Universitat de Girona

TOOLS FOR 3D POINT CLOUD REGISTRATION

Ferran Roure Garcia

Universitat de Girona

Tools for 3D Point Cloud Registration

By

FERRAN ROURE GARCIA

DOCTORAL THESIS

Doctoral Program in Technology
Supervised by:
Joaquim Salvi and Yago Diez

Thesis submitted to University of Girona in fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

2017

A la meva família i amics,
i sobretot, a la Mare.
"Qui no es cansa, bé que alcança".

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

Ferran Roure

Dedication and acknowledgements

I would like to express my gratitude to the people who have supported me during the elaboration of this PhD thesis. First and foremost to my advisors Quim Salvi and Yago Diez, for believing in me and encouraging me during these years. I would also to thank Xavi Lladó for his selfless dedication and help.

I also thank all the members of ViCOROB, especially Josep Q., Ricard C., Ricard P., Habib, Sandra, Eloy R., Marc M., Mariano, Ferran, Pablo, Albert, Sergi, Arnau, Konstantin, Oliver, Josep F., Xevi C., Jordi F., Robert, Aina, Mireia, Joseta and Anna.

To all the co-authors of the publications for their collaboration.

To all the UCL team and especially Prof. Niloy Mitra for their kind welcome during my stay in London.

A tots els amics de la colla, de Germà Negre i de sempre, per haver-me suportat i encoratjat tot aquest temps.

També vull agrair moltíssim a la meva família tot el suport que m'han donat: Padrí, Tia Teresa, Tia Sussi, Neus, Nuri, Josep, Toni, Joel i Saio.

Gràcies Pare i Eloi per esperonar-me sempre.

Gràcies Caterina per ser tant pacient amb mi i recolzar-me cada dia.

I moltes gràcies Mare, perquè tot i no ser aquí, sé que n'estàs orgullosa.

Publications derived from this thesis

The work developed in this thesis led to the following publications:

- Diez, Y., **Roure, F.**, Lladó, X., and Salvi, J. "A Qualitative Review on 3D Coarse Registration Methods". *ACM Computing Surveys*, 47(3), p. 45. 2015 (Chapter 1 and 2)
- **Roure, F.**, Diez, Y., Lladó, X., Forest, J., Pribanic, T. and Salvi, J. "An Experimental Benchmark for Point Set Coarse Matching". In *10th International Conference on Computer Vision Theory and Applications*, 2015. (Chapter 2 and 3)
- **Roure, F.**, Diez, Y., Lladó, X., Forest, J., Pribanic, T. and Salvi, J. "A Study on the Robustness of Shape Descriptors to common scanning artifacts". In *14th IAPR International Conference on Machine Vision Applications* (pp. 522-525). 2015 (Chapter 2 and 3)
- **Roure, F.**, Lladó, X., Salvi, J. and Diez, Y. "Range Searching Data Structures for Point Cloud Matching". In *the 9th Annual Meeting of the Asian Association for Algorithms and Computation*, 2016 (Chapter 3)
- Pribanic, T., Diez, Y., **Roure, F.** and Salvi, J. "An efficient surface registration using smartphone". In *Machine Vision and Applications*, 27(4), pp.559-576. 2016 (Chapter 3)
- **Roure, F.**, Lladó, X., Pribanic, T., Salvi, J. and Diez, Y. "Hierarchical techniques to improve Hybrid Point Cloud Registration". In *12th International Conference on Computer Vision Theory and Applications*, 2017 (Chapter 3 and 4)

Publications under review

- **Roure, F.**, Lladó, X., Salvi, J. and Diez, Y. "GKDtree: a new hybrid data structure for range searching". (Under review). In *Transactions on Image Processing*, 2017 (Chapter 4)

CERTIFICATE OF THESIS DIRECTION

Dr. Joaquim Salvi from University of Girona and Dr. Yago Diez from Tohoku University

DECLARE:

That the work entitled *Tools for 3D Point Cloud Registration* presented by Ferran Roure Garcia to obtain the degree in Doctor of Philosophy has been developed under our supervision and fulfills the requirements to obtain the International Mention.

Therefore, in order to certify the aforesaid statement, we sign this document.

Girona, Gener 2017

Dr. Joaquim Salvi

Dr. Yago Diez

List of Tables

List of Figures

TABLE OF CONTENTS

ABSTRACT

The interest of digitization[1] of the real world has been growing over the years. Computers help us to make it possible and research in this field has intensified with the emergence of new technologies in the commercial world. Representing our environment in formats that computers can understand is an essential step in the development of this technology. One way to capture our world is using sensors that digitize the objects around us using mathematical representations. These sensors are responsible for translating the real world into a language that allows computers to represent this information, understand it and modify it. In this case, when we digitize an element, we get a finite representation, which translates to points located in a virtual 3D space. These point clouds are the interpretation of the machines of our real world.

However, the sensors have limited vision —like a camera when capturing an image— and many times it is not possible to digitize everything we want at once. That is why we need to make more captures from different points of view. The problem is that we get different views of an object without knowing where they have been captured from and therefore, we get unconnected pieces that must be aligned to get the whole object. This alignment process is called *3D registration*.

Thus, 3D registration becomes a very important part, especially in the reconstruction of environments and objects. Currently, registration methods are used in a variety of applications, such as medical imaging or heritage preservation, as well as in many types of industrial processes or to scan objects in order to modify it and print it.

In this thesis, we did an in-depth review of the state of the art of 3D registration, evaluating the most popular methods. Given the lack of standardization in the literature, we also proposed a nomenclature and a classification to unify the evaluation systems and to be able to compare the different algorithms under the same criteria.

The major contribution of the thesis is the Registration Toolbox, which consists of software and a database of 3D models. The software presented here consists of a 3D Registration Pipeline written in C ++ that allows researchers to try different methods, as well as add new ones and compare them. In this Pipeline, we not only implemented the most popular methods of literature, but we also added three new methods that contribute to improving the state of the art. On the other hand, the database provides different 3D models to be able to carry out the tests to validate the performances of the methods. Finally, we presented a new hybrid data structure specially

[1]Here we consider "digitization" in the broadest sense of the word, defined as the conversion of text, pictures, sound or other real information, into a digital form that can be processed by a computer.

focused on the search for neighbors. We tested our proposal together with other data structures and we obtained very satisfactory results, overcoming in many cases the best current alternatives. All tested structures are also available in our Pipeline.

This Toolbox is intended to be a useful tool for the whole community and is available to researchers under a Creative Commons license.

Resum

La digitalització del món real és una necessitat que ha crescut amb els anys. Els ordinadors ens ajuden a fer-ho possible i la recerca en aquest camp s'ha intensificat amb l'aparició de noves tecnologies en el món comercial. Representar el nostre entorn en formats que els ordinadors puguin entendre és un pas imprescindible en el desenvolupament d'aquesta tecnologia. Una manera de capturar el nostre món és utilitzar sensors que digitalitzen els objectes que ens envolten fent servir representacions matemàtiques. Aquests sensors són els encarregats de traduir el món real en un llenguatge que permet als ordinadors representar aquesta informació, entendre-la i modificar-la. En aquest cas, quan digitalitzem un element, obtenim una representació finita, que es tradueix en punts ubicats en un espai 3D virtual. Aquests núvols de punts són la interpretació que fan les màquines del nostre món real.

Tot i això, els sensors tenen una visió limitada —com una càmera de fotos quan captura una imatge— i molts cops no és possible digitalitzar tot el que volem d'una sola vegada. Per això, ens cal fer més captures des de diferents punts de vista. El problema és que obtenim diferents vistes d'un objecte sense saber des d'on han estat capturades i, per tant, obtenim trossos inconnexos que han de ser alineats per tal d'aconseguir l'objecte complet. Aquest procés d'alineament s'anomena *registre 3D*.

Així doncs, el registre 3D esdevé un part molt important, sobretot en reconstrucció d'entorns i objectes. Actualment els mètodes de registre es fan servir en aplicacions molt diverses, com ara en imatge mèdica o en conservació del patrimoni, així com en processos industrials de tot tipus o en l'escaneig d'objectes per poder-los modificar i imprimir posteriorment.

En aquesta tesi, hem fet una revisió en profunditat de l'estat de l'art del registre 3D, avaluant els mètodes més populars. Donada la falta d'estandardització de la literatura, també hem proposat una nomenclatura i una classificació per tal d'unificar els sistemes d'avaluació i poder comparar els diferents algorismes sota els mateixos criteris.

La contribució més gran de la tesi és el Toolbox de Registre, que consisteix en un software i una base de dades de models 3D. El software presentat aquí consisteix en una Pipeline de registre 3D escrit en C++ que permet als investigadors provar diferents mètodes, així com afegir-n'hi de nous i comparar-los. En aquesta Pipeline, no només hem implementat els mètodes més populars de la literatura, sinó que també hem afegit tres mètodes nous que contribueixen a millorar l'estat de l'art de la tecnologia. D'altra banda, la base de dades proporciona una sèrie de models 3D per poder dur a terme les proves necessàries per validar el bon funcionament dels mètodes.

Finalment, també hem presentat una nova estructura de dades híbrida especialment enfocada a la cerca de veïns. Hem testejat la nostra proposta conjuntament amb altres estructures de dades i hem obtingut resultats molt satisfactoris, superant en molts casos les millors alternatives actuals. Totes les estructures testejades estan també disponibles al nostre Pipeline.

Aquesta Toolbox està pensada per ésser una eina útil per tota la comunitat i està a disposició dels investigadors sota llicència Creative-Commons.

Resumen

La digitalización del mundo real es una necesidad que ha crecido con los años. Los ordenadores nos ayudan a hacerlo posible y la investigación en este campo se ha intensificado con la aparición de nuevas tecnologías en el mundo comercial. Representar nuestro entorno en formatos que los ordenadores puedan entender es un paso imprescindible en el desarrollo de esta tecnología. Una manera de capturar nuestro mundo es utilizar sensores que digitalizan los objetos que nos rodean utilizando representaciones matemáticas. Estos sensores son los encargados de traducir el mundo real en un lenguaje que permite a los ordenadores representar esta información, entenderla y modificarla. En este caso, cuando digitalizamos un elemento, obtenemos una representación finita, que se traduce en puntos ubicados en un espacio 3D virtual. Estas nubes de puntos son la interpretación que hacen las máquinas de nuestro mundo real.

Sin embargo, los sensores tienen una visión limitada —como una cámara de fotos cuando captura una imagen— y muchas veces no es posible digitalizar todo lo que queremos de una sola vez. Por eso, necesitamos hacer más capturas desde diferentes puntos de vista. El problema es que obtenemos diferentes vistas de un objeto sin saber desde donde han sido capturadas y, por lo tanto, obtenemos trozos inconexos que deben ser alineados para conseguir el objeto completo. Este proceso de alineamiento se llama *registro 3D*

Así pues, el registro 3D se convierte en un parte muy importante, sobre todo en reconstrucción de entornos y objetos. Actualmente los métodos de registro se utilizan en aplicaciones muy diversas, como en imagen médica o en conservación del patrimonio, así como en procesos industriales de todo tipo o en el escaneo de objetos para poder modificar e imprimir posteriormente.

En esta tesis, hemos hecho una revisión en profundidad del estado del arte del registro 3D, evaluando los métodos más populares. Dada la falta de estandarización de la literatura, también hemos propuesto una nomenclatura y una clasificación para unificar los sistemas de evaluación y poder comparar los diferentes algoritmos bajo los mismos criterios.

La mayor contribución de la tesis es el Toolbox de Registro, que consiste en un software y una base de datos de modelos 3D. El software presentado aquí consiste en una Pipeline de registro 3D escrito en C ++ que permite a los investigadores probar diferentes métodos, así como añadir otros nuevos y compararlos. En esta Pipeline, no sólo hemos implementado los métodos más populares de la literatura, sino que también hemos añadido tres métodos nuevos que contribuyen a mejorar el estado del arte de la tecnología. Por otra parte, la base de datos proporciona una serie de modelos 3D para poder llevar a cabo las pruebas necesarias para validar

el buen funcionamiento de los métodos. Finalmente, también hemos presentado una nueva estructura de datos híbrida especialmente enfocada a la búsqueda de vecinos. Hemos testeado nuestra propuesta conjuntamente con otras estructuras de datos y hemos obtenido resultados muy satisfactorios, superando en muchos casos las mejores alternativas actuales. Todas las estructuras testeadas están también disponibles en nuestro Pipeline.

