ_{max}, which depends on the point cloud size. If ρ_{index} exceeds the limit of ρ_{max}, the voting process stops; if, however,

it assumes a negative value, θ_{index} and ϕ_{index} are recalculated for angles θ and ϕ in the opposite direction.

Votes in a HT 3-D accumulator tend to be sparsely distributed. Thus, during initialization, we only allocate space for the angular discretization of θ and ϕ. The third dimension (ρ) is allocated as needed during the voting procedure. Therefore, if a range of ρ values are only required around certain values of θ and ϕ, they will only be allocated there. This strategy lends to considerable memory savings, allowing such memory to be used towards better angular discretization, resulting in more accurate detections. Figure 3.4 illustrates the data structures used for implementing the spherical accumulator: a 1-D array with size ϕ_{max} represents the discretization of the elevation angle ϕ. Each element of this array contains a pointer to another 1-D array representing the discretization of the azimuthal angle (θ) at that specific elevation. In turn, each element of a θ array stores a pointer to yet another 1-D ρ array that stores the number of votes cast to cells indexed by (θ, ϕ, ρ). Note that the arrays at this last level are only allocated if necessary.

Figure 3.4: The spherical accumulator and its representation in memory (the bottom half is just indicated). The angular discretization (θ, ϕ) behaves like a sphere. The indexing of the azimuthal angle (θ) uses modular arithmetic (*i.e.*, it wraps around). Each (θ, ϕ) cell has a points to a vector (allocated as needed during the voting process) storing the actual votes associated with the distances from the origin (ρ), thus covering all possible orientations and positions.



The accumulator discretization can be adjusted by choosing the number of ϕ cells, since the number of θ cells are automatically calculated to represent the same proportion of the arc length discretization. The number of cells used for radial discretization (ρ) is adjusted according to size of the point cloud to emphasize either performance or precision.

Accumulator structures used for HT store a discrete number of votes, which is true even for the KHT (FERNANDES; OLIVEIRA, 2008). In our solution, a vote represents a plane and the uncertainly associated to its exact location and orientation. Thus, our accumulator uses floats instead integers to reduce the influence of rounding errors. This improves the accuracy of our solution allowing the accumulation of fractional votes, at the expense of a small increment (approximately 4%) in the execution time. Since our solution already achieves real-time performance for relatively large datasets, this additional cost is not perceived by the users. The rounding errors, recently mentioned, would difficult the ordering of accumulator bins in the peak detection, because many bins could have the same value stored. The voting process does not guarantee that cells receive votes

from the center to the borders of the Gaussian (in all directions). This precludes the use of stable sorting algorithms that would keep elements with equal votes in the same relative order as they were voted.

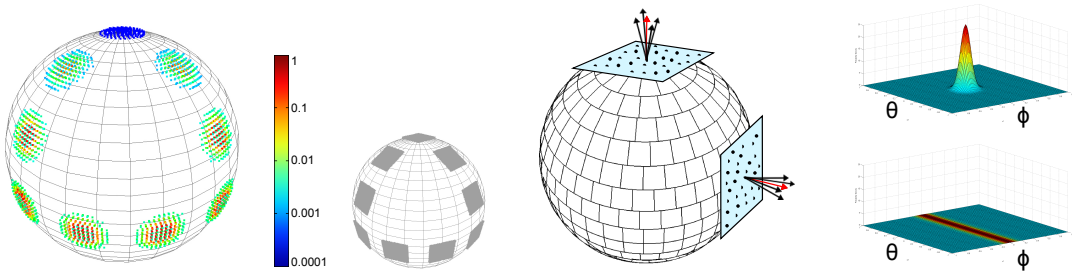
3.6 Peak Detection

The last stage of a Hough-transform consists of detecting peaks of votes in the accumulator. We optimize this process by using an auxiliary 1-D array to store the (θ,ϕ,ρ) coordinates of accumulator cells as they receive votes for the first time during the voting procedure. Only cells in this array need to be inspected for peak detection. As demonstrated in the KHT (FERNANDES; OLIVEIRA, 2008), low-pass filtering the accumulator smooths the voting map, consolidating adjacent peaks. Therefore, before peak detection is actually performed, we apply a low-pass filter to the accumulator cells whose indices have been stored in the auxiliary array. For this, we use a 3-D isotropic kernel comprised of seven cells (the central one and its six-connected neighborhood in 3-D). This topology was used to handle the discretization of the accumulator, which has a singularity at the poles. The kernel weights should satisfy $w_c, w_n > 0$, $w_c + 6w_n = 1$, and $w_c \geq w_n$, where w_c and w_n are the weights of the central and neighbor cells, respectively. Although various combinations of w_c and w_n values produce good results in practice, according to our experience the use of $w_c = 0.2$ and $w_n = 0.133$ seem to provide the best compromise between peak consolidation and smoothness.

The low-pass-filtered values of the voted cells are copied from the accumulator to an auxiliary array and sorted in decreasing order. Iterating over this array, the algorithm selects each peak candidate and checks if the corresponding accumulator cell has already been visited. If not, the cell is chosen as a peak and its (up to) 26 neighbors are tagged as *visited*. If the cell has already been visited, their neighbors are also tagged as *visited*. This procedure guarantees that only true peaks will be selected to represent the output planes.

The number of votes cast by a plane on the cells of a spherical accumulator decreases as one moves from the equator to the poles. This is illustrated in Figure 3.5(a), which compares the distribution of votes cast by a cluster as it is rotated around the origin. The color scale indicates the number of cast votes, while the thumbnail image on its right shows the best fitting planes corresponding to the rotated cluster. Note that the number of votes cast on cells around the poles is significantly smaller than the ones near the equator. Figures 3.5(b) and 3.5(c) explain this behavior, for a fixed value of the parameter ρ. Figure 3.5(b) shows two versions of the rotated point cloud: one near the equator and the other near the north pole. The noise in the point cloud lends to some uncertainty on the plane's orientation, which is represented by a cone of normals around the normal of the best-fitting plane (shown in red). On the equator, such uncertainty causes some votes to be cast in a small θ and ϕ neighborhood around the (θ, ϕ, ρ) coordinates of the best fitting plane. There, equal angular steps in θ and in ϕ correspond to arc lengths of equal sizes, resulting in an isotropic Gaussian kernel in the (θ, ϕ) subspace. Such a Gaussian is illustrated on the top portion of Figure 3.5(c). Near a pole, on the other hand, the uncertainty on the plane's normal lends to a small uncertainty in the parameter ϕ, but to a huge uncertainty in the parameter θ, as at the pole the value of θ varies from 0 to 360 degrees. This results in a highly anisotropic Gaussian kernel in the (θ, ϕ) subspace, as shown by the truncated kernel at the bottom of Figure 3.5(c). This wider and lower Gaussian covers a large θ neighborhood, but the voting procedure constrains voting to values of at least g_{min} (see line 10 of Algorithm 3), which is reached much earlier than

Figure 3.5: The number of votes cast by a cluster as it is rotated varies with the position on the spherical accumulator (a). The color scale indicates the number of votes, while the thumbnail image on its right shows the best-fitting planes corresponding to the rotated clusters. (b) On the equator, the uncertainty on the plane orientation lends to votes on a small isotropic neighborhood in the (θ, ϕ) subspace. At (next to) a pole, the same uncertainty on the plane orientation lends to a small uncertainty in the ϕ dimension, but to a big uncertainty in the θ dimension, as θ can range from 0 to 360 degrees. (c) isotropic (top) and truncated anisotropic (bottom) Gaussian kernels in the (θ, ϕ) subspace associated to the cluster near the equator and near the pole, respectively.



(a) (b) (c)

the two standard-deviation limit is reached. This explains the smaller number of votes per cell as the cluster moves from the equator the the poles.