Esta Toolbox está pensada para ser una herramienta útil para toda la comunidad y está a disposición de los investigadores bajo licencia Creative-Commons.

CHAPTER 1

INTRODUCTION

3D Registration[1] represents a fundamental problem in a variety of areas such as medical imaging [48], environment reconstruction [44], shape retrieval [73] and industrial applications [79]. Specific issues include alignment of temporal 3D images for lesion monitoring, modeling of structures and the reconstruction of an object from several views. More recently, interest for virtual and augmented reality applications, where 3D reconstruction methods play an important role, is growing rapidly due to the new products recently presented like Microsoft HoloLens [13] or Oculus Rift [14]. Also, the growing research field of autonomous cars, where the passengers safety depends on part on the environment reconstruction capabilities, is becoming more popular. The research in 3D Registration is an important part of all these technologies and tools to improve them.

The goal of this thesis is to explore and unify the overall research field of 3D Registration and to provide tools that are useful for the research community, as well as to propose new methods to advance the state of the art. The scope of our work is focused on rigid alignment of 3D geometry. We tackled this problem from the algorithmic point of view, without restrictions in the acquisition protocol. For our system, every input data set is equally considered and only geometric information (and also other features like color, etc..) is taken into account for the solution. For this reason, we consider that there are no restrictions affecting the input data sets, beyond the complexity of its geometry.

In this chapter we introduce the 3D Registration topic and we propose a pipelined classification for the methods involved in the registration process. Our aims are to relate divergent notations addressing similar issues, review the most popular methods for each application area and classify them according to the aspects of the matching process they deal with.

[1] Note that we understand the words "registration", "matching" and "alignment" as synonyms, and we use them interchangeably throughout the thesis.

1.1 3D Registration

Registration methods work with different types of input data that we categorize as: 1) synthetic data (totally computer-made), 2) processed data (filtered and modified scanned data) and 3) real data (scanned data without any modification). Current methods of data acquisition (scanners, structured light, etc.) are able to provide huge amounts of data corresponding to very precise reconstructions. Depth cameras are an emerging acquisition technology, a representative example being Microsoft Kinect [76], which is gaining popularity because it provides good performance at reasonable prices. However, in these cases, the raw depth data must be processed in order to obtain the geometric primitives used in registration algorithms [67]. The scanned information obtained from any type of scanner, can be handled using a variety of these geometric primitives. Point clouds are the simplest representation and are widely used in the literature. However, many methods need more complex structures, such as triangular meshes.

The size of the input data used in most specific applications makes the development of efficient algorithms a key issue. For example, in object reconstruction, the most popular strategy is still to get many different views of the model and subsequently register them onto a common coordinate system. 3D registration allows for full model reconstruction but, if a high degree of precision is required, e.g. a huge number of points in the cloud, the process requires highly efficient methods to achieve registration in a reasonable amount of time. Although matching algorithms have seen many improvements over recent years, there is still no algorithm that can be considered standard in the sense that it can be used reliably in all situations and with the desired data sizes.

Due to the high number of application fields of registration, different scientific communities produce contributions related to it. These communities include Computer Graphics (Eurographics Conference[2], SIGGRAPH Conference[3]), Computational Geometry (SoCG conference[4], JCG journal[5]) and Computer Vision (Pattern Recognition[6], IJVC[7], PAMI[8]), to name a few. This dispersion of contributions makes the organization of information more difficult. The main problems are the lack of a common notation, the diversity of interests when approaching similar problems and the lack of common evaluation criteria.

Although we focus on 3D rigid registration methods, many other registration-related problems exist such as non-rigid alignment [63, 70], shape morphing [19], deformation transfer, self-similarity detection or time-varying surface reconstruction. For further details on these areas, we recommend two qualitative reviews: [118] and [113].

[2] http://www.eg.org/
[3] http://www.siggraph.org/
[4] http://www.uniriotec.br/~socg2013/
[5] http://jocg.org/index.php/jocg
[6] http://www.journals.elsevier.com/pattern-recognition/
[7] http://link.springer.com/journal/11263
[8] http://www.computer.org/portal/web/tpami

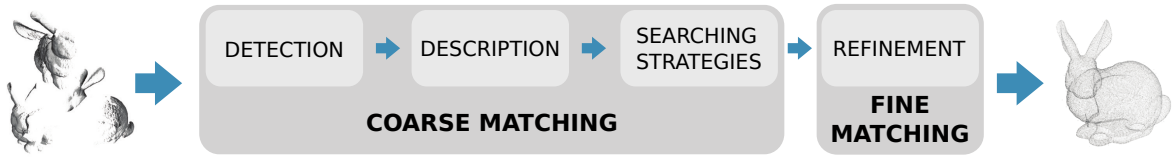

Figure 1.1: Point Cloud Registration Pipeline

1.2 Overview of the registration problem

The registration process consists of several steps. We propose a pipelined classification (Figure 1.1), to organize existing techniques in each step of the registration process. Methods are divided into two different categories: **Coarse** and **Fine** Matching. Coarse Matching, on which we focus in this thesis, encompasses all techniques that return a rough initial alignment of the input point clouds placed anywhere, without any initial alignment. In the literature we can find these kinds of methods under different terms such as *Rough* or *Coarse* alignment, or *Global* or *Crude* registration. On the other hand, Fine Matching includes methods that start from one such approximation and aim at finding a registration as accurate as possible. Coarse Matching can be further divided into three different steps: **Detection**, **Description** and **Searching Strategies**. As we will see throughout this thesis, most approaches are focused only on one part of the Pipeline. Most frequently, this part is the detection or description step. In most cases, the rest of the Pipeline is completed using very basic methods or even brute force.

Although the distinction is clearly defined between Coarse and Fine Matching, some methods within Coarse Matching are difficult to categorize. There are methods encompassing different steps, such as Detection and Description or Description and Searching Strategies.

Thus, our pipelined classification is structured as follows: first of all, a Detection step is used to reduce the number of points being considered. It consists in detecting a certain number of key-points that are prominent according to a specific criterion. The sizes of input data make the Detection step necessary in many approaches, in order to obtain computationally manageable datasets. The second step of the Pipeline is called Description and consists in assigning values to the detected key-points according to the properties of the shape around them. The functions that perform this are called Local Shape Descriptors. Finally, Searching Strategies are used in order to find correspondences between points in the two point sets. A correspondence between two points from different point clouds reinforces the assumption that these two points will be the same in the final registered shape. Descriptor values are used to prioritize the best apparent correspondences. A minimum of three correspondences are needed to determine the coarse alignment in 3D. The goal in this case is to avoid exhaustive search of the whole correspondence space. This exhaustive search would lead to asymptotic costs of $O(n^6)$ for 3D registration (n being the number of points of the sets), because corresponding triplets that determine the movement are found by checking

all point combinations from both shapes. After achieving coarse alignment, a Refinement step is applied. This step consist in using iterative methods in order to align the shapes as accurately as possible. These methods are usually very fast, but cannot be used unless a rough initial alignment is available.

1.3 Problem definition

The variety of registration applications produces divergent notations throughout the literature. Often similar notions receive different names and in some cases formal definitions of commonly used concepts are not widely available. In order to improve the readability of this review and unify related concepts in this section we introduce a formal definition of the problem.

1.3.1 Input data

Each of the many applications that use registration techniques has its preferred data type. Essentially, there are three types of input data used in the literature: point clouds, triangular meshes and volumetric data. The simplest is the former, which is a collection of 3D points with no other information. The second is composed of a point cloud and connectivity information between points, usually presented as a graph. The most commonly used format is a triangular mesh, i.e. a Delaunay mesh. Volumetric data is often used in medical imaging (MRI, Tomography,, etc...) due to the nature of acquisition. These types of data are considered to be easily processed in parallel. In this thesis, we focus on point clouds and meshes.

The input data of the registration problem consists, thus, of two point clouds $\mathcal{A}$ and $\mathcal{B}$, being $\mathcal{A} = \{a_1, ..., a_n\}$ with $a_i = (x_{a_i}, y_{a_i}, z_{a_i})$ and $\mathcal{B} = \{b_1, ..., b_m\}$ with $b_i = (x_{b_i}, y_{b_i}, z_{b_i})$. Note how, in some cases, such as the reconstruction of an object from several views, more than two objects might be involved in the registration problem. As these problems can be reduced to a series of pairwise registration instances, we do not explicitly include these problems in this section. Whenever the problem requires the use of meshes, we name them $M_{\mathcal{A}}$ and $M_{\mathcal{B}}$. These meshes are graphs $M_{\mathcal{A}} = (\mathcal{A}, E_{M_{\mathcal{A}}})$, $M_{\mathcal{B}} = (\mathcal{B}, E_{M_{\mathcal{B}}})$ where $E_{M_{\mathcal{A}}}, E_{M_{\mathcal{B}}}$ (graph edges) contains relationship information between the points of the object (graph vertices).

When it comes to the use of meshes, nearly all functions aiming at describing shape in the vicinity of a point a_i are based on the *neighbors* of a_i. These neighbors might be defined in terms of Euclidean distance, requiring a range searching data structures for their computation, or in terms of a mesh, where the neighbors of a_i correspond to its adjacent points, connected by $E_{M_{\mathcal{A}}}$. Note how either notions do not always coincide, especially as meshes might be constructed using a variety of criteria. Nevertheless, meshes are used often as they provide fast and convenient access to neighbors.

1.3.2 Desired output

The registration problem aims at finding a rigid transformation[9] $\mu : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ that brings set $\mathcal{A}$ as close as possible to set $\mathcal{B}$ in terms of a designated set distance. A commonly used distance is the root mean squared distance (RMSD) defined as follows:

$$\text{RMSD}(X,Y) = \sqrt{\frac{\sum_i^n |x_i - y_i|^2}{n}} \tag{1.1}$$

In applications where only partial matches are expected, it is sometimes desirable to fix an upper threshold thr for the distance between a_i and $\mu(b_j)$ so that points without correspondences do not influence the measure. The matching process returns a set of correspondences C between $\mathcal{A}$ and $\mathcal{B}$ where correspondences with distance farther than thr are discarded. We define C as an overlapping region (or "overlap") of both point sets:

$$\begin{aligned} C = \{&(a_i, b_j) \text{ with } a_i \in \mathcal{A}, b_j \in \mathcal{B} \\ &\text{such that } \forall b_k \in \mathcal{B}\ d(a_i, \mu(b_k)) \geq d(a_i, \mu(b_j)) \text{ and } d(a_i, \mu(b_j)) < thr\} \end{aligned} \tag{1.2}$$

Then we redefine the RMSD as:

$$\text{RMSD}(\mathcal{A}, \mu(\mathcal{B})) : a_i, b_j \in C = \sqrt{\frac{\sum\limits_C d(a_i, \mu(b_j))^2}{|C|}} \tag{1.3}$$

where $\mu(b_j)$ is the nearest neighbor (closest point) of a_i and $|C|$ is the cardinality of set C.

1.3.3 Detectors and Descriptors

Two very important steps of the registration problem are *detection* and *description*. In the first, the goal is to select those points of the sets that are more distinctive according to a chosen criterion (in most cases, the shape of the object). Besides, descriptors aim at encoding the shape around a point in terms of a set of numerical values. Consequently, although detectors and descriptors are focused on different targets, both are based on the same key-issue: the local shape of the input data. The key-points detected from an input data are selected according to the salience and uniqueness of the descriptor value at these points. We present both steps in separate sections (2.1 and 2.2) as most papers focus only on one aspect. In this section we highlight that both topics are very close, because both are based on the study of the shape around a certain point.

We define this *Shape Function* of a certain point a_i as $f^D(a_i) : N_{a_i} \subset \mathbb{R}^3 \longrightarrow P(\mathbb{R})$ being N_{a_i} the neighborhood of a_i, where the superscript D identifies the method. $P(\mathbb{R})$ is the power set of

[9] Such that $d(a_i, b_j) = d(\mu(a_i), \mu(b_j)) \forall a_i, b_j \in \mathbb{R}^3$, $d()$ being the euclidean distance.

$\mathbb{R}$ (e.g. the set of all subsets of $\mathbb{R}$). For each point a_k in N_{a_i}, $f^D(a_i)$ outputs a set of real values corresponding to the shape of N_{a_i} around a_i. Usually the same descriptor function is used for the two sets involved in the matching of $\mathcal{A}$ and $\mathcal{B}$. In order to avoid some cumbersome notation, from now on we will obviate this particular set N_{a_i} and refer to these functions as $f^D(a_i) : \mathbb{R}^3 \longrightarrow P(\mathbb{R})$. Some examples of these functions are:

- $f^{\mathrm{PS}}(a_i) : \mathbb{R}^3 \longrightarrow \mathbb{R}^2 \times \ldots \times \mathbb{R}^2$ for Point Signature descriptor [37], where $f^{\mathrm{PS}}(a_i)$ is a list of paired values for each point a_i in $\mathcal{A}$.
- $f^{\mathrm{SI}}(a_i) : \mathbb{R}^3 \longrightarrow$ *2D Histogram*, for Spin Image descriptor [65], where $f^{\mathrm{SI}}(a_i)$ is a distribution histogram of the points in the neighborhood of a_i.
- $f^{\mathrm{HKS}}(a_i) : \mathbb{R}^3 \longrightarrow \mathbb{R}$ for Heat Kernel Signature descriptor [110], where $f^{\mathrm{HKS}}(a_i)$ is the value of the heat diffusion function around a_i.

In the detection step, the most distinctive points are selected according to f^D. Often a threshold is used for this task, and only the points that satisfy this threshold are kept. We define this subset of selected points as $S_{\mathcal{A}} \subset \mathcal{A}$ that will be used in the rest of the Pipeline. In the description step we use f^D to obtain a value that represents the shape around the point. We define a Boolean correspondence function $cf^D : S_{\mathcal{A}} \times S_{\mathcal{B}} \longrightarrow \{0,1\}$ that checks whether or not the descriptor values at two given points are close enough for the shapes around the points to be considered to be the same:

$$cf^D(a_i, b_j) = \begin{cases} \text{TRUE} & \text{if } f^D(a_i) \approx f^D(b_j) \\ \text{FALSE} & \text{if } f^D(a_i) \neq f^D(b_j) \end{cases} \tag{1.4}$$

As an example, we present the correspondence function cf of Spin Image descriptor:

$$cf^{SI}(a_i, b_j) = \begin{cases} \text{TRUE} & \text{if } ||\alpha_{a_i} - \alpha_{b_j}|| < \epsilon \quad \text{and} \quad ||\beta_{a_i} - \beta_{b_j}|| < \epsilon \\ \text{FALSE} & \text{otherwise} \end{cases} \tag{1.5}$$

where $f^{SI}(a_i) = (\alpha_{a_i}, \beta_{a_i})$ and $f^{SI}(b_j) = (\alpha_{b_j}, \beta_{b_j})$. α and β are the projection over x and y axis of each point.

1.3.4 The computation of output motions

Throughout the literature, all motions μ considered as candidate outputs for the registration problem are computed using a number of point correspondences. First, points in sets $\mathcal{A}$ and $\mathcal{B}$ (a_i, b_j) are identified as possibly corresponding points. If a descriptor is being used, (a_i, b_j) must hold $cf^D(a_i, b_j) = \text{TRUE}$. Then, once a number of these "corresponding couples" have been identified, a motion is computed following specific criteria. Usually, the criterion used is the

least squares distances between sets. In a typical scenario [47], three corresponding couples: $(a_{i_1}, b_{j_1}), (a_{i_2}, b_{j_2}), (a_{i_3}, b_{j_3})$ are identified and μ is usually the rigid transformation holding that the Root Mean Squared Distance (RMSD) between sets $\{a_{i_1}, a_{i_2}, a_{i_3}\}$ and $\{\mu(b_{j_1}), \mu(b_{j_2}), \mu(b_{j_3})\}$ is the minimum possible. In order to determine a 3D rigid transformation at least three point correspondences are mandatory (although more might be used [17, 120]). The number of point correspondences varies for other types of motions, e.g. only one point correspondence is needed to determine a 3D translation.

These sets of points used to compute candidate motions are a commonly used concept. Henceforth, we will refer to this concept as a *base*.

A *base* $B_{\mathcal{A}} = \{a_{i_1}, ..., a_{i_k}\} \subset S_{\mathcal{A}}$ of set $\mathcal{A}$ stands for the set of k points used to determine a rigid transformation which is a candidate to be the output of the registration problem. Each point of $B_{\mathcal{A}}$ must have a correspondence point in an analogous $B_{\mathcal{B}} = \{b_{j_1}, ..., b_{j_k}\} \subset S_{\mathcal{B}}$ such that $cf^D(a_{i_l}, b_{i_l}) = \text{TRUE}$.

Note how, although only few points from each set are usually considered when computing candidate motions, the measure of how close the two sets are is computed using all the points in the sets or, at least, all the matched points.

1.3.5 Problem statement

A formal summary of this section follows:

- Given two point sets $\mathcal{A} = \{a_1, ..., a_n\}$ and $\mathcal{B} = \{b_1, ..., b_m\}$ and a shape function $f^D : \mathbb{R}^3 \longrightarrow P(\mathbb{R})$ with correspondence function $cf^D : S_{\mathcal{A}} \times S_{\mathcal{B}} \longrightarrow \{0, 1\}$.
- Let $S_{\mathcal{A}} \subset \mathcal{A}$ and $S_{\mathcal{B}} \subset \mathcal{B}$ the points that are most distinctive in term of a shape descriptor function $f^D : \mathbb{R}^3 \longrightarrow P(\mathbb{R})$
- A solution to the rigid registration problem is a rigid transformation μ such that:
  - RMSD($\mathcal{A}$, $\mu(\mathcal{B})$) is minimum.
  - There exist two bases $B_{\mathcal{A}} = \{a_{i_1}, ..., a_{i_k}\} \subset S_{\mathcal{A}}$, $B_{\mathcal{B}} = \{b_{j_1}, ..., b_{j_k}\} \subset S_{\mathcal{B}}$ holding that for all corresponding couples, $cf^D(a_{i_l}, b_{j_l}) = \text{TRUE}$.

1.4 Goals of the thesis

While reviewing the literature we realized that the area had serious limitations in terms of unified notation and result reproducibility. Specifically, the problem was approached from different perspectives using different notations and most papers used particular data sets that where not available to the general public. Although papers are regularly published, code is often not accessible and comparison of method performance is infrequent. Consequently, this thesis aims at achieving three main goals:

The first goal is to study in depth the problem of coarse point cloud matching. We aim at structuring a matching Pipeline so all existing algorithms related to the problem can be classified and categorized. We also intend to define a standard notation to describe the magnitudes used and the results obtained by these algorithms. Furthermore, we aim at studying the behavior of state-of-the-art methods in several steps of the Pipeline in order to obtain a realistic overview of the available technology.

The second goal of this thesis is to advance towards overcoming the limitations existing in the area regarding dataset and algorithm availability. Consequently, we aim at developing a 3D Registration Toolbox which provides a useful tool for researchers in the area. The toolbox includes datasets with information on correct final alignment results as well as information used by some of the algorithms in the intermediate steps of the Pipeline. It also includes all the algorithms studied throughout the thesis. Users of the toolbox should be able to run the included methods and use the data, but they could also take advantage of it in their own research. Specifically, they should be able to run the included methods using their own data and, more importantly, insert their own proposed methods in the corresponding step of the Pipeline. This would allow them to develop algorithms that improve just one step of the Pipeline while still using state-of-the art approaches for the rest of the steps without having to implement all methods from scratch.

Finally, our last objective consist of creating a new data structure for residue computation called GridDS. All experiments performed in this thesis gave us a detailed knowledge of the Registration Pipeline performance. Computing a residue between two aligned views is costly but also indispensable. The mainstream data structures achieve fast runtimes but dedicated structures are preferable. We tested the most popular data structures for residue computation and we presented a new hybrid data structure which outperforms all other.

1.5 Thesis outline

The organization of this thesis follows the aforementioned objectives. First, Chapter 1 introduces the 3D registration topic. In Chapter 2 we evaluate the state-of-the-art with a qualitative review of each method and we provided our conclusions about them. In Chapter 3 we present our 3D Registration Toolbox that we developed from the scratch. In Chapter 4, where we present a new hybrid data structure for residue computation. Finally, Chapter 5 contains the conclusions of this work.

CHAPTER 2

STATE OF THE ART

The literature of 3D Registration is extensive because a large variety of application problems can be found in different communities: aligning pairs of views from small objects, heritage reconstructions, object recognition or even machine learning problems. Our aim in this chapter is to review the state of the art and categorize each solution according to Pipeline shown in Section 1.2, which consist in four steps (see Figure 1.1): Detection, Description, Searching Strategies and Refinement. We analyze the most used methods and qualitatively evaluate their strengths and weaknesses. See Tables 2.1, 2.2 and 2.3 for an overview of reviewed methods.

2.1 Detectors

The registration of large point clouds is costly and impractical. In order to reduce both the computation time and the number of points to be considered, the most common strategy is to use only those points that can effectively contribute to finding a good enough solution. In other words, the goal is to obtain a subset of points that maintain as much as possible the shape of the object. In order to identify these relevant points, detection methods are used. Nowadays, this is a rapidly growing research field, motivated by 3D shape retrieval problems [31, 64, 74, 108, 114]. This step is also called *filtering* because the non-relevant points removal becomes a direct consequence of this detection step. Several criteria exist in order to decide which points should be kept and which points should be discarded. Note that many methods explained here are used in combination with a descriptor, usually presented under the same name. In this section we introduce the most remarkable methods in the literature according to the results presented in each paper and in different reviews and benchmarks [28, 29, 52, 103, 123].

2.1.1 Normal Space Sampling

[100] reviewed several methods, such as uniform [117] or random [82] sampling. The main problem with these methods is that the selection of points does not depend on surface characteristics. Dealing with smooth models with small irregularities (e.g. a plane), the process might result in sampling many points that essentially contain the same information in terms of normal vectors. For this reason, the authors introduced the Normal-Space Sampling (NSS) method. This strategy consists of 1) grouping points in "buckets" according to the angles between their normal vectors (considered in the unit sphere) and the coordinate axes, and 2) sampling uniformly over the resulting buckets, providing a downsampling of the points with more "frequent" normal vectors.

[47] presented an improvement of NSS called Hierarchical Normal-Space Sampling. This method groups points hierarchically, according to the distribution of their normal vectors with each level in the hierarchy representing a NSS instance. The search for correspondences then proceeds hierarchically between points of the two sets. The huge search space is navigated taking advantage of geometric information until a solution is found. The authors use a RANSAC-based method inside the hierarchical structure in order to find a transformation that roughly aligns the two point sets. A significant reduction in computation time is observed when compared to pure RANSAC-based methods.

Different versions of NSS can be implemented using other point characteristics. In our Toolbox (See Chapter 3) we implemented a Color Space Sampling detector, which uses the three RGB color components as attributes for the sampling. As in NSS, this method gives more importance to non-frequent parts of the model in terms of color grading.

2.1.2 Maximally Stable Volumes (MSV)

MSV [51] is a 3D extension of Maximally Stable Extremal Regions (MSER) [83]. MSV detects the most stable regions in a volume across different binary thresholds. Given a volumetric shape, the points inside the regions that remain visible under a set of binary thresholdings, will share good key-points for a registration process.

A 3D volume can be interpreted as a weighted graph. Each voxel of the volume represents one node in the graph and its value (e.g the intensity value of an MRI data) is the weight of this node. The connectivity between nodes is given by the spatial neighborhood of the voxels. A level set L_w of a weighted graph contains the set of nodes with a weight above a given threshold w. The connected nodes within the same level are grouped in connected components. In order to find a MSV, the authors propose the use of a rooted data structure namely a *component tree*. A component tree of a weighted graph is an ordered representation of the graph. The component tree of a 3D volume has connected volumes C_i^w as tree nodes. Each level of the component tree contains the connected volumes of a specific level set L_w at weight w. The MSVs are identified as the connected volumes with the highest stability along a thresholding process through all levels

of the component tree. There are different algorithms for computing the component tree, but the most efficient is the algorithm proposed by [87] that runs in quasi-linear time.