Let C_1 and C_2 be two clusters with the same number of samples and same variances, C_1 located near the equator and C_2 near a pole. Considering only the number of votes, C_1 would always be detected first, as its peak is higher. To retrieve the detected planes based on how representative they are for a scene, as opposed to just on the heights of their associated peaks, the list of detected planes is sorted based on the sum of the weights (Equation 3.7) of all clusters that voted for each plane π_i:

$$w_{sum}(\pi_i) = \sum_{voted\ for\ \pi_i} w_{cluster}. \tag{3.8}$$

3.7 Algorithm Complexity

The overall steps of our plane-detection technique can be summarized in Algorithm 4. The cost of constructing an octree for a set of $|P|$ samples is $O(|P| \log |P|)$. Checking whether a cluster C_i with $|C_i|$ samples (inside an octree node) is approximately coplanar requires computing its covariance matrix $\Sigma_{(x,y,z)}$ and comparing its eigenvalues (lines 7 and 8 in Algorithm 2). These operations are performed in time $O(|C_i|)$. Since this operation is used to decide if a node needs to be subdivided, it has to be performed in all nodes of the octree, resulting in a total cost of $O(|P| \log |P|)$. Transforming $\Sigma_{(x,y,z)}$ to the (θ, ϕ, ρ) space requires computing a Jacobian matrix and multiplying three 3×3 matrices (see Equations 3.3 and 3.4), which has cost $O(1)$. Computing the eigenvalues of $\Sigma_{(\theta,\phi,\rho)}$ costs $O(1)$. These operations (lines 5 and 6 in Algorithm 4) are executed once per cluster, whose number is bounded by $O(|P|)$. Each cluster C_i casts votes over a total v_{C_i} cells. Note that since in a spherical accumulator each cell represents a (set of) plane orientation(s), each cluster only votes for a relatively small number of cells. Thus, typical numbers for v_{C_i} vary from 20 to 50, depending on the distribution of samples

in the cluster, and on the resolution of the accumulator. Let B be the total number of accumulator bins that received some votes. Since there are $O(|P|)$ clusters and each cluster votes for a finite number of cells, $B = O(|P|)$. Filtering any given accumulator cell is performed in $O(1)$ time, for a total cost of $O(|P|)$. Sorting the B voted cells is accomplished in $O(|P| \log |P|)$, and peak detection has cost $O(|P|)$. Thus, the overall time complexity of the algorithm is $O(|P| \log |P|)$.

Algorithm 4 Algorithm Summary and Asymptotic Cost

Require: P {point cloud}
1: Octree generation; $\{O(|P| \log |P|)\}$
2: **for** each octree node **do**
3: Compute cluster covariance matrix $\Sigma_{(x,y,z)}$; $\{O(|C_i|)\}$
4: **if** cluster is approximate co-planar **then**
5: Transform covariance matrix $\Sigma_{(x,y,z)}$ to (θ, ϕ, ρ) space; $\{O(1)\}$
6: Compute eigenvalues of $\Sigma_{(\theta,\phi,\rho)}$; $\{O(1)\}$
7: Cast cluster votes and update the auxiliary array (AA); $\{O(1)\}$
8: **end if**
9: **end for**
10: Filter accumulator cells pointed by AA; $\{O(|P|)\}$
11: Copy accumulator votes to AA and sort them; $\{O(|P| \log |P|)\}$
12: Iterate over sorted AA detecting peaks; $\{O(|P|)\}$

3.8 Space Complexity

The amount of memory required by our 3-D Kernel-based Hough transform consists basically of the octree (used to store point-cloud samples and indexes), the voting map, and the trivariate kernels. Except for the root, which stores the samples, each octree node only stores (integer) indexes for the sample array. Since each sample stores its (x, y, z) coordinates as doubles, the memory required for the octree is given by $3 \times 8 \times |P|$ bytes for the samples themselves, and $4 \times |P| (\log_8 |P| - 1)$ bytes for the remainder of the octree. Thus, for instance, assuming a point cloud with 20 million samples, the octree would require approximately 1.50 GB of memory allocation. The voting map, in turn, depends on the discretization of the Hough space (θ, ϕ, ρ) and on how many cells receive votes (only voted cells are allocated in memory). Each accumulator cell consists of one float for storing its votes, and one boolean to indicate if it has already been inspected by the peak-detection procedure. An accumulator with discretization of $\phi = 30$ and $\rho = 300$ would require 1690.84 KB if all cells were initialized. Finally, a trivariate kernel stores a 3×3 covariance matrix and the (θ, ϕ, ρ) Hough coordinates of the best fitting plane, resulting in 12 doubles per cluster. While voting could be performed in parallel on-the-fly as we reach the octree leaf nodes, we have not implemented concurrency control mechanisms for accessing the accumulator, and voting is performed in a serial fashion (see Algorithm 4). The space complexity of the algorithm is then $O(|P| \log_8 |P|)$.

3.9 Summary

This Chapter described the pipeline used to detect planes in unorganized point clouds. It also discussed challenges involved in maximizing computing performance. Firstly, it

described our clustering approach for segmenting coplanar samples efficiently. Following, it explained how to compute Gaussian kernels for the purpose of voting using 3-D Gaussian distributions. It is illustrated how votes are stored and manipulated in the accumulator, and showed how the peak-detection procedure finds the most predominant bins in the accumulator.

4 RESULTS

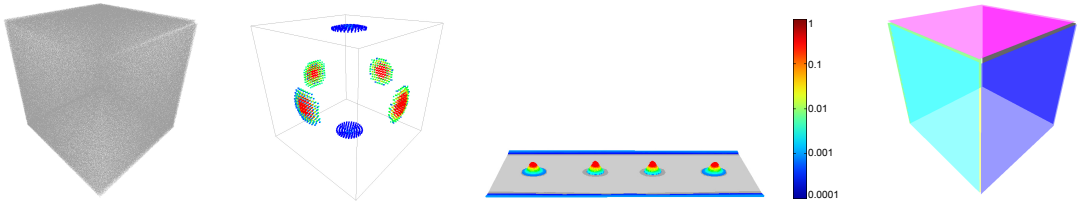
We have implemented our technique in C++, using OpenMP to parallelize the octree generation, *dlib* (DLIB, n.d) to compute eigenvalue decompositions, and OpenGL to render the detected planes. We have used this implementation to automatically detect planes in a large number of unorganized point clouds, and compared its performance against the state-of-the-art approaches: the Randomized Hough transform (RHT) and RANSAC. Since *surface-growing* techniques are not as fast as RHT and RANSAC, and require information about neighbor samples (see Section 2.4), they were not included in our performance comparisons.

4.1 Accuracy Tests

To evaluate the accuracy of our approach, we created a point cloud (Box) by sampling the faces of a cube centered at the origin and with 400 units. Each face of the cube contains $160,801$ samples with 2.5% of uniformly-distributed noise. This point cloud is shown in Figure 4.1 (a) and was also used for the RANSAC experiment shown in Figure 2.9. A 3D perspective and an unfolded slice of the spherical accumulator displaying the six peaks detected by our technique is shown in Figure. 4.1(b). Four of these peaks are equally distributed on the central line (equator) of the accumulator, while gray indicates zero votes. Such peaks correspond to the lateral faces of the cube. The two additional peaks are at the poles (shown as the blue lines on top and at the bottom of the gray region), and correspond to the top and bottom faces of the cube. The detected planes are shown in Figure 4.1(c). Note that only six planes were founded as our trivariate Gaussian kernel naturally handles the noise in the dataset.

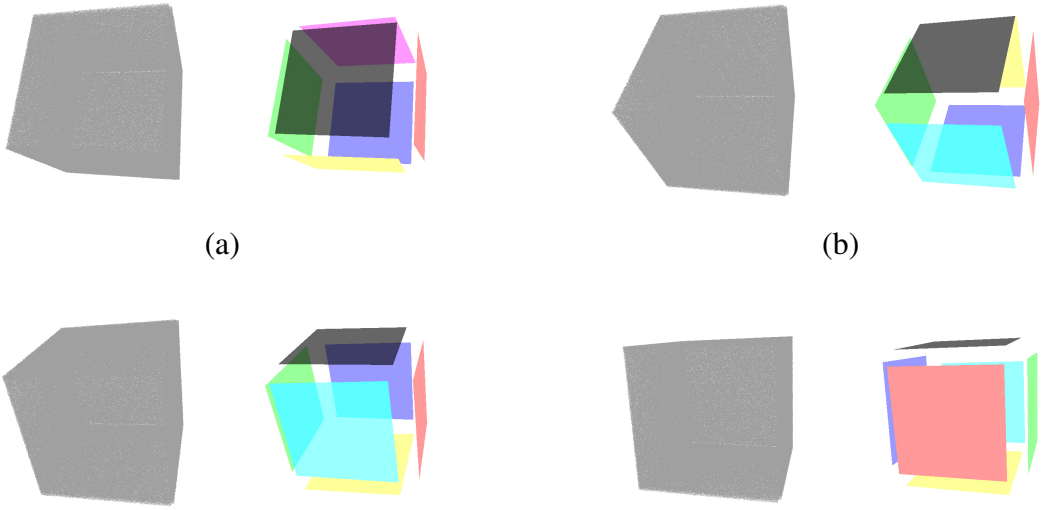
We have rotated the Box point cloud by arbitrary amounts and around arbitrary axes and verified that our technique accurately detected the planes in all cases. In Figure 4.2, we show an example of rotating the cube only in the x-axis for better viewing.