There are different options for the key-point selection, for example a random sampling of the surface of the MSV, or a selection of the center of the ellipsoid contained inside the MSV as a key-point.

In [123], MSV is tested against other detectors like Harris 3D, SURF or MeshDoG, obtaining the best performance results and being robust to noise and rotation. Compared with the other methods, MSV detects few key-points in the input surface but the ratio of correspondences between two registered shapes is, at least twice as big as the other detectors. The main drawback is the computation time as the search algorithm for stable regions is less efficient in 3D than in 2D.

2.1.3 Heat Kernel-based features

As we will see in Section 2.2, Heat Kernel Signature (HKS) is a point descriptor presented by [110]. However, the authors use the same concept in the mentioned reference: the heat diffusion in a surface over a temporal domain, as a key-point detector.

The nature of this method makes it possible to use the HKS as a shape function f^{HKS} in order to detect the parts of the shape with zones that are more salient in terms of descriptors, like zones with high curvature. High values of f^{HKS} identifies the key-points of a shape.

HKS is one of the best detectors in the literature due to its high repeatability results ($\approx 90\%$). HKS tends to return less key-points than others detectors, but with high distinctiveness. In the SHREC 2010 benchmark [29], HKS obtained the best results; and in [52], when it is compared with human-generated ground truth, it performs much better than the other methods, close, in fact, to human selected key-points. Additionally, it can also be used for non-rigid registration.

2.1.4 MeshDoG

[124] presented a point detector based on the Difference of Gaussians (DoG) operator. MeshDoG finds the extrema of the Laplacian of a scale-space representation of any scalar function defined on a discrete manifold.

Assuming a uniformly sampled triangulated mesh $M_{\mathcal{A}}$ as input data, the authors find the key points using the DoG operator. For each point $a_i \in M_{\mathcal{A}}$, the extrema of the Laplacian function are found across scales using a one-ring neighborhood N_{a_i}. Then, the feature points are selected as the maxima of the scale-space across scales. Finally, only 5% of feature points are selected in order to make this detection step more accurate. Only those feature points exhibiting corner characteristics are considered.

MeshDoG achieves high repeatability results ($\approx 85\%$) tested in [28], being robust to rigid transformation and scale modifications.

Figure 2.1: Three steps of MeshDoG algorithm applied on a (a) Mesh: (b) Scale-space extrema detection, (c) Thresholding and (d) Corner detection. Image taken from [124].

2.1.5 Intrinsic Shape Signatures

ISS [126] is a point descriptor that possesses its own detection method. As we will see in section 2.2, ISS uses the eigendecomposition of the neighborhood's covariance matrix of a point to describe it. These values are used to select the most representative points $S_{\mathcal{A}}$ and $S_{\mathcal{B}}$ from the point clouds. Key-points are selected as the points with large three dimensional variations in their neighborhood. These variations are measured using the smallest eigenvalue of the covariance matrix of its spherical neighborhood.

ISS demonstrates high repeatablity results ($\approx 70\%$) with processed data, even in noisy scenes, identifying few but strong key-points. However, with real data, these results drops to $\approx 30\%$.

Another detector called Key-Point Quality (KPQ) [85] is very similar to ISS. KPQ also uses the neighborhood's eigendecomposition of the covariance matrix to establish a Local Reference Frame (LRF). As a major difference from ISS, KPQ defines a key-point quality measure of each point based on the principal curvatures of the local surface within a neighborhood. A smoothed surface S is fitted over the original data. S is divided in a grid in order to sample the surface. Principal curvatures k_1 and k_2 and the Gaussian curvature $K = k_1 k_2$ of each sample of S are used to calculate the key-point quality. The method selects the key-points with high quality value and, with processed data, this method performs worse than ISS. It obtains better results, however, with real models.

2.1.6 Harris 3D

[109] presented a 3D version of the Harris operator. The idea is to apply the Harris Operator in a 2D projection of the points, without losing relevant information. In order to find the best projection, where the points exhibit a good spread, the authors translate and rotate the set of points of $\mathcal{A}$ according to the following criterion: for each point $a_i \in \mathcal{A}$, a neighborhood N_{a_i} is defined. The centroid of N_{a_i} is computed and all points in $\mathcal{A}$ are translated so that the centroid coincides with the origin of the coordinate system. Then, a fitting plane to the translated points is

computed. Authors apply Principal Component Analysis (PCA) to the set of points and choose the eigenvector with the lowest associated eigenvalue as the normal of the fitting plane. Afterwards, they rotate the set of points until the normal of the plane coincides with the z-axis. Finally, the resulting XY-plane (2D projection) is used to calculate the derivatives. These derivatives are computed using a six-term quadratic surface (paraboloid) fitted to the set of transformed points. The Harris operator value in the studied point is calculated with:

$$f^H(a_i) = \det(E) - k(\mathrm{tr}(E))^2 \tag{2.1}$$

where E is a matrix calculated from the points using the quadratic surface mentioned above. The vertices with highest Harris values are considered feature points, obtaining a constant number of vertices.

Due to the simplicity of the algorithm, Harris is faster. However, it is not robust to noise because the corner detector methods are sensible to the perturbations of the surface [29]. Although the method detects many key-points, the ratio of correspondences is small, around 20%, and decreases considerably when the noise increases.

2.2 Descriptors

The shape function f^D, also frequently referred to simply as the descriptor of a_i, can be defined as a set of values representing the shape characteristics of object $\mathcal{A}$ around a_i. A desirable characteristic for 3D rigid registration is for this representation to be invariant under translation, rotation and scaling.

In terms of the number of papers published, Descriptors are the most active research field in the Registration Pipeline. We classify many existing approaches according to certain common characteristics. Following the work of [116], we arrange the approaches in *Signatures* and *Histograms*. The former includes methods that offer a numerical result as a descriptor of a given point. The latter, compute a histogram.

Another distinguishing factor is the type of the input data. Most methods work with point clouds without any added structure ($\mathcal{A}$), but some methods need to produce richer representations, like meshes ($M_{\mathcal{A}}$). A triangulation with good shape properties, such as the Delaunay triangulation, where the distribution between vertexes, edges and faces is approximately regular, and comes with a high computational cost ($O(n^2)$). This cost, however, is incurred only once in a preprocessing step.

Another factor we use for the discussion in this section are reference frames. In papers like [116, 126], the authors note the importance of achieving a good Local Reference Frame (LRF) in order to improve the accuracy of the detectors/descriptors. This accuracy stems mainly from having an unambiguous local reference frame for every point allowing for detailed descriptions of local shapes. As we see in Section 2.5, one possible drawback of this approach lies in its sensibility

to noise, especially occlusions. These factors greatly perturb the local neighborhoods of points and, thus, affect the computations of LRF.

Finally, we classify the methods according to their geometrical or topological nature. Although topological methods are primarily used in non-rigid registration, they are also used in rigid.

In order to make this classification easier, we present the methods of this section with the following acronyms:

- **S / H**: Signature-based or Histogram-based method.
- **P / M**: Point cloud or Mesh, as an input data type.
- **LRF / nLRF**: A Reference Frame is used or not.
- **G / T**: Geometrical or Topological method.

As mentioned above, even though non-rigid registration methods are out of the scope of this review, we include some of them because they obtain successful results even in rigid registration.

2.2.1 Principal Curvature [S,P,nLRF,T]

Principal Curvature stands for the maximum and the minimum curvature of the surface at a given point. [54] proposed to use it as a descriptor. In this approach, key-points are described by the Principal Curvatures (k_1, k_2), the normal vector of the point $(\vec{n})$ and the principal directions $(\vec{e_1}, \vec{e_2})$ corresponding to the principal curvatures. To search for correspondences between two point clouds, the algorithm considers an initial point a_i on surface $\mathcal{A}$ with a descriptor $f_{a_i}^{PC} = (a_i, \vec{e_{1i}}, \vec{e_{2i}}, \vec{n_i})$, and a set of possible candidates on the second surface $\mathcal{B}$, each one with a descriptor $f_{b_j}^{PC} = (b_j, \vec{e_{1j}}, \vec{e_{2j}}, \vec{n_j})$, similar to $f_{a_i}^{PC}$. Two rigid transformations are defined, D and D', where D aligns $f_{a_i}^{PC}$ with $f_{b_j}^{PC}$ and D' aligns $f_{a_i}^{PC}$ with $f_{b_j}^{PC'} = (b_j, -\vec{e_{1j}}, -\vec{e_{2j}}, \vec{n_j})$. Note that both rigid transformations, D and D', are computed because there is no way to choose between them, due to the fact that the direction of $\vec{n_i}$ is ambiguous. Afterwards, the transformation matrix that aligns both views is computed and evaluated. The authors consider all correspondences between $\mathcal{A}$ and $\mathcal{B}$ at a distance smaller than a certain threshold. If not enough correspondences are found, the algorithm chooses another initial point a_i and iterates. Otherwise, the alignment is computed using the available correspondences.

The principal curvature of a certain point in a surface depends on the normal vector associated. Hence, the computation of these normal vectors plays an important role. As the value of a normal vector depends on the position of its neighbors [62] the presence of noise may cause a lack of accuracy in its computation. Although this problem is largely tackled in the community [86], the quality of the registration process is being influenced by the normal computation.

The Principal Curvature method is also used in non-rigid registration because surface curvatures are invariant to isometric deformations [54].

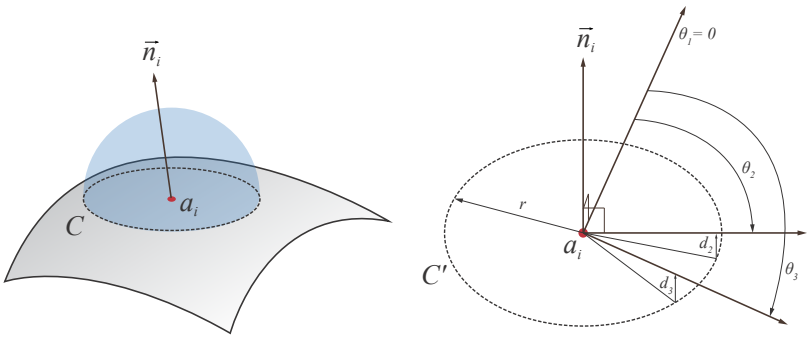

Figure 2.2: Representation of Point Signature. Left: Intersection between the surface $\mathcal{A}$ and the sphere centered at a_i, giving a curve C. Right: Distances and angles of different points in C projected to C'.

The main problem of this algorithm is that only one correspondence is used to compute the rigid transformation. As the algorithm stops when it finds a good correspondence, other possible correspondences might not be considered. Better alignments might be missed if the motion found is affected by noise or occlusion [104].

2.2.2 Point Signature [S,P,LRF,G]

Point Signature is a descriptor introduced by [37]. For a point a_i on a surface $\mathcal{A}$, a sphere of radius r centered at a_i is considered. The intersection between the surface $\mathcal{A}$ and the sphere determines a curve C. This curve is projected on a plane tangent at a_i and perpendicular to $\vec{n_i}$, giving a contour C'. Then, taking a_i as center of coordinates, the authors define an orientation axis with the normal vector $\vec{n_i}$, a reference vector $\vec{n_1}$ and the cross product between them. Each point in C will be described by a signed distance between itself and its projection on C', and the rotation angle from the reference vector n_1. The Point Signature of a_i will be expressed as the set of distances and angles of the points on C. To find correspondences between two point clouds, Point Signatures of all the points are compared following the example seen in Equation 1.5. Figure 2.2 shows a representation of the descriptor.

Although the matching process is fast, the cost of the intersection between the surface and the sphere requires the use of range searching data structures.