Figure 4.1: Box dataset. (a) A point cloud representing a cube centered at the origin. Each face consists of $160,801$ points with 2.5% of uniformly-distributed noise. (b) A 3D visualization of an unfolded slice of the accumulator representing all pairs (θ, ϕ) for one value of ρ after the voting procedure. There are six detected peaks: four equally distributed on the gray region represent the lateral faces of the cube, plus two at the poles (shown as the blue lines) corresponding to the top and bottom faces. (c) Reconstructed planes from the peaks detected by our approach.



(a) (b) (c)

Figure 4.2: Point clouds and detected planes after rotating the point cloud by 20, 40, 60 and 80 degrees around the x-axis. The 3D KHT exactly found the six rotated planes in all experiments. They were randomly colored at each execution, except for the dark plane.



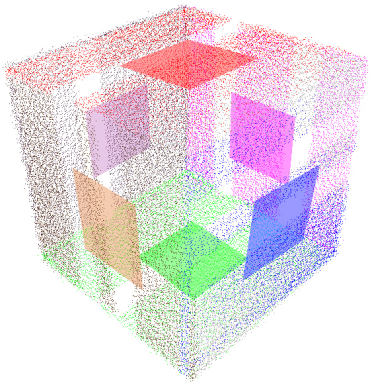
(a) (b)

(c) (d)

To evaluate the robustness of our technique to missing samples and noise, we down-sampled the Box dataset to 48,000 points and added 1% of Gaussian noise. For this experiment, each face of the cube corresponds to three discontinuous stripes of samples covering approximately 60% of its original area. The resulting point cloud is illustrated in Figure 4.3, which also shows six colored squares representing the most important detected planes. As in the previous experiment, the point cloud was rotated by arbitrary amounts around arbitrary axes, consistently producing the same results.

Table 4.1 presents detailed information about the experiments performed with our technique on each dataset. These times were computed by averaging the results of 50 executions. It shows that our approach processes the *Computer* dataset with its 68K samples in approximately 22 milliseconds, and the *Museum* dataset, which contains 179K samples, in 25 milliseconds. For larger point clouds (e.g., *Bremen*, with 20 million samples)

Figure 4.3: Downsampled version of the cube, where each face is covered by three stripes of samples covering approximately 60% of its original area. The squares indicate the detected planes.



the octree creation (column 6) becomes the bottleneck of 3D KHT. Still, our technique processes the Bremen dataset $3,586$ times faster than RANSAC. The available implementation of the RHT could not handle the entire dataset. Working on the full dataset, our technique is still 20 times faster than the RHT working on a subset containing only 10% of the original samples (Table 4.2). Table 4.1 also shows that even though the octree might segment large coplanar structures into several clusters, such clusters end up voting for the same regions in the accumulator, resulting in the detection of the right planes. In our experiments, all datasets were processed in their original scales.

Table 4.1: Data on the experiments performed with our technique. Number of samples in the point clouds, numbers of detected clusters, number of samples used in the voting procedure, octree-generation time, voting time, peak-detection time, and number of detected planes.

Point Cloud			Octree			Time (sec)			Result
Name	Size	Bounding Box	Clusters	Used Points	Rate(%)	Clustering	Voting	Peaks	Planes
Computer	68 852	$1.3 \times 3.0 \times 0.9$	119	30 630	44.48	0.005	0.009	0.008	8
Room	112 586	$29.2 \times 14.5 \times 3.1$	339	66 682	59.22	0.009	0.02	0.012	40
Utrecht	160 256	$75.8 \times 75.8 \times 37.3$	393	92 839	57.93	0.024	0.005	0.011	38
Museum	179 744	$72.4 \times 132.8 \times 23.1$	232	121 943	67.84	0.013	0.007	0.005	21
Box	964 806	$409.9 \times 409.9 \times 409.9$	144	584 028	60.53	0.054	0.008	0.015	6
Bremen	20 332 246	$110.2 \times 379.3 \times 84.6$	7 489	17 929 145	88.18	2.05	0.033	0.022	202

4.2 Comparison with Other Methods

For performance comparisons, we used the RANSAC implementation for plane detection in point clouds available in the *Point Cloud Library* v1.7 (RUSU; COUSINS, 2011), a modern C++ library for 3D point-cloud processing. For RHT, we used the implementation by Borrmann et al. (2011). These implementations proved to be the most efficient

ones for plane detection using RANSAC and RHT, respectively. All experiments were performed on an Intel i7-2600 3.4 GHz CPU with 16 GB of RAM. The codes for the three techniques (RHT, RANSAC, and ours) were compiled using Visual Studio 2012 for 64 bits on Windows 7 to exploit the full potential of the hardware. Figure 4.5 shows the datasets used to compare the performance of the three techniques. On the right, it also shows the most representative planes detected by our approach (for the more complex examples, some planes are not shown to avoid cluttering the images). These datasets include a computer desk (*Computer*), a room (*Room*), a set of fa cades from a city block (*Utrecht*), the interior of a museum (*Museum*), and a partial scanning of some buildings in the city of Bremen (*Bremen*). These datasets were obtained using 3D scanners or extracted from a collection of photographs. They were chosen because they span a large range of parameters, varying in number of points, sampling rate, occupied volume, and number of detectable planes.

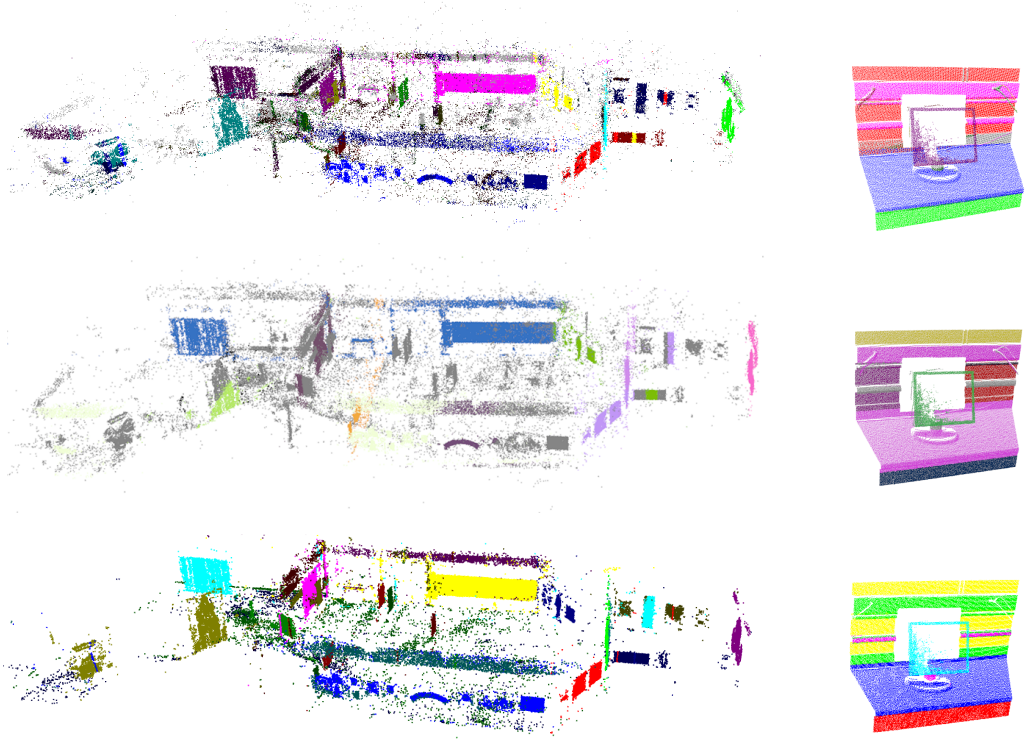
Table 4.2 compares the performance of our technique to RHT and RANSAC. These results show that our approach is one to four orders of magnitude faster than the competing ones. Although RHT and RANSAC are relatively fast on small datasets containing low noise and just a few planar structures (e.g., *Computer*), they are not as efficient on bigger and noisier datasets (e.g., *Museum* and *Bremen*). Our approach, on the other hand, can efficiently handle both large and noisy datasets. RANSAC takes approximately 2 hours to detect 16 planes in the *Bremen* dataset, even though the 3 most representative planes were detected in about 3 minutes. This happens because RANSAC removes samples associated to the detected planes, increasing the proportion of outlier samples among the remaining ones. In contrast, our approach took 2.1 seconds to detect all the representative planes (Figure 4.5).

Table 4.2: Performance comparison of our approach (3D KHT) against RHT and RANSAC for various datasets. The entries of the table show the execution times (in seconds) of the three techniques for these datasets. (*) The RHT was computed with a simplified version of Bremen dataset containing only 2 million samples, because the available implementation did not support larger inputs.

	Computer	Room	Utrecht	Museum	Bremen
3D KHT	0.022	0.041	0.040	0.025	2.105
RHT	0.121	6.313	2.814	11.96	42.824 *
RANSAC	0.424	3.293	15.412	302.61	7531.01

While 3D KHT and RANSAC use the same number of parameters (less than RHT), the 3D KHT is less dependent on them. This is because our approach performs adaptive clustering based on relative measurements of the samples' variances, instead of using specific thresholds. Since the running times of the three algorithms are affected by the selected parameters, we chose values that optimize their execution times. Figure **??** compares the planes detected by the three techniques on two datasets. The results are similar, but our technique is significantly faster (Table 4.2).

Figure 4.4: Examples of plane detection for two datasets using our technique (top), RHT (middle), and RANSAC (bottom). While our approach detected 20 planes in the Museum dataset, RHT detected 10 and RANSAC detected 22. The actual results are similar, but our technique is significantly faster. For the Museum dataset, our technique is two orders of magnitude faster than RHT and three orders of magnitude faster than RANSAC (Table 4.2).



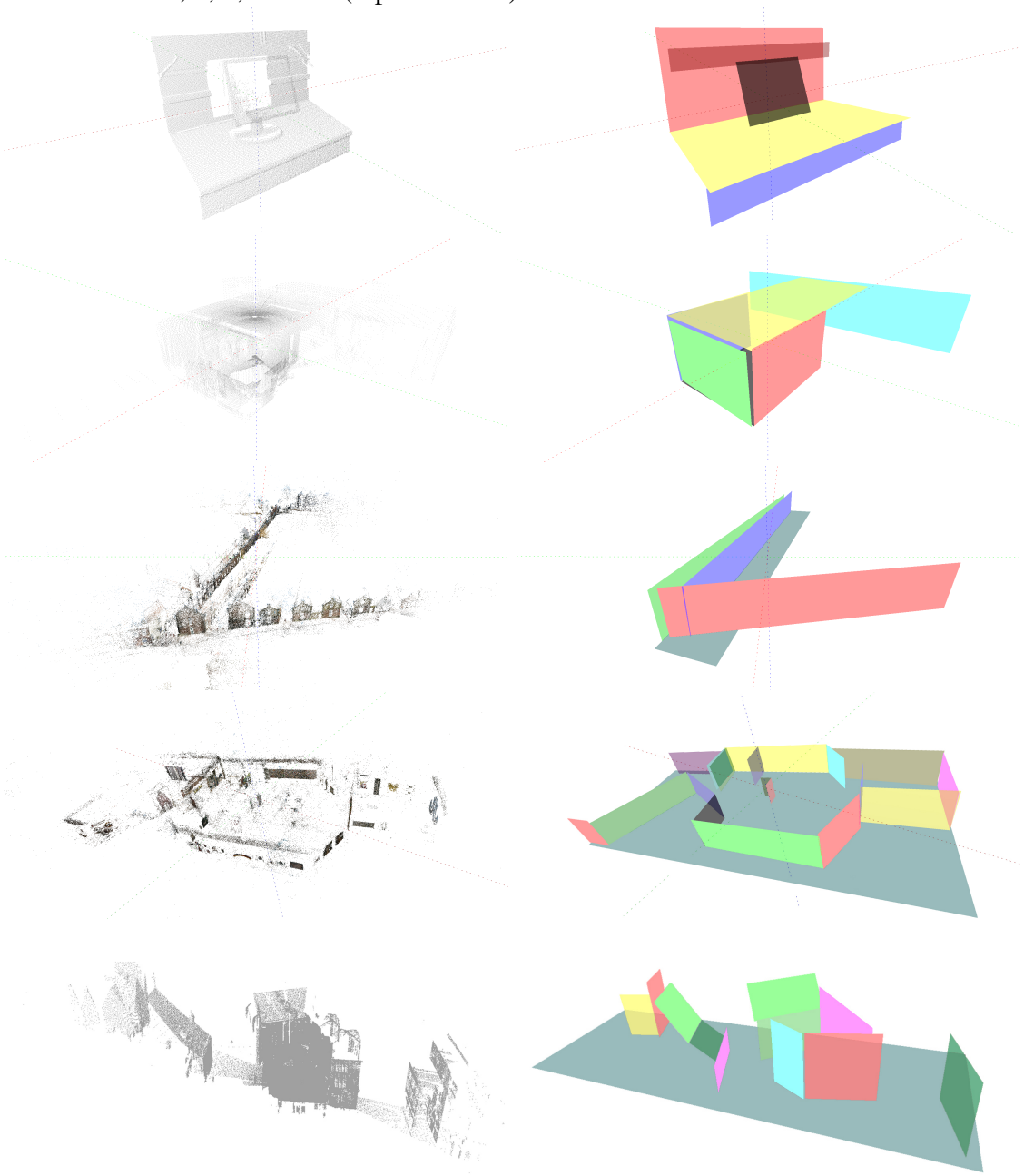
4.3 Limitations

Our technique has some limitations. Even though the best-fitting plane for any given cluster can have arbitrary position and orientation, the position and orientation of the detected planes is constrained by the accumulator discretization. Such a restriction is shared by all Hough-transform techniques. If a valid plane is immersed in very high levels of noise, our approach might not be able to detect it. This may happen if the portions of the point cloud inside the octree nodes are still sufficiently noisy to not allow the detection of approximately coplanar clusters. Also, if the samples in an octree node are left-overs from its neighbors, our technique may fit a spurious plane through these samples. According to our experience, starting the approximate-coplanarity check after the third level of the octree subdivision tends to avoid this problem.

4.4 Summary

This Chapter reported the use of our technique in different types of point clouds, varying in size (number of points), sampling (density) and complexity (number of planes). It has shown that our technique is significantly faster than the state-of-the-art approaches and is able to accurately detect planes.

Figure 4.5: Datasets used for performance comparison. Point clouds (left) and the most representative planes detected by our technique (right). From top to bottom, the datasets are: *Computer*, *Room*, *Utrecht*, *Museum*, and *Bremen*. Their numbers of samples are shown in Table 4.1. The detected planes were resized for better visualization. For all datasets, the accumulator discretization was obtained using $\phi_{max} = 30$ and $\rho_{max} = 300$. The threshold s_{ms} was set to 30. s_{level} was chosen for each point cloud to generate the best results as 2, 4, 5, 6 and 7 (top to bottom).



5 CONCLUSIONS AND FUTURE WORK

This thesis presented an $O(n \log n)$ Hough-transform technique to perform deterministic plane detection in unorganized point clouds. Our approach uses a fast and robust algorithm to segment clusters of approximately coplanar samples, and casts votes for individual clusters, instead of for individual samples, on a spherical accumulator. For this, we use a trivariate Gaussian kernel that models the uncertainty about the position and orientation of the plane represented by the cluster.

While previous approaches for plane detection have basically resorted to randomly selecting a subset of the samples as a way to reduce execution time, we have undertaken the more fundamental strategy of designing an efficient algorithm with lower asymptotic cost.

Probabilistic approaches are good at finding the first few best planes (usually four or five on average). However, as the points that lie on these planes are removed, the amount of noise relative to the number of left samples tends to increase. Thus, the odds of finding additional relevant planes in the resulting point cloud tend to decrease. In contrast, our approach scans the entire point cloud without removing partial information, thus keeping the inliners/outliers ratio constant.

Our experiments have shown that our approach is several orders of magnitude faster than existing (non-deterministic) techniques for plane detection in point clouds, such as RHT and RANSAC, and scales better with the size of the datasets, since asymptotic constants of previous techniques are too big. It is also robust to noise and to irregularly-distributed samples. As such, it has the potential to enable a new range of applications that require fast detection of planar features on large datasets.

Our approach can be further optimized using CUDA to obtain a more efficient subdivision procedure. The use of concurrency control mechanisms for accessing the accumulator would allow voting to be performed in parallel.

REFERENCES

BESL, P. J.; JAIN, R. C. Segmentation through Variable-Order Surface Fitting. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Washington, DC, USA, v.10, n.2, p.167–192, Mar. 1988.

BIOSCA, J. M.; LERMA, J. L. Unsupervised robust planar segmentation of terrestrial laser scanner point clouds based on fuzzy clustering methods. **ISPRS Journal of Photogrammetry and Remote Sensing**, [S.l.], v.63, n.1, p.84–98, 2008.

BORRMANN, D.; DORIA, D. **Hough Transform Plane Detector**. [S.l.]: Jacobs University, 2011.

BORRMANN, D. et al. The 3D Hough Transform for plane detection in point clouds: a review and a new accumulator design. **3D Research**, Secaucus, NJ, USA, v.2, n.2, p.32:1–32:13, mar 2011.

CHALMOVIANSKY, P.; JUTTLER, B. Filling Holes in Point Clouds. In: WILSON, M.; MARTIN, R. (Ed.). **Mathematics of Surfaces**. [S.l.: s.n.], 2003. p.196–212. (Lecture Notes in Computer Science, v.2768).