2.2.3 Spin Images [H,M,LRF,G]

In 1997, Johnson presented a descriptor based on the position of the neighbors of a given point in [65, 66]. The authors consider a point a_i and its associated normal vector $\vec{n_i}$. They define a plane P tangent to a_i and perpendicular to $\vec{n_i}$. The neighborhood N_{a_i} around a_i will be registered based on two variables: distance α between each point and the normal vector $\vec{n_i}$, and the distance

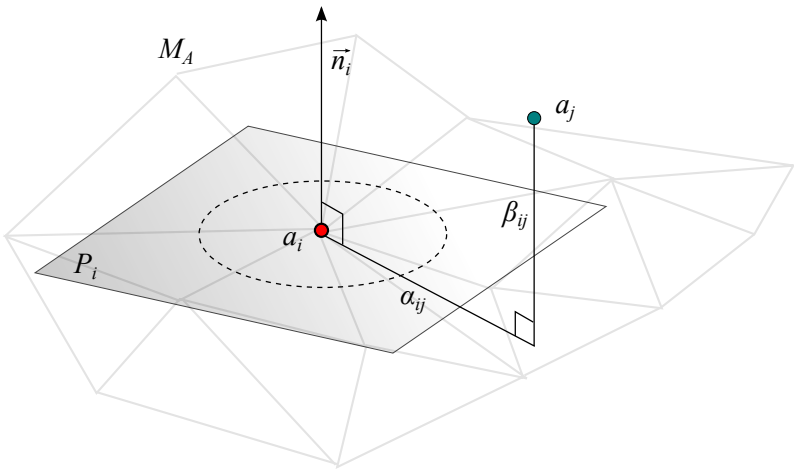

Figure 2.3: Spin Image representation: an oriented point basis created at a vertex in a surface mesh. The position of the oriented point is the 3D position of the vertex, and the direction of the oriented point is the surface normal at the vertex.

β between each point and the tangent plane P (see Figure 2.3). The following functions are used to calculate both parameters.

$$\alpha = \sqrt{||x - a_i||^2 - (\vec{n_i}(x - a_i))^2} \tag{2.2}$$

$$\beta = \vec{n_i}(x - a_i) \tag{2.3}$$

A table called Spin-Map is generated with this information, where each point x around a_i is projected according to α on the *x-axis* and β on the *y-axis*. Each cell of the Spin-Map contains the number of points belonging to the corresponding region. The generation of the shape function f^{SP} can be visualized as a rotating sweep over $\vec{n_i}$, where all the Spin-Maps are accumulated. Then, in order to find the correspondences between two different shapes, Spin-Images are compared counting the points falling in the corresponding bins of both Spin-Images.

This method is invariant to rigid transformations. It is, however, sensitive to symmetries and noise. Another problem is that the result of the method depends largely on the resolution used. [32] proposed an improvement called Face-based Spin Image to solve these problems, where the numbers of points in each Spin-Image are uniformly assigned.

Spin-Image is the base of numerous recent approaches. Two examples are Intrinsic Shape Signatures (ISS) [126] and SHOT [116]. Both methods stress the importance of choosing a good reference frame (RF). These RFs are chosen via eigendecomposition of the covariance matrix from neighboring points. The eigenvectors with higher eigenvalues are used as the axes of the RF. Then, the authors use this LRF to compute a version of Spin-Image. ISS makes an occupational

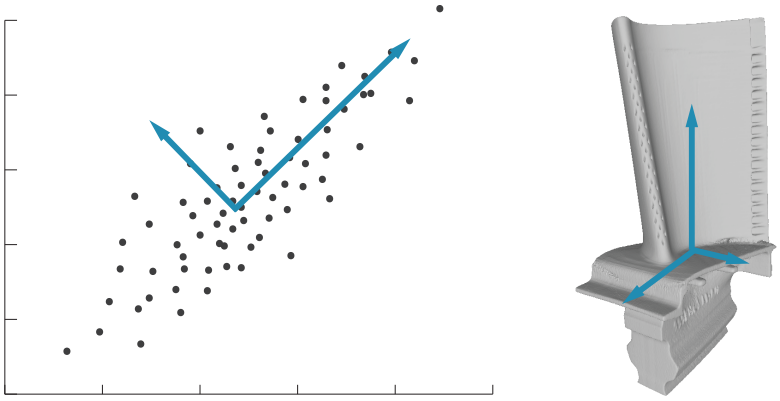

Figure 2.4: Examples of PCA in 2D (left) and 3D (right) point clouds.

histogram of points inside the supporting sphere neighborhood around the point. SHOT makes a histogram of differences between the point and the neighbors inside the supporting sphere. Although both methods obtain satisfactory results with processed data, as we see in [103], neither achieve sound results with real data.

[125] presented Improved Spin Image (ISI), using angle information between the normals of feature points and neighboring points. The β parameter is replaced by signed angles. This method can be explained as a distribution of the angles among different rings. The authors claim that their descriptor outperforms both the classic implementation of Spin Image and also the SHOT method.

Besides these approaches, other works following the path opened by Spin Image are: *Spherical Spin Image* [99] *Local Surface Patches* [35] or *Scale Invariant Spin Image* [43].

2.2.4 Principal Component Analysis [S,P,LRF,G]

Initially, the theoretical basis of this method was presented by [90] in order to transform a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. It was exported to other fields, such as statistics, computer vision or computational geometry. In point cloud registration, this method is used to find the principal axes that describe the shape of a point cloud (see Figure 2.4). Given two point clouds $\mathcal{A}$ and $\mathcal{B}$ from the same object, if the main axes are coincident, we can find a transformation that aligns both coordinate systems.

[38] presented a registration algorithm based on PCA, using the covariance matrix to determine the transformation μ between two point clouds. This method can also be considered a Searching Strategy, due to the global understanding of the algorithm, because it finds the principal component of all points in the point cloud. However, there are many other algorithms that implement Local PCA, and obtain principal components of local neighborhoods, considering

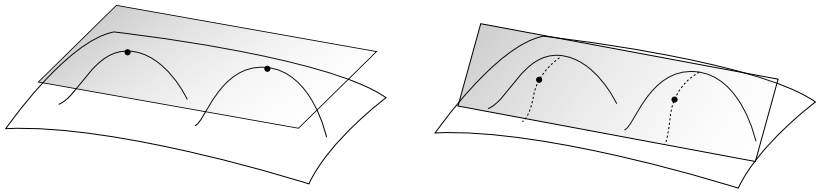

Figure 2.5: Representation of a bitangent plane from Line-based method, rolling over a surface and describing two bitangent curves. This picture is taken from [119].

it a point descriptor.

There are many approaches that use PCA: In [92, 122] PCA is used to find principal components of a local sphere neighborhood. In [43, 109] PCA is used as an interest point detector. [65] uses PCA to compress Spin-Images while [69] uses it to find the orientation of 3D shapes. [75] presented an improvement of PCA for rigid and non-rigid registration, robust to noise and outliers, using the least median of squares (LMS) techniques.

PCA is a very fast method, but it has some drawbacks that constrain its use in some practical applications. The algorithm needs large set overlap (≥50%) to find good correspondences and symmetries in the surface. Furthermore, the presence of noise in the original point cloud may influence the alignment [23].

2.2.5 Line-based algorithm [S,P,nLRF,G]

[119] presented a descriptor based on bitangent curves. The reason for using these types of curves instead of typical surface curvatures is that they are easier to calculate using dual space. Using two bitangent points lying in the same plane and rolling the plane over the surface, we obtain two bitangent curves which are used as a shape descriptor f^{LB} (See Figure 2.5).

The key to this method is that, in transforming the range images on dual space, the bitangent points of the surface are coincident. This transformation decreases the computing time and improves the robustness. The main problem of this method is that, in some cases, the number of bitangent curves may be insufficient for the achievement of a good registration result. Moreover, noise hampers the search for correspondences.

2.2.6 3D Shape Contexts [H,P,nLRF,G]

[69] presented a 3D extension of 2D Shape Contexts. This method consists of describing a certain point in relation to the other points in the object, and not only the points in the neighborhood around it. Due to the size of the data sets, this algorithm only uses random sampled points instead of the full-sized data. The complexity of this detection step is $O(Slog(n))$, being S the number of samples taken from n points.

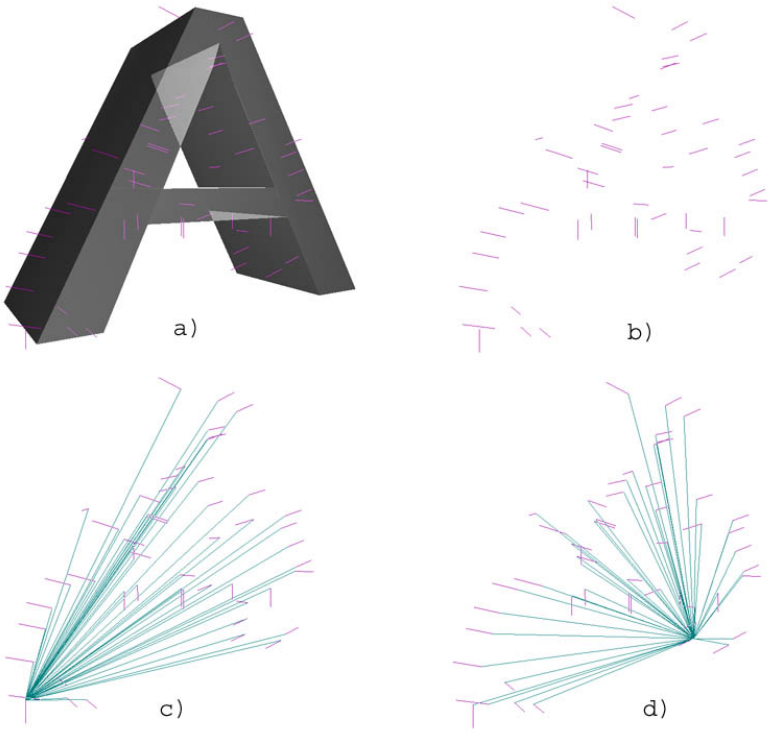

Figure 2.6: Example of 3D Shape Context taken from [69]. a) Mesh with 50 samples. b) Just the 50 samples. c) 49 Vectors originating from one sample point. d) 49 Vectors originating from another sample point.

Given a sampled point a_i, the method finds the vectors from a_i to all the other sampled points (see Figure 2.6). These vectors express the appearance of the entire shape, in relation to the reference point a_i. For each point on the sampled set, the shape function $f^{3\text{DSC}}$ is defined as a coarse histogram of the relative coordinates of the remaining $N-1$. To create this histogram, a sphere-space division is used, with center at a_i. This sphere is divided into bins. This space discretization is enough to obtain a robust descriptor.

The last part looks for correspondences. This is the most computationally expensive part of the algorithm. Two specific techniques are used in order to match the descriptors between two different shapes: Local Matching and Global Matching. The first one combines three concepts: Shape, Appearance and Position. The goal is to obtain an invariant shape descriptor for each sampled point. For Global Matching, two different strategies are used: Hard and Soft assignments. Hard assignments stand for a one-to-one searching method, and require high computational cost. However, this cost is reduced on Soft Assignments by carrying out a pre-selection of the candidates in the second shape for each point in the first shape. Using this approach the authors achieve a cost of $O(n_1 n_2)$ instead of $O(n^3)$ from Hard Assignments (n_1 and n_2 are the number of

samples for each shape).

The authors claim that the method is robust to noise, topological and geometrical artifacts and invariant under transformations. These claims are backed by an experimental study carried out using processed data. One question that remains is whether this method may have some problems with real scanned data, especially regarding shapes with a low ratio of overlapping. Questions naturally arise from the fact that the descriptor is computed using sampled points from the entire model.

2.2.7 Dynamical Systems [S,P,nLRF,T]

[45] presented a topological shape segmentation method called Dynamical Systems. Instead of focusing on local geometric properties of the shape, this approach identifies and segments the main sections of the shape from a global point of view. A single point represents an entire segment, where the weight of this point is the volume of the segment. First, given a set of points $\mathcal{A}$, the Voronoi diagram and the Delaunay triangulation are computed. Following the theory of the flow induced by a shape, critical points are selected. These points are defined to be the intersection points of the Voronoi objects with the Delaunay objects. For each critical point a_i, a stable manifold $S(a_i)$ is defined as the set of points that flow into a_i, grouping a set of Delaunay tetrahedra. So, the closure of these stable manifolds stands for the features of the shape. Then, the authors defined a *signature* as a set of features of $\mathcal{A}$. Each feature yields a representative point r, which is the weighted average of the centroids of all Delaunay tetrahedra from each feature. The weight of r is the volume of the feature. Finally, the matching process is performed by computing the similarity of the signatures of two shapes.