CHAPERON, T.; GOULETTE, F. Extracting Cylinders in Full 3D Data Using a Random Sampling Method and the Gaussian Image. In: VISION MODELING AND VISUALIZATION CONFERENCE 2001, 2001. **Proceedings...** Aka GmbH, 2001. p.35–42. (VMV '01).

CHEKHLOV, D. et al. Ninja on a Plane: automatic discovery of physical planes for augmented reality using visual slam. In: IEEE AND ACM INTERNATIONAL SYMPOSIUM ON MIXED AND AUGMENTED REALITY, 2007., 2007, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2007. p.1–4. (ISMAR '07).

CHEN, D. S. A Data-Driven Intermediate Level Feature Extraction Algorithm. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Washington, DC, USA, v.11, n.7, p.749–758, July 1989.

DESCHAUD, J.-E.; GOULETTE, F. A fast and accurate plane detection algorithm for large noisy point clouds using filtered normals and voxel growing. In: FIFTH INTERNATIONAL SYMPOSIUM ON 3D DATA PROCESSING, VISUALIZATION AND TRANSMISSION (3DPVT'10), 2010, Paris, France. **Proceedings...** [S.l.: s.n.], 2010.

DLIB. **dlib c++ library**. n.d.

DUAN, F.; WANG, L.; GUO, P. RANSAC based ellipse detection with application to catadioptric camera calibration. In: NEURAL INFORMATION PROCESSING: MODELS AND APPLICATIONS - VOLUME PART II, 17., 2010, Berlin, Heidelberg. **Proceedings...** Springer-Verlag, 2010. p.525–532. (ICONIP'10).

DUDA, R. O.; HART, P. E. Use of the Hough transformation to detect lines and curves in pictures. **Communications of the ACM**, New York, NY, USA, v.15, n.1, p.11–15, Jan. 1972.

FERNANDES, L. A. F.; OLIVEIRA, M. M. Real-time line detection through an improved Hough transform voting scheme. **Pattern Recognition**, New York, NY, USA, v.41, n.1, p.299–314, Jan. 2008.

FISCHLER, M. A.; BOLLES, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. **Communications of the ACM**, New York, NY, USA, v.24, n.6, p.381–395, June 1981.

FISCHLER, M. A.; BOLLES, R. C. Perceptual Organization and Curve Partitioning. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Washington, DC, USA, v.8, n.1, p.100–105, Jan. 1986.

FUCHS, H.; KEDEM, Z. M.; USELTON, S. P. Optimal Surface Reconstruction from Planar Contours. **Communications of the ACM**, New York, NY, USA, v.20, n.10, p.693–702, Oct. 1977.

HAUSNER, C. **SynthExport v. 1.1.0**. 2010.

HE, X.; NIYOGI, P. Locality Preserving Projections. In: THRUN, S.; SAUL, L.; SCHöLKOPF, B. (Ed.). **Advances in Neural Information Processing Systems 16**. Cambridge, MA: MIT Press, 2004. p.153–160.

HOUGH, P. **Method and means for recognizing complex patterns**. 1962.

HUANG, H.; BRENNER, C.; SESTER, M. 3D building roof reconstruction from point clouds via generative models. In: ACM SIGSPATIAL INTERNATIONAL CONFERENCE ON ADVANCES IN GEOGRAPHIC INFORMATION SYSTEMS, 19., 2011, New York, NY, USA. **Proceedings...** ACM, 2011. p.16–24. (GIS '11).

KAUCIC, R.; HARTLEY, R.; DANO, N. Plane-based projective reconstruction. In: COMPUTER VISION, 2001. ICCV 2001. EIGHTH IEEE INTERNATIONAL CONFERENCE ON, 2001. **Proceedings...** [S.l.: s.n.], 2001. v.1, p.420–427 vol.1.

KIRYATI, N.; ELDAR, Y.; BRUCKSTEIN, A. M. A probabilistic Hough transform. **Pattern Recognition**, New York, NY, USA, v.24, n.4, p.303–316, Feb. 1991.

KOLLURI, R.; SHEWCHUK, J. R.; O'BRIEN, J. F. Spectral surface reconstruction from noisy point clouds. In: EUROGRAPHICS/ACM SIGGRAPH SYMPOSIUM ON GEOMETRY PROCESSING, 2004., 2004, New York, NY, USA. **Proceedings...** ACM, 2004. p.11–21. (SGP '04).

LANMAN, D. et al. Multi-flash 3D photography: capturing shape and appearance. In: ACM SIGGRAPH 2006 RESEARCH POSTERS, 2006, New York, NY, USA. **...** ACM, 2006. (SIGGRAPH '06).

LIU, Y. Automatic registration of overlapping 3D point clouds using closest points. **Image and Vision Computing**, [S.l.], v.24, n.7, p.762–781, 2006.

MATAS, J.; GALAMBOS, C.; KITTLER, J. Progressive Probabilistic Hough Transform. In: BRITISH MACHINE VISION CONFERENCE, 1998. **Proceedings...** BMVA Press, 1998. p.26.1–26.10. doi:10.5244/C.12.26.

MCLACHLAN, G. **Discriminant Analysis and Statistical Pattern Recognition**. [S.l.]: Wiley, 2004. (Wiley Series in Probability and Statistics).

MEDIONI, G.; LEE, M.; TANG, C. **A Computational Framework for Segmentation and Grouping**. [S.l.]: Elsevier Science, 2000.

MITRA, N. J.; NGUYEN, A. Estimating Surface Normals in Noisy Point Cloud Data. In: NINETEENTH ANNUAL SYMPOSIUM ON COMPUTATIONAL GEOMETRY, 2003, New York, NY, USA. **Proceedings...** ACM, 2003. p.322–328. (SCG '03).

NGUYEN, H. H. et al. Accurate and Fast Extraction of Planar Surface Patches from 3D Point Cloud. In: INTERNATIONAL CONFERENCE ON UBIQUITOUS INFORMATION MANAGEMENT AND COMMUNICATION, 7., 2013, New York, NY, USA. **Proceedings...** ACM, 2013. p.84:1–84:8. (ICUIMC '13).

NING, X. et al. Segmentation of architecture shape information from 3D point cloud. In: INTERNATIONAL CONFERENCE ON VIRTUAL REALITY CONTINUUM AND ITS APPLICATIONS IN INDUSTRY, 8., 2009, New York, NY, USA. **Proceedings...** ACM, 2009. p.127–132. (VRCAI '09).

OGUNDANA, O. O. et al. Automated detection of planes in 3-D point clouds using fast Hough transforms. **Optical Engineering**, [S.l.], v.50, n.5, p.053609, may 2011.

PARRATT, L. **Probability and experimental errors in science**: an elementary survey. [S.l.]: Wiley, 1961. (Science Editions).

PETERNELL, M.; STEINER, T. Reconstruction of piecewise planar objects from point clouds. **Computer-Aided Design**, [S.l.], v.36, n.4, p.333–342, 2004.

PHOTOSYNTH. **Photosynth**. [S.l.]: Microsoft, 2008.

POPPINGA, J. et al. Fast plane detection and polygonalization in noisy 3D range images. In: INTELLIGENT ROBOTS AND SYSTEMS, 2008. IROS 2008. IEEE/RSJ INTERNATIONAL CONFERENCE ON, 2008. **Proceedings...** [S.l.: s.n.], 2008. p.3378–3383.

RICHTSFELD, M.; VINCZE, M. **Robotic Grasping of Unknown Objects**. [S.l.: s.n.], 2009.

ROTHWELL, C. A. et al. Planar object recognition using projective shape representation. **International Journal of Computer Vision**, Hingham, MA, USA, v.16, n.1, p.57–99, Sept. 1995.

RUSU, R.; COUSINS, S. 3D is here: point cloud library (pcl). In: ROBOTICS AND AUTOMATION (ICRA), 2011 IEEE INTERNATIONAL CONFERENCE ON, 2011. **Proceedings...** [S.l.: s.n.], 2011. p.1–4.

SCHNABEL, R.; WAHL, R.; KLEIN, R. Efficient RANSAC for Point-Cloud Shape Detection. **Computer Graphics Forum**, [S.l.], v.26, n.2, p.214–226, June 2007.

SEITZ, S. et al. A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms. In: COMPUTER VISION AND PATTERN RECOGNITION, 2006 IEEE COMPUTER SOCIETY CONFERENCE ON, 2006. **Proceedings...** [S.l.: s.n.], 2006. v.1, p.519–528.

SHAKARJI, C. M. Least-squares Fitting algorithms of the NIST algorithm testing system. **Journal of Research of the National Institute of Standards and Technology**, [S.l.], v.103, n.6, p.633–641, 1998.

SIMON, G.; FITZGIBBON, A.; ZISSERMAN, A. Markerless tracking using planar structures in the scene. In: AUGMENTED REALITY, 2000. (ISAR 2000). IEEE AND ACM INTERNATIONAL SYMPOSIUM ON, 2000. **Proceedings...** [S.l.: s.n.], 2000. p.120–128.

TAO, D. et al. General Averaged Divergence Analysis. In: SEVENTH IEEE INTERNATIONAL CONFERENCE ON DATA MINING, 2007., 2007, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2007. p.302–311. (ICDM '07).

TARSHA-KURDI, F.; LANDES, T.; GRUSSENMEYER, P. 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 Sciences, ISPRS**, [S.l.], 2007.

TONG, Y. L. **The Multivariate Normal Distribution**. [S.l.]: Springer, 1990.

TRIGGS, B. Autocalibration from Planar Scenes. In: EUROPEAN CONFERENCE ON COMPUTER VISION-VOLUME I - VOLUME I, 5., 1998, London, UK, UK. **Proceedings...** Springer-Verlag, 1998. p.89–105. (ECCV '98).

VIDAL, R. Subspace Clustering. **Signal Processing Magazine, IEEE**, [S.l.], v.28, n.2, p.52–68, March 2011.

VIDAL, R.; MA, Y.; SASTRY, S. Generalized principal component analysis (GPCA). In: COMPUTER VISION AND PATTERN RECOGNITION, 2003. 2003 IEEE COMPUTER SOCIETY CONFERENCE ON, 2003. **Proceedings...** [S.l.: s.n.], 2003. v.1, p.I–621–I–628 vol.1.

VOSSELMAN, G.; DIJKMAN, E. 3D building model reconstruction from point clouds and ground plans. **International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences**, Annapolis, MA, USA, v.34, p.37–44, Oct 2001.

VOSSELMAN, G. et al. Recognizing Structure in laser scanner point clouds. **International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. 46**, [S.l.], 2004.

WANG, J.; OLIVEIRA, M. M. Filling holes on locally smooth surfaces reconstructed from point clouds. **Image and Vision Computing**, [S.l.], v.25, n.1, p.103–113, 2007. SIBGRAPI.

XU, L.; OJA, E.; KULTANEN, P. A new curve detection method: randomized hough transform (RHT). **Pattern Recognition Letters**, New York, NY, USA, v.11, n.5, p.331–338, May 1990.

YLä-JääSKI, A.; KIRYATI, N. Adaptive Termination of Voting in the Probabilistic Circular Hough Transform. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Washington, DC, USA, v.16, n.9, p.911–915, Sept. 1994.

ZHANG, Z. Microsoft Kinect Sensor and Its Effect. **IEEE MultiMedia**, Los Alamitos, CA, USA, v.19, n.2, p.4–10, Apr. 2012.

APPENDIX A DERIVING THE JACOBIAN MATRIX

This Section details the derivation of the Jacobian matrix in Equation 3.4. The following equations show the partial derivatives of ρ with respect to p_x, y_p and p_z:

$$\rho = \sqrt{p_x^2 + p_y^2 + p_z^2} \tag{A.1}$$

$$\frac{\partial \rho}{\partial p_x} = \frac{1}{2}(p_x^2 + p_y^2 + p_z^2)^{-\frac{1}{2}} \times 2p_x \tag{A.2}$$

$$\frac{\partial \rho}{\partial p_x} = \frac{p_x}{\sqrt{p_x^2 + p_y^2 + p_z^2}} = n_x \tag{A.3}$$

$$\frac{\partial \rho}{\partial p_y} = \frac{1}{2}(p_x^2 + p_y^2 + p_z^2)^{-\frac{1}{2}} \times 2p_y \tag{A.4}$$

$$\frac{\partial \rho}{\partial p_y} = \frac{p_y}{\sqrt{p_x^2 + p_y^2 + p_z^2}} = n_y \tag{A.5}$$

$$\frac{\partial \rho}{\partial p_z} = \frac{1}{2}(p_x^2 + p_y^2 + p_z^2)^{-\frac{1}{2}} \times 2p_z \tag{A.6}$$

$$\frac{\partial \rho}{\partial p_z} = \frac{p_z}{\sqrt{p_x^2 + p_y^2 + p_z^2}} = n_z \tag{A.7}$$

Since the vectors $\vec{p} = (p_x, p_y, p_z)$ and $\vec{n} = (n_x, n_y, n_z)$ have the same direction, each component of $\vec{p}$ is normalized in Equations A.3, A.5 and A.7, so that the norm of $\vec{p}$ is one, corresponding to $\vec{n}$. The following set of Equations show the partial derivatives of ϕ (from Equation A.8 to A.15), and θ (from Equation A.16 to A.24), with respect to p_x, p_y and p_z:

$$\phi = \arccos \frac{p_z}{\rho} \tag{A.8}$$

$$\phi = \cos^{-1}\left(\frac{p_z}{\sqrt{p_x^2+p_y^2+p_z^2}}\right) \tag{A.9}$$

$$\frac{\partial\phi}{\partial p_x} = \frac{p_x p_z}{\sqrt{\frac{p_x^2+p_y^2}{p_x^2+p_y^2+p_z^2}}(p_x^2+p_y^2+p_z^2)^{\frac{3}{2}}} \tag{A.10}$$

$$\frac{\partial\phi}{\partial p_x} = \frac{p_x p_z}{\sqrt{\frac{w}{\rho^2}}(\rho^2)^{\frac{3}{2}}} \tag{A.11}$$

$$\frac{\partial\phi}{\partial p_x} = \frac{p_x p_z}{\sqrt{w}\rho^2} \tag{A.12}$$

$$\frac{\partial\phi}{\partial p_y} = \frac{p_y p_z}{\sqrt{\frac{p_x^2+p_y^2}{p_x^2+p_y^2+p_z^2}}(p_x^2+p_y^2+p_z^2)^{\frac{3}{2}}} \tag{A.13}$$

$$\frac{\partial\phi}{\partial p_y} = \frac{p_y p_z}{\sqrt{\frac{w}{\rho^2}}(\rho^2)^{\frac{3}{2}}} \tag{A.14}$$

$$\frac{\partial\phi}{\partial p_y} = \frac{p_y p_z}{\sqrt{w}\rho^2} \tag{A.15}$$

$$\frac{\partial\phi}{\partial p_z} = -\frac{\sqrt{\frac{p_x^2+p_y^2}{p_x^2+p_y^2+p_z^2}}}{\sqrt{p_x^2+p_y^2+p_z^2}} \tag{A.16}$$

$$\frac{\partial\phi}{\partial p_z} = -\frac{\sqrt{\frac{w}{\rho^2}}}{\sqrt{\rho^2}} \tag{A.17}$$

$$\frac{\partial\phi}{\partial p_z} = -\frac{\sqrt{w}}{\rho^2} \tag{A.18}$$

$$\theta = \arctan\frac{p_y}{p_x} \tag{A.19}$$

$$\frac{\partial\theta}{\partial p_x} = -\frac{p_y}{p_x^2+p_y^2} \tag{A.20}$$

$$\frac{\partial\theta}{\partial p_x} = -\frac{p_y}{w} \tag{A.21}$$

$$\frac{\partial\theta}{\partial p_y} = \frac{p_x}{p_x^2+p_y^2} \tag{A.22}$$

$$\frac{\partial\theta}{\partial p_y} = \frac{p_x}{w} \tag{A.23}$$

$$\frac{\partial\theta}{\partial p_z} = 0 \tag{A.24}$$

APPENDIX B DETECÇÃO EM TEMPO REAL DE REGIÕES PLANARES EM NUVENS DE PONTOS NÃO ESTRUTURADAS

Resumo da Dissertação em Português

Detecção automática de regiões planares em nuvens de pontos é um importante passo para muitas aplicações gráficas, de processamento de imagens e de visão computacional. Entre as principais aplicações, podem ser citadas reconstrução de modelos para engenharia reversa (VOSSELMAN; DIJKMAN, 2001; KAUCIC; HARTLEY; DANO, 2001) (TARSHA-KURDI; LANDES; GRUSSENMEYER, 2007; HUANG; BRENNER; SESTER, 2011; FUCHS; KEDEM; USELTON, 1977), calibração automática de câmeras (TRIGGS, 1998), reconhecimento de objetos (ROTHWELL et al., 1995) (PETERNELL; STEINER, 2004), realidade aumentada (SIMON; FITZGIBBON; ZISSERMAN, 2000; CHEKHLOV et al., 2007) e segmentação (BIOSCA; LERMA, 2008; NING et al., 2009). A recente popularização de digitalizadores a laser levou a um aumento no tamanho e na quantidade de dados disponíveis. Além disso, aplicações como o *SynthExport* (HAUSNER, 2010) e o *Photosynth* (PHOTOSYNTH, 2008) nos permitiram extrair nuvens de pontos enormes a partir de fotografias digitais georreferenciadas. Infelizmente, técnicas anteriores para detecção de planos em nuvens de pontos são computacionalmente caras e não escalam bem com o aumento do conjunto de dados, sendo incapazes de alcançar desempenho em tempo real para conjunto de dados contendo dezenas de milhares de pontos, mesmo quando a detecção é feita de um modo não determinístico. Para aumento desse desempenho, várias técnicas exploraram o uso de estratégias não determinísticas, como por exemplo, processar apenas com um subconjunto randômico do original, fazendo com que os resultados dependam desse conjunto selecionado de amostras e não garantindo que todos os planos relevantes serão encontrados e que os resultados serão consistentes ao longo de múltiplas execuções.