Few signatures are used to align two shapes. However, the cost of the algorithm is $\Theta(mn)$[1], which is a drawback if more signatures are needed. Furthermore, the authors did not test the method with real or noisy data.

2.2.8 Integral Invariants [S,M,LRF,G]

The descriptors based on differential geometry, like curvatures [50, 91], are not robust to noise and perturbations, and require data smoothing and prior de-noising in order to achieve better results. This is problematic when you are working with real data which usually contains noise and outliers. Integral Invariants produce a good solution for this problem, obtaining a descriptor based on the volume under the surface of an object. Given a point a_i on a surface $\mathcal{A}$, a sphere of radius r centered at a_i is computed. The method calculates the enclosed volume of the sphere $V_r(a_i)$ under the surface and the value of this volume stands for the Integral Invariant descriptor f^{II}. Figure 2.7 shows a 2D representation of the process. In order to find corresponding points, the value of f^{II} is used for comparison purposes.

[1]The cost is mn not only maximum but also minimum.

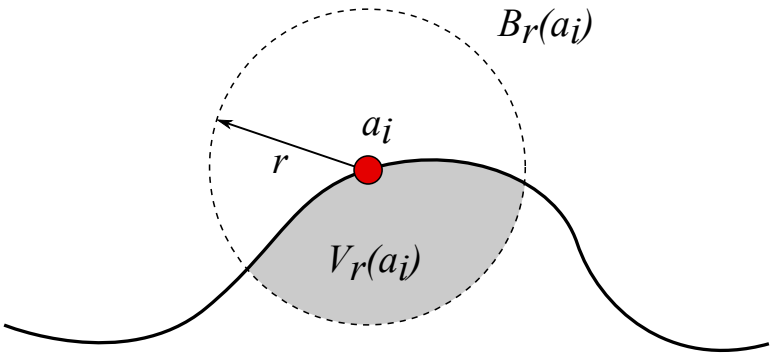

Figure 2.7: 2D representation of the Integral Invariant descriptor.

The first authors to study the applications of the volume descriptors were [78]. The authors claim that the numerical differentiation methods applied on point descriptors are sensitive to noise. Integral Invariant Signatures, presented in his paper, are robust to noise, including discretization artifacts, and present a multi-scale behavior. In [92] the authors presented a stability analysis of Integral Invariants based on distance functions. This method is based on Principal Component Analysis of local neighborhoods defined by kernel balls of various sizes. [122] presented an experimental paper about Integral Invariants obtained by integration over local neighborhoods. This short paper compares the method based on applying PCA over a ball or sphere neighborhoods from Pottman [92] with the *Normal Cycles* method [39] and the *Osculating Jet* method [33]. The authors conclude that Integral Invariants are more robust to noise than the other methods while exhibiting the desired scaling behaviour. However, the papers in question do not test the method with real data, without smoothing and de-noising. In this situation, with high amounts of perturbations, as well as with low overlapping regions and holes, Integral Invariants may not produce satisfactory results.

Pottman et. al. proposed three different ways to compute Integral Invariants: the Fast Fourier Transform-based method, the octree based method and the triangulation based method. All these methods generate running times of the same order.

2.2.9 Curve-skeleton [S,P,nLRF,T]

Curve-skeleton or skeletal graphs were introduced by [25]. This method consists in describing a shape by a thinned representation (see Figure 9). A skeleton (stick figure) of the shape is extracted and converted to a skeletal graph that preserves the topological properties of the shape. Then, the graphs of the two objects being matched are compared in order to register two different shapes. This method can be used either with point clouds, meshes or volumetric data.

More recently, [41] presented a thorough state-of-the-art example of curve-skeleton approaches. According to [41] curve skeleton methods have many useful properties for shape

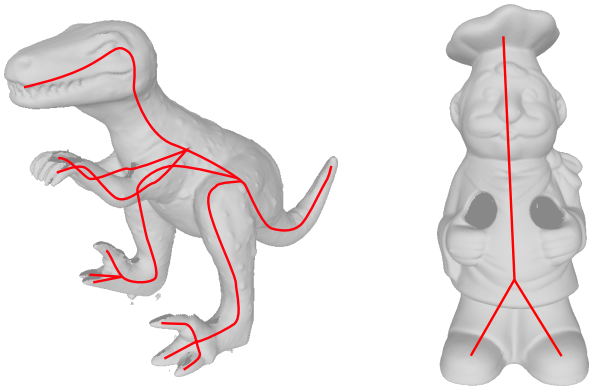

Figure 2.8: Curve-skeleton representation of T-Rex and Chef models.

registration, such as homotopy, invariance under isometric transformation, robustness and efficiency, among others. One of the most common ways to compute a curve skeleton is the Voronoi diagram, which represents the space subdivision of the shape. The internal edges and faces of the Voronoi diagram are used to approximate the skeleton. There are different matching methods that use the curve-skeleton as a feature descriptor, such as [40] which uses a low-dimensional vector, whose components are based on the eigenvalues of the subgraph's (0,1) adjacency matrix, or [111] which consists in using a distribution-based similarity measure designed to evaluate dissimilarity between two multi-dimensional distributions.

A Curve-skeleton descriptor can be used for matching incomplete point clouds [112] and also for non-rigid registration [64].

2.2.10 Point Feature Histograms [H,P,LRF,G]

Point Feature Histograms (PFH) were presented by [102] in 2008. This method consists in extracting geometrical information from the neighborhood of a given point. Given a query point a_i from a point cloud $\mathcal{A}$, a sphere of radius r encloses the neighborhood N_{a_i}. All points lying inside the sphere are connected with the others via a fully interconnected mesh (see Figure 2.9 left). For each point $a_j \in N_{a_i}$ with a normal vector $\vec{n_j}$, the algorithm selects another point $a_k \in N_{a_i}$ where the angle between $\vec{n_j}$ and the vector defined by $(a_k - a_j)$ is minimum. Basically, that means that the algorithm is focused on concave zones of the shape. For each pair of points a_j and a_k $(j \neq k)$, a reference frame called *Darboux uvn frame* is computed $(\vec{u} = \vec{n_j}, \vec{v} = (a_k - a_j) \times \vec{u}, \vec{w} = \vec{u} \times \vec{v})$. Then, the angular information is calculated with these functions:

$$\alpha = v \cdot \vec{n_k} \tag{2.4}$$

$$\phi = \frac{u \cdot (a_k - a_j)}{||a_k - a_j||} \tag{2.5}$$

$$\theta = arctan(w \cdot \vec{n_k}, u \cdot \vec{n_k}) \tag{2.6}$$

Finally, the algorithm builds a histogram divided into bins, where bin space is arranged covering all values of the features. For each query point a_i, a descriptor histogram is computed according to the value of angular information of each pair of neighbors (a_j, a_k).

The correspondences between points from different shapes are found by sampling a number of described points. For each sampled point in $S_{\mathcal{A}}$, a set of points in $S_{\mathcal{B}}$ are selected. From these, one point is randomly selected and a transformation μ computed. The quality of μ is evaluated by computing its error metric.

The main drawback of the PFH is its high complexity $O(n \cdot m^2)$, where n is the number of points of $\mathcal{A}$ and m is the number of neighbors of each point. For this reason, the authors simplified the method and presented the Fast Point Feature Histogram (FPFH) [101], which reduces the complexity to $O(n \cdot m)$. Given a query point a_i, instead of calculating the relationships between all points in N_{a_i}, only the direct neighbors of a_i are taken into account (see Figure 2.9). The angular information of these pairs of points are computed and a Simplified Point Feature Histogram (SPFH) is made using this information. Thus, each point in $\mathcal{A}$ has its own SPFH, computed only with its direct neighbors. Afterwards, the SPFHs of the points inside the neighborhood N_{a_i} are used to weight the histogram of a_i, obtaining a $f^{\text{FPFH}}(a_i)$ (see Figure 2.9 right):

$$f^{\text{FPFH}}(a_i) = \text{SPFH}(a_i) + \frac{1}{m} \sum_{i=1}^{m} \frac{1}{w_m} \cdot \text{SPFH}(a_m) \tag{2.7}$$

where w_m is the distance between the query point a_i and its neighbor point a_m, used to weight the final value of FPFH(a_i).

FPFH is tested with real data with an overlap of ≈45%, and obtains good results combined with a specific Searching Strategy called SAmple Consensus Initial Alignment (SAC-IA) [101].

2.2.11 MeshHOG [H,M,LRF,G]

[124] dealt with local feature detection and description methods. The authors presented a 3D feature detector MeshDoG (Difference of Gaussians) (see Section 2.1) and a 3D feature descriptor MeshHOG (Histogram of Oriented Gradients) for uniformly triangulated meshes. The latter is a generalization of the histogram of oriented gradients (HOG) descriptor and uses two different parameters together in order to improve the surface registration: geometric and photometric information are extracted from the model in order to obtain more accurate results.

The shape function $f^{\text{MH}}(a_i)$ of this method is computed using support regions, defined using a neighborhood ring N_{a_i}. In order to make the descriptor invariant to rotation, a local coordinate system is taken into account. For each vertex of the neighborhood of a_i, the gradient information is computed. These gradient vectors are projected onto the three orthogonal planes from the local coordinate system in order to make the representation of the descriptor more compact. For each

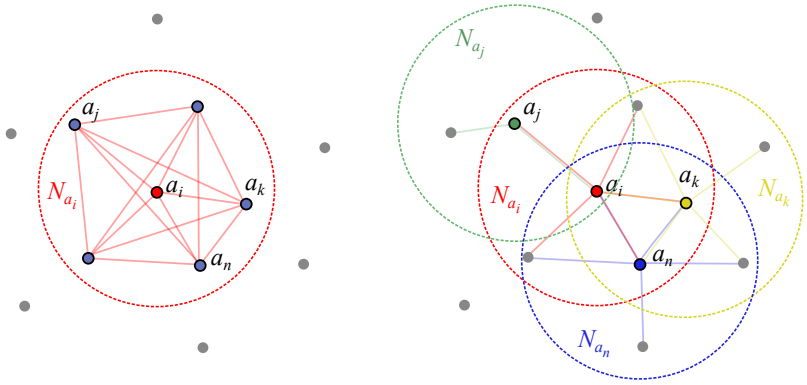

Figure 2.9: Left: PFH representation. All relations between neighbors in N_{a_i} are taken into account to compute $f^{\text{PFH}}(a_i)$. Right: FPFH representation. Each point only uses its direct neighbors to compute his own SPFH. Then, the neighboring SPFH's are used to weight the final descriptor value of $f^{\text{FPFH}}(a_i)$. Note: Only 3 neighbors of a_i are shown in the figure for major clarity.

of the planes, the authors compute a 2-level histogram. First, the plane is divided into b_s= 4 polar slices. For each slice, the algorithm computes an orientation histogram, with b_o= 8 bins for each projected gradient vector. f^{MH} is finally computed by concatenating $b_s \times b_o$ for each of the three planes. We present the sequence of the algorithm in Figure 2.10.