Apresentamos uma abordagem determinística para detecção de planos em nuvens de pontos não estruturadas que apresenta complexidade computacional $O(n \log n)$ no número de amostras de entrada. Ela é baseada em um método eficiente de votação para a transformada de Hough. Nossa estratégia agrupa conjuntos de pontos aproximadamente coplanares e deposita votos para estes conjuntos em um acumulador esférico, utilizando núcleos Gaussianos trivariados. Uma comparação com as técnicas concorrentes mostra que nossa abordagem é consideravelmente mais rápida e escala significativamente melhor que as técnicas anteriores, sendo a primeira solução prática para detecção determinística de planos em nuvens de pontos grandes e não estruturadas.

Este trabalho apresenta as seguintes contribuições:

- Uma técnica baseada na transformada de Hough para detectar regiões planares em nuvens de pontos não estruturadas. Nossa solução é robusta a ruído, e independente a amostragem da distribuição. Ela é algumas ordens de magnitude mais rápida e escala melhor que as técnicas do estado da arte. Uma implementação em *software* da nossa técnica consegue processar até 10^5 pontos em tempo real em um computador típico;
- Uma estratégia eficiente de votação para detecção de planos. Nossa solução utiliza uma estratégia de segmentação robusta para identificar *clusters* de amostras aproximadamente coplanares. Votos são depositados para *clusters* ao invés de para amostras individuais.

B.1 Trabalhos Relacionados

Apresentamos nesta secção trabalhos relacionados a detecção de planos em nuvens de pontos. Estes estão subdivididos em cinco áreas: transformada de Hough, *random sample consensus*, *surface growing*, *tensor voting* e *linear subspace learning*. Como poderemos observar, técnicas anteriores, sendo elas determinísticas ou não determinísticas, não são capazes de processar nuvens de pontos suficientemente grandes em tempo real.

B.1.1 Transformada de Hough

A transformada de Hough (HT), inicialmente proposta por Paul Hough (1962) é uma técnica para detecção de características, podendo essas estarem presentes em dados de qualquer dimensão, desde que possam ser escritas a partir de uma equação paramétrica. Hoje em dia, a versão da transformada de Hough utilizada universalmente é a de Duda e Hart (1972) chamada de transformada de Hough padrão (SHT), que utiliza uma parametrização mais eficiente do que a utilizada por Paul Hough. Embora sejam robustas a ruído, ambas as técnicas são computacionalmente caras.

Para contornar o alto custo computacional do processo de votação da transformada de Hough, visto que este é o gargalo do algoritmo, várias técnicas foram propostas. A *Probabilistic Hough transform* proposta por Kiryati et al.(1991) busca diminuir o número de amostras processadas, selecionando e votando apenas para um subconjunto randômico de pontos. Contudo, é difícil encontrar um tamanho ótimo para o subconjunto, ao longo que, quanto menor ele for maiores serão os ganhos no tempo de processamento, porém piores serão os resultados. Visto isso a *Adaptive Probabilistic Hough transform* (YLäJääSKI; KIRYATI, 1994) monitora o acumulador durante o processo de votação em busca de estruturas estáveis, fazendo que não seja preciso encontrar um número exato para o tamanho do subconjunto selecionado. A *Progressive Probabilistic Hough transform* (MATAS; GALAMBOS; KITTLER, 1998) tenta diminuir a influência de ruído no acumulador, já que quando um número menor amostras é selecionado, este pode ser mais prejudicado por amostras incorretas. Por fim, a *Randomized Hough transform* (RHT) (XU; OJA; KULTANEN, 1990) vota com uma confiança maior no acumulador. Diferentemente de votar para cada amostra, a RHT vota para planos que ajustam três pontos escolhidos aleatoriamente. Esta abordagem mostrou-se a mais eficiente e é considerada como estado da arte em detecção de planos, considerando a transformada de Hough.

Quando a transformada de Hough é estendida para três dimensões, o acumulador deve adequar-se a este novo espaço. A extensão imediata de um acumulador 2D para 3D é

trivial, adicionando-se uma nova dimensão à estrutura de dados. Porém um acumulador 3D apresenta problemas de discretização para as coordenadas esféricas, contendo várias células que apresentam a mesma posição e inclinação. Para contornar este problema Borrmann (2011) propôs a utilização de um acumulador esférico parametrizado nas mesmas variáveis (θ, ϕ e ρ). Este acumulador possui, para a mesma discretização de um acumulador 3D comum, um número menor de células, visto que ele não apresenta células redundantes. Observando isso, nossa técnica utiliza o acumulador esférico ao invés do acumulador tradicional.

B.1.2 Random Sample Consensus

Outra importante classe de algoritmos que detectam planos é a *random sample consensus* (RANSAC) (FISCHLER; BOLLES, 1981). RANSAC é um técnica amplamente utilizada para detecção de modelos matemáticos, sendo robusta mesmo em altas proporções de *outliers*. Ela pode ser generalizada para resolver diversos tipos de problemas, qualquer que sejam suas dimensões. RANSAC realiza detecção de planos escolhendo randomicamente três pontos, calculando o plano definido por estes e contando quantos pontos (da nuvem de pontos) ajustam o plano com certa tolerância. O número de pontos encontrado representa a pontuação do plano. Fazendo isso iterativamente, o algoritmo vai melhorando a solução, até que a probabilidade de encontrar um plano com uma pontuação maior seja muito pequena. RANSAC depende dos parâmetros de entrada (tolerância) para que sejam encontradas boas soluções em um baixo tempo de execução.

B.1.3 Surface Growing

A terceira classe de técnicas utilizada para identificar planos em nuvens de pontos é *surface growing* (FISCHLER; BOLLES, 1986) - a análoga de *region growing* em três dimensões. Este tipo de técnicas realizam buscas locais, a partir de sementes escolhidas no início do algoritmo, com o objetivo de expandir regiões com características similares. Por realizarem uma busca local, estes algoritmos são ineficientes computacionalmente. Este tipo de técnica também necessita de informações adicionais a respeito da nuvem de pontos, bem como informações de vizinhança e às vezes normais e cores das amostras.

B.1.4 Tensor Voting

Tensor voting (TV) é um *framework* que identifica, ao mesmo tempo, todas estruturas salientes de um conjunto de dados. Ele é baseado em dois componentes: o cálculo tensorial para representação dos dados e uma votação não linear para comunicação da informação. TV consegue detectar estruturas em qualquer dimensionalidade, sendo ainda robusta a ruídos e preservando descontinuidades. Porém, como TV é naturalmente multidimensional, ele não pode ser aplicado para detectar tipos predefinidos de modelos de uma maneira eficiente.

B.1.5 Linear Subspace Learning

O objetivo de técnicas baseadas em *linear subspace learning* (LSL) é encontrar possíveis subespaços lineares ou afins que acomodem a maior quantidade possível de objetos de um conjunto de dados, deixado a detecção mais trivial em espaços de menor dimensionalidade. Estas incluem técnicas baseadas em *principal component analysis* (PCA), *linear discriminant analysis* (LDA), *general averaged divergence analysis* (GADA) e *locality preserving projections* (LPP). Uma revisão mais completa sobre algoritmos LSL

pode ser encontrada em (VIDAL, 2011). Em contraste com técnicas baseadas em reduzir a dimensionalidade dos dados, os quais conseguem resolver problemas mais genéricos, a nossa técnica busca entrar planos em um espaço tridimensional de maneira eficiente.

B.2 Detecção Eficiente de Planos em Nuvens de Pontos

O objetivo do nosso algoritmo é processar qualquer tipo de nuvem de pontos com a mesma garantia. As otimizações propostas neste trabalho para a transformada de Hough tridimensional permitem que uma implementação em software opere em tempo real para conjunto de dados suficientemente grandes (10^5).

B.2.1 Agrupando Amostras Aproximadamente Coplanares

Agrupar amostras aproximadamente coplanares é a chave do nosso método, já que ele aperfeiçoa o processo de votação, o qual é o gargalo da transformada de Hough. Para isso nos utilizamos uma abordagem global de subdivisão (*octree*). Esta abordagem mostrou-se eficiente, pois ela fornece grande controle sobre as dimensões dos nodos da árvore, já que todos possuem as mesmas dimensões para o mesmo nível das ramificações.

O processo de subdivisão inicia com o nodo pai que inclui todo conjunto de dados, o qual é subdividido recursivamente a fim de refinar a árvore. A cada subdivisão são verificadas duas condições: a quantidade de informação e as proporções do subconjunto de dados. Primeiramente é verificado o nível da *octree*, para evitar cálculos desnecessários, já que nos primeiros níveis não há normalmente informações de regiões planares. No segundo passo, verificamos as proporções da distribuição com *principal componente analysis*, identificando quando um conjunto de pontos representa um plano. Quando as duas condições forem satisfeitas, o processo de subdivisão para naquele ramo, caso contrário o processo continua até que não haja informação suficiente mais no nodo, ou seja, quando o número de pontos for menor que um valor predeterminado.

B.2.2 Calculando Núcleos Trivariados para Votação

A votação indicará os possíveis planos que podem ser detectados na nuvem de pontos. Para realizar a votação de um modo eficiente, votos são depositados para cada cluster ao invés de para cada amostra. Para isso, votos gaussianos são depositados no acumulador, centralizados na célula do acumulador que representa o plano que melhor ajusta a distribuição.

Para votarmos no acumulador esférico, é necessário conhecer as variâncias das distribuições dos clusters em relação aos parâmetros do acumulador. Como conhecemos as variâncias em relação ao eixo cartesiano, calculadas anteriormente por PCA, basta propagarmos as variâncias do espaço cartesiano para o espaço paramétrico (esférico). Para isso, utilizamos uma propagação de incertezas de primeira ordem, multiplicando a matriz de covariância nas coordenadas (x, y, z) pela matriz Jacobiana à esquerda e sua transposta a direita, propagando assim as variâncias de (x, y, z) para (θ, ϕ, ρ).

B.2.3 Votando para Clusters Utilizando Distribuições Gaussianas

Uma vez que temos a matriz de covariância calculada nas coordenadas corretas, utilizamos uma distribuição Gaussiana trivariada para amostrar a quantidade de votos que será depositada em cada célula. Votando no acumulador utilizando apenas as variâncias das distribuições não é o suficiente, já que o espaço tridimensional, onde estão as

amostras, é contínuo. Deste modo, propomos uma ponderação dos votos pela área e número de pontos relativo de cada cluster.

B.2.4 Detecção de Picos

O último estágio da transformada de Hough compreende a detecção de picos no acumulador, os quais representarão os planos mais importantes da nuvem de pontos. O processo de detecção de picos é otimizado por um vetor que armazena as células do acumulador que receberam votos durante o processo de votação. Primeiramente, este vetor é convoluído por um filtro Gaussiano que utiliza os 6 vizinhos imediatos, ou seja, variando um eixo a cada amostragem. Este processo suaviza o mapa de votos, consolidando picos adjacentes no acumulador. Após isso, os valores do vetor são ordenados em ordem decrescente e iterados linearmente. Neste processo, vizinhos de cada célula são inspecionados em busca de células que já foram inspecionadas por este mesmo processo. Caso nenhum vizinho de uma célula fora anteriormente inspecionado, esta célula é escolhida como pico. À medida que os planos se tornam mais horizontais, a variância em θ tende a ficar muito grande, fazendo com que os votos sejam menores nas regiões polares do acumulador. Desde modo, após termos os planos detectados, eles são ordenados por representatividade. A representatividade de cada plano se dá pelo somatório das representatividades dos clusters que votaram naquele plano, ou seja, levando em conta a área e número de pontos relativo de cada plano.

B.3 Resultados

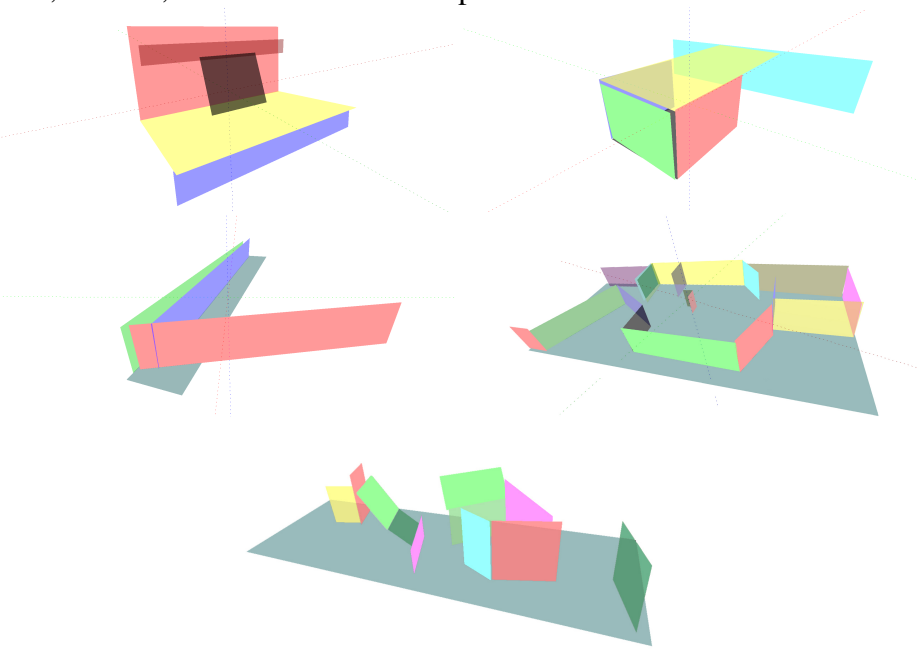
Para comprovar a eficácia da nossa técnica, planos foram detectados em nuvens de pontos, que variam em tamanho (número de pontos), amostragem (densidade) e complexidade (número de planos). Para testes de acurácia, criamos uma nuvem de pontos sintética de uma caixa com 964,806 pontos. Nela, testamos a detecção de planos utilizando versões rotacionadas em eixos e angulações arbitrárias com a presença de $2,5\%$ de ruído distribuído uniformemente. Em todos os casos, nossa técnica foi capaz de detectar os seis planos mais representativos da caixa em suas devidas inclinações e posições. Também foi criada uma nuvem de pontos degenerada de uma caixa, onde fatias dos planos foram removidas, deixando aproximadamente 60% da área original dos planos. Para este conjunto de dados, foi adicionado ruído Gaussiano de 1%. Do mesmo modo, testamos a detecção de planos em versões rotacionadas desta nuvem e verificamos que os seis planos eram sempre detectados.

Para comparação de desempenho testamos nossa técnica contra as técnicas do estado da arte as quais mostraram ser as mais eficientes: a *random sample consensus* e a *randomized Hough transform*. Todas implementações foram feitas em C++ e testadas em um mesmo computador. Tempos obtidos podem ser vistos na Tabela B.1. Respectivas detecções da nossa técnica podem ser vistas na Figura B.1.

Table B.1: Comparação de desempenho entre a nossa técnica e as técnicas do estado da arte. Como podemos perceber, nossa abordagem é entre uma e quatro ordens de magnitude mais rápida que as outras. (*) A RHT foi processada com uma versão simplificada da nuvem de pontos *Bremen* contendo apenas 2 milhões de amostras, já que a implementação disponível não funcionava para conjunto de dados maiores.

	Computer	Room	Utrecht	Museum	Bremen
3D KHT	0.022	0.041	0.040	0.025	2.105
RHT	0.121	6.313	2.814	11.96	42.824 *
RANSAC	0.424	3.293	15.412	302.61	7531.01

Figure B.1: Exemplo de detecção de planos utilizando nossa técnica. Em cada imagem são mostrados os planos mais representativos encontrados nas nuvens de pontos *Computer*, *Room*, *Utrecht*, *Museum* e *Bremen* respectivamente.



B.4 Conclusão

Este trabalho apresentou uma técnica baseada na transformada de Hough que detecta de planos em nuvens de pontos não estruturadas de maneira determinística. Nossa abordagem utiliza um algoritmo rápido e robusto para a segmentação de pontos aproximadamente coplanares. Após isso, votos são depositados no acumulador utilizando uma distribuição Gaussiana que modela a incerteza com respeito a posição e orientação de cada agrupamento. Nossos experimentos mostraram que nossa técnica é algumas ordem de magnitude mais rápida que técnicas concorrentes e escala melhor com o aumento do conjunto de dados.