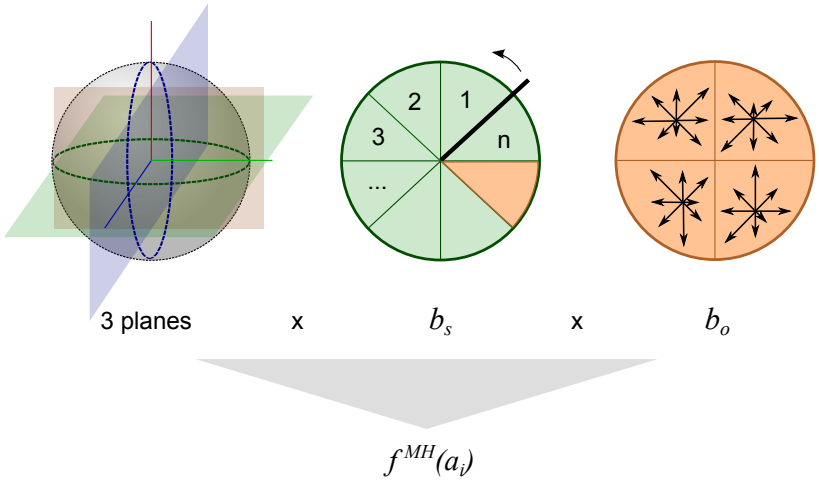

Figure 2.10: Construction of Histogram of Oriented Gradients (HoG). Left: Choosing 3 orthogonal planes onto which to project the gradient vectors. Middle: Polar coordinate system used for creating histograms via binning of 2D vectors. Right: Example of a typical spatial and orientation histograms, using 4 spatial polar slices and 8 orientation slices.

The authors use an intuitive greedy heuristic algorithm as a correspondence function cf^{MH} for descriptor matching. Given two surfaces $\mathcal{A}$ and $\mathcal{B}$, two sets of descriptors $S_{\mathcal{A}} \subset \mathcal{A}$ and $S_{\mathcal{B}} \subset \mathcal{B}$ are

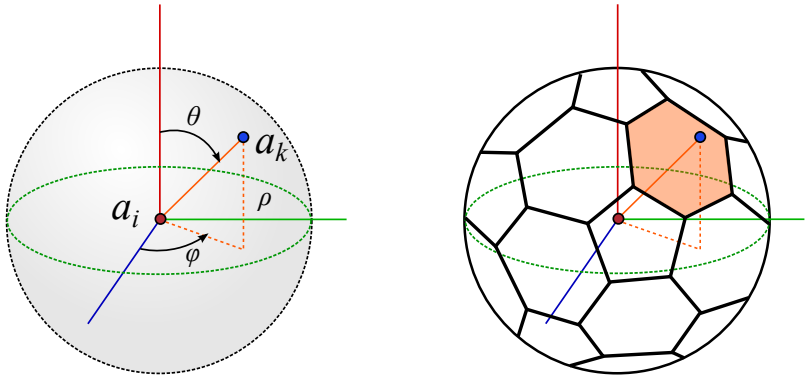

Figure 2.11: ISS representation. Left: Polar coordinates for neigboring points (ρ,θ,φ). Right: Sherical grid used to divide the angular space.

extracted from both shapes. For each descriptor in $S_{\mathcal{A}}$ the algorithm finds the best correspondence in $S_{\mathcal{B}}$ in terms of Euclidean distance. Then, a cross validation is performed by checking, for each descriptor in $S_{\mathcal{B}}$, the best correspondence in $S_{\mathcal{A}}$. The total cost of the matching process is $O(n^2)$.

The running time of the algorithm depends on the size of N_{a_i}. Authors conclude that the descriptor is robust under rigid transformations and outperforms the traditional purely photometric descriptors used in images.

2.2.12 Intrinsic Shape Signatures [H,P,LRF,G]

Intrinsic Shape Signatures (ISS) [126] is a point descriptor focused on shape retrieval problems. This method describes a 3D point using two different pieces of information: a Local Reference Frame (LRF) based on the eigendecomposition of the neighborhood's covariance matrix, and a 3D occupational histogram of the points in its spherical neigborhood.

Given a point set $\mathcal{A}$, a point a_i and a supporting radius r, the LRF is computed using the eigenvectors of the weighted covariance matrix of the neighborhood of N_{a_i} (e_i^x, e_i^y, e_i^z). Then, a feature vector is computed using a 3D occupational histogram of the supporting neighborhood N_{a_i}. Each neighbor a_k is coded using its polar coordinates with reference to LRF of a_i. A discrete spherical grid is used in order to simplify the histogram. Figure 2.11 shows an example of the process.

The ISS descriptor $f^{\text{ISS}}(a_i)$ is a combination of a LRF of a_i and the 3D shape feature vector. In order to find the correspondences between two shapes, the authors compare the feature vectors of the candidates using χ^2 statistics to compute the distance between two shape feature vectors.

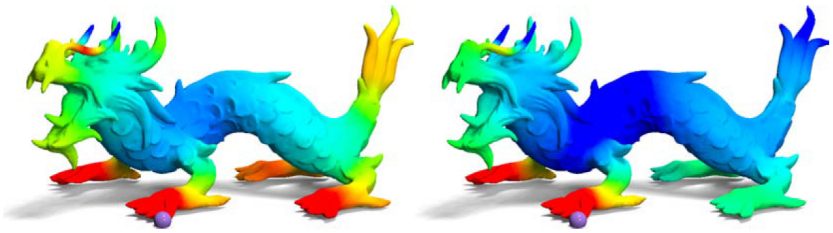

Figure 2.12: Color plot of the difference between the HKS defined by the range of scales $[t_1, t_2]$ of the point marked by the purple sphere and the signatures of other points on the shape. The difference increases as the color changes from red via green to blue. Left: both t1 and t2 are small; Right: t1 is small, while t2 is large. Figure taken from [110].

2.2.13 Heat-kernel Signature [S,M,nLRF,T]

Heat Kernel Signature (HKS), presented by [110], is a variation of the Heat Kernel which is a fundamental solution of Heat equation. HKS is based on the concept of heat diffusion on a surface over time. The authors propose HKS as both a detector and a descriptor method which possesses many properties: intrinsic, informative, multi-scale and stable against noise and perturbations. In order to reduce complexity, HKS focuses on the measure of heat diffusion on the considered point alone. The authors use a time parametrization because the time parameter provides a natural notation of scale to describe a shape around a point. The method thereby computes the heat that remains at a certain point at time t. Given a point a_i on a mesh $M_{\mathcal{A}}$, the authors define its Heat Kernel Signature ($f^{\text{HKS}}(a_i)$) as a function over the temporal domain, maintaining all the information of the Heat Kernel:

$$f^{\text{HKS}}(a_i) : \mathbb{R}^+ \rightarrow \mathbb{R}, \text{HKS}(a_i, t) = k_t(a_i, a_i) \tag{2.8}$$

The original Heat Kernel function $k_t(a_i, a_j) : \mathbb{R}^+ \times M \times M$ (M being a Riemannian manifold) can be interpreted as the amount of heat that is transferred from a_i to a_j in time t given a unique heat source at a_i. Due to the complexity of the computation, the authors restricted the function to a subset of $\mathbb{R}^+ \times M$. Despite this restriction, they showed that the Heat Kernel function $\{k_t(a_i, a_i)\}_{t<0}$ keeps all the information of $\{k_t(a_i, a_j)\}_{t<0}$.

The scale of the descriptor is given at timed intervals. For small values of t, the descriptor is focused on small neighborhoods, which provide a more detailed description. It can be used to describe the curvature of the surface. For large values of t, large neighborhoods are taken into account, obtaining a global descriptor of the shape, distinguishing large parts of the same object. Figure 2.12 is an example of the performance of the HKS: at small scales the claws are similar to each other. With large values of t, we can distinguish different parts of the dragon like front feet, back feet, head or tail.

The authors use local maxima of the function $k_t(a_i, a_i)$ for large t. Point a_i is a feature if $k_t(a_i, a_i) > k_t(a_j, a_j)$ for all a_j in the two ring neighborhood of a_i. The correspondence function cf is basically the comparison between both descriptors $f^{\text{HKS}}(a_i)$ and $f^{\text{HKS}}(b_j)$.

According to [110], despite the restrictions applied to the heat kernel, HKS preserves all the shape information, and also the stability against perturbations. The main drawbacks are the computation of the eigendecomposition, which is costly. This means that the computing time, in certain conditions with very large point clouds, can prove to be impractical.

[46] presented an application of HKS for the pose-oblivious matching of incomplete models. HKS is used to obtain a segmentation of the model in order to perform shape retrieval from a data base of complete, incomplete or partial models. HKS is used in conjunction with a point selection method based on persistent homology that consist of selecting a subset of the maximum values of HKS across different scales with large topological persistence.

[89] presented a method that uses HKS to conduct matching with isometries using only one-point correspondence on rigid and non-rigid transformations. The authors presented a new approach called Heak Kernel Maps. Given a fixed point in a manifold $\mathcal{A}$, this method creates a global shape descriptor. For each point a_i in $\mathcal{A}$, a heat kernel function is computed:

$$\Phi_p^{\mathcal{A}} : \mathcal{A} \rightarrow \boldsymbol{F}, \Phi_p^{\mathcal{A}}(a_i) = k_t^{\mathcal{A}}(p, a_i), \tag{2.9}$$

where $\boldsymbol{F}$ is the space of functions from $\mathbb{R}^+$ to $\mathbb{R}^+$. Thus, $\Phi_p^{\mathcal{A}}$ associates a real-valued function to every point $a_i \in \mathcal{A}$. This function is a one parameter function (t) given by $k_t^{\mathcal{A}}(p, a_i)$. The key issue in this approach is that only one correspondence is needed for the matching process. This is the direct consequence of the authors proving formally that the Heat Kernel Map is *injective*.

2.2.14 Rotational Projection Statistics (RoPS) [S,M,LRF,G]

RoPS [60] is a local feature descriptor that describes a point using a coarse partition of a 2D projection plane with rotational statistics of the surface, in combination with a robust Local Reference Frame (LRF) which is invariant to clutter and occlusions.

As we can also see in other approaches [38, 126], a robust LRF is computed by performing an eigendecomposition of the covariance matrix of the neighborhood around a given point. This LRF makes the descriptor invariant to rotation and translation changes. However, the sign ambiguity of the LRF results in a lack of precision.

Given a point a_i from $\mathcal{A}$, only a neighborhood N_{a_i} is considered using a sphere of radius r centered at a_i, for the descriptor computation. The neighboring points are rotated at an angle θ_k along the x axis of the LRF ($N_{a_i}^{\theta_k}$). Then, all points are projected into the xy plane, resulting in a 2D representation of $N_{a_i}^{\theta_k}$. This plane is divided into regular cells and, for each cell, the falling points are counted, producing a distribution matrix D. In order to make the descriptor more compact, several statistics are computed, such as *moment* and *entropy*, from each distribution

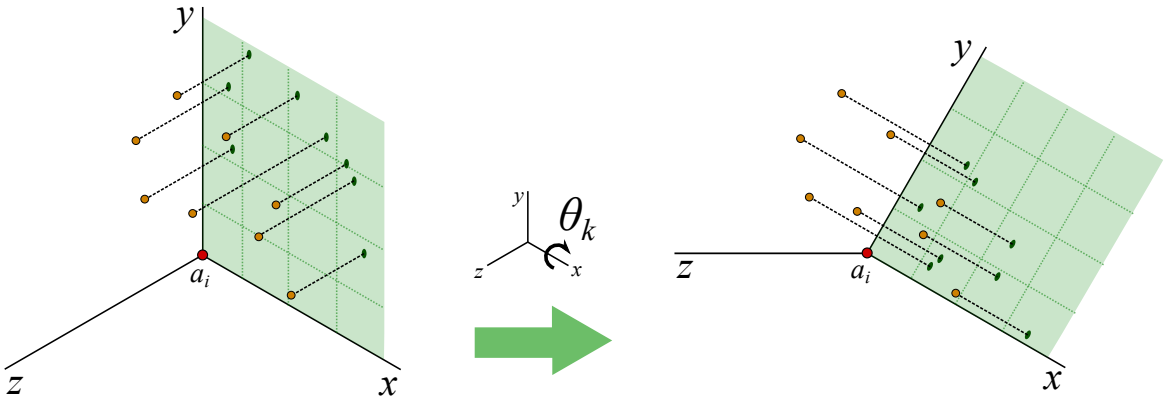

Figure 2.13: RoPS representation. Left: Projection of the N_{a_i} into xy plane from LRF of a_i. This step is done for yz and xz planes too. Right: Projection of the same N_{a_i} into a rotated xy plane over x axis.

matrix. Next, this process is repeated by projecting the point cloud into yz and xz planes, obtaining one feature vector for each projection. This operation is repeated with different angles θ_k, $k = 1,2,...,T$. Finally, all these steps are repeated for the other local axes y and z. Figure 2.13 shows a representation of the algorithm. The overall process produces many feature vectors that are concatenated to make the RoPS descriptor.

In [60], RoPS was compared to other methods such as Spin Image [66], Local Surface Patches [35] and SHOT [116], obtaining good results in noisy scenes as well as with varying mesh resolutions. However, the tests were made with processed data, taking only 1,000 random feature points of the original models.

2.3 Searching Strategies

Once the points in the two sets $(\mathcal{A}, \mathcal{B})$ to be matched have been filtered and their shape described, registration algorithms need to find the proper point correspondences between the two sets.

Methods that extract few key-points are able to use brute-force in order to find correspondences. However, this process is computationally expensive. Some methods reduce the computation time minimizing the search space, such as 3D Shape Context [69] which pre-selects the possible candidates satisfying certain criteria and then, applies brute-force with these candidates. Nevertheless, in most situations, more elaborate algorithms are necessary in order to report results in a reasonable amount of time.

As at least three non-coplanar points in each set are needed to determine a rigid transformation between two 3D point sets unambiguously, the asymptotic cost of such approaches is in $O(n^6)$. Consequently, the space to be navigated in the search for correspondences is huge. Devising a sophisticated search strategy that is able to take advantage of detection and description infor-

mation, has the potential to greatly reduce computation costs and, thus, increase the range of application of such registration algorithms. Existing methods implementing Searching Strategies already achieve very good results in comparison with typical brute-force methods.

As opposed to Fine Matching algorithms, the finality of these Searching Strategies is to achieve only a rough alignment. The idea is to identify the arbitrary position of input shapes and find the transformations between them, as quickly as possible. Precision is, thus, not the most important factor. Instead, robustness is key providing guarantees to subsequent Fine Matching. Henceforth we describe the most relevant Searching Strategies, in chronological order.

2.3.1 Algebraic Surface Model

Many methods work with triangulated meshes, or, at least, with correspondences between points in meshes. [115] proposed a Coarse Matching method that estimates a transformation using a polynomial model as a surface representation, without the need for point correspondences. This method consists of creating two polynomial models of the registering surfaces. The authors use a linear algorithm based on Least Squares called *3L Fitting* in order to obtain a distance function between the polynomial model and the points of the shape. Unlike other implicit polynomial fitting methods, the linear algorithm does not incur in high computational costs. The only requirement is to have the normal vector of each point in the surface to estimate the model. Furthermore, the computation time is faster than other Searching Strategies because this method does not need to calculate point correspondences.

However, normal vectors are required to estimate the models. These vectors are not easy to compute. The main drawback of this method is that the overlap between surfaces is required to be high (85% or more). This is not usually the case in real life applications.

2.3.2 RANSAC-based methods

RANdom SAmple and Consesus (RANSAC) is an iterative method designed to find the parameters of a model from a set of data which contains outliers. Given an input noisy data, RANSAC finds the parameters that adjust the input data to a given model, discarding the outliers. This approach is the base of a wide variety of methods. One of them is the approach presented by [34] which is based on the fact that we can determine a rigid transformation with only three points (a base B). The idea is to find a base in one of the shapes and find the corresponding base in the other shape. The algorithm works as follows: first, determine three different points randomly in the first surface: primary (a_p), secondary (a_s) and auxiliary (a_a). Consider the distances between these three points to be d_{ps}, d_{pa} and d_{sa}. Each point in the second surface is considered as the corresponding point b_p of the primary point a_p on the first surface. Then, the correspondence of the secondary point is searched on the second surface at distance d_{ps} from b_p. If no point around b_p at distance d_{ps} exists, discard b_p and start again with another primary point in the second surface. However, if there is a secondary point b_s look for the auxiliary point b_a that

satisfies the distances. The transformation between both surfaces can be determined when the base $B_{\mathcal{B}}$ in the second surface is identified. This search is repeated for all the bases found. The best transformation is the one with the highest number of corresponding points.

Although this method is robust even with outliers, the main drawback is its computation time. In fact, this method is only usable with a small amount of input data, as was stated in [47, 104].

[120] presented an improvement of classic RANSAC called Random Sampling (RANSAM). This approach consists of randomly selecting four oriented points (points with their normal vector) from both surfaces ($(a_i, a_h) \in \mathcal{A}$ and $(b_j, b_k) \in \mathcal{B}$) using a *Monte-Carlo* algorithm. Bases of two oriented points are sufficient to determine a rigid transformation. This yields a searching complexity of $O(n^2)$.

The correspondences between points are encoded in 4D relation vectors. These vectors consist of the Euclidean distance between the points, the angles of inclination between their normal vectors, the line connecting them, and the rotation angle between the normals around the connection line. The search for correspondences is performed using a hash table that stores these relation vectors. The use of this hash table allows the complexity of the algorithm to drop to $O(n)$.

Whenever a correspondence is found, a rigid transformation is computed. The quality of the registration is measured, then, by estimating the proportion of the overlapping areas between the two point clouds. A movement is considered a solution if the distance between points in both surfaces is smaller than a certain threshold. The selection of points can be improved using descriptors that weight the random selection.

Another method which uses randomized algorithms is *property testing* [96]. This approach consists of determining whether a given object has a predetermined property or is "far" from any object having the property. This methodology can be applied in computational geometry problems [42].

2.3.3 Robust Global Registration

[58] presented a Coarse Matching approach based on looking for correspondences using a branch-and-bound algorithm. Given two shapes $\mathcal{A}$ and $\mathcal{B}$, the method consists of extracting a set of key points $S_{\mathcal{A}}$ from $\mathcal{A}$ using integral invariants volume descriptor [78]. In order to increase the robustness, the authors use multi-scale resolution in the description process. For each feature point a_i in $S_{\mathcal{A}}$, the algorithm finds a sub-set of points in $\mathcal{B}$, called $C_{\mathcal{B}}(a_i)$, with a high correspondence with a_i. In order to reduce the correspondence list of points, a thresholding function is applied over $C_{\mathcal{B}}(a_i)$: considering a pair of points (a_i, a_j) from $S_{\mathcal{A}}$ and a pair of potential corresponding points (b_i, b_j) from $C_{\mathcal{B}}(a_i)$. The distance between a_i and a_j needs to be approximately the same as the distances between their correspondences in the model.

The Searching Strategy uses a branch-and-bound algorithm that creates a solution-candidate tree where each branch represents one possible correspondence set of points from $\mathcal{B}$. In each level of the tree, one possible candidate is added to the solution. If one of these possible candidates

does not pass the threshold test and thus, the Root Mean Squared Distance (RMSD) between $\mathcal{A}$ and $\mathcal{B}$ is not improved, the entire branch is pruned. The whole tree is explored finding the best correspondence set where all correspondences pass the threshold and provide the minimum error (RMSD) between both shapes.

This method is robust to noise, and works well with occluded scenes and partial registration. However, the algorithm needs strong feature points in order to obtain good alignments. The uncertainty created due to the use of weak feature points, increases the error both between registered points and the searching time.

The authors also extend their algorithm to detect symmetries registering an object with a copy of itself.

2.3.4 4-point Congruent Set

[17] presented a searching strategy that takes advantage of the geometric properties of coplanar groups of four points in order to devise a method that, while using more than the usual three points to determine motions, can be shown to incur lower asymptotic costs. The method finds a transformation between two views using a coplanar set of points with no assumption about the initial alignment.

The authors use 4 coplanar points from $\mathcal{A}$ to build a base $B_{\mathcal{A}}$, and find its correspondent base in $\mathcal{B}$. Although the extra point is not mandatory in order to compute a movement, it makes the process more robust and allows the authors to provide proof of reduced asymptotic costs. The key to this method in terms of speed is the use of wide bases. Figure 2.14 presents a comparison between wide and narrow bases. Performing the alignment process with wide bases makes the registration more robust because the alignment is affected less by errors in accuracy. With narrow bases, a small perturbation of the base might ripple away to induce a noticeable displacement of the full object.

The algorithm works as follows: given two surfaces $\mathcal{A}$ and $\mathcal{B}$, 4 almost coplanar points are selected from $\mathcal{A}$ (base $B_{\mathcal{A}} = \{a_i, a_j, a_k, a_l\}$). The algorithm chooses 3 random points close to each other and selects the remaining point such that the 4 points together form a wide base which is approximately coplanar. In order to find the best 4-points set in surface $\mathcal{B}$ that are approximately congruent to $B_{\mathcal{A}}$ (up to an approximation level δ), the authors use a descriptor of 4-point set, based on distance ratios between points in the base:

$$r_1 = ||a_i - e||/||a_i - a_j|| \tag{2.10}$$

$$r_2 = ||a_k - e||/||a_k - a_l|| \tag{2.11}$$

where e is the intersection point between $a_i a_j$ and $a_k a_l$ lines. These two ratios are invariant under affine transformations. Thus, 4-point sets from $\mathcal{B}$ that have approximately the same ratios than $B_{\mathcal{A}}$ are identified. The full algorithm runs in $O(n^2)$.

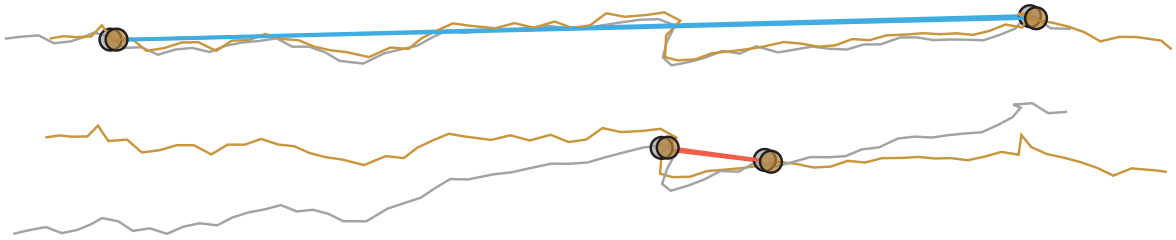

Figure 2.14: Comparison between wide-base (top) and narrow-base (bottom) of the 4PCS method, taken from [17]. Golden and gray curves represent two different surfaces to be registered.

The authors justify not using local descriptors because they are not robust to noise and outliers, applied with real data. Instead, the authors rely on the Principle of Large Numbers. This principle applies in the sense that, although particular point correspondences might be overlooked, the high number of corresponding points between the two sets allow for a large number of solutions. In this approach, this principle requires solving the Largest Common Pointset (LCP) problem. LCP under δ-congruence reports a subset of $\mathcal{B}$ that has the largest possible cardinality, where the distance between $\mathcal{A}$ and $\mu(\mathcal{B})$ is less than δ.

The authors report and provide experimental proof of how the combination of wide bases and LCP makes the registration method resilient to noise and outliers.

4PCS is compared with a combination of local descriptors with RANSAC. The authors use Spin-Image [72] and Integral Invariants [92]. As we can see in the Figure 2.15, 4PCS outperforms LD-RANSAC. An additional aspect of 4PCS in this case is that, as opposed to LD-RANSAC, it does not need parameter tuning.

In 2014 Mellado et. al. introduce an improved version of 4PCS named Super4CPS (S4PCS) [84], which runs in linear time. The key insight of this approach is to remove the quadratic complexity in the original 4PCS algorithm by using an efficient yet practical data structure to solve the core instance problem, i.e., finding all point pairs that are within a distance range $(r-\epsilon, r+\epsilon)$. Specifically, Super4PCS runs in $O(n+k1+k2)$ time where $k1$ is the number of pairs in $\mathcal{B}$ at a given distance r and $k2$ is the number of congruent sets. The proposed data structure naturally extends to higher dimensions and allows a unified treatment of spatial and angular proximity queries.

2.3.5 Evolutionary methods

Evolutionary methods (EM) are Searching Strategies based on computational models of evolutionary processes that carry out the registration without any initial estimation of the initial alignment and without needing refinement. The idea is to use fitness functions to measure the quality of each potential solution. A remarkable example of the application of genetic algorithms for surface registration can be found in [36]. [18] presented a game-theoretical approach for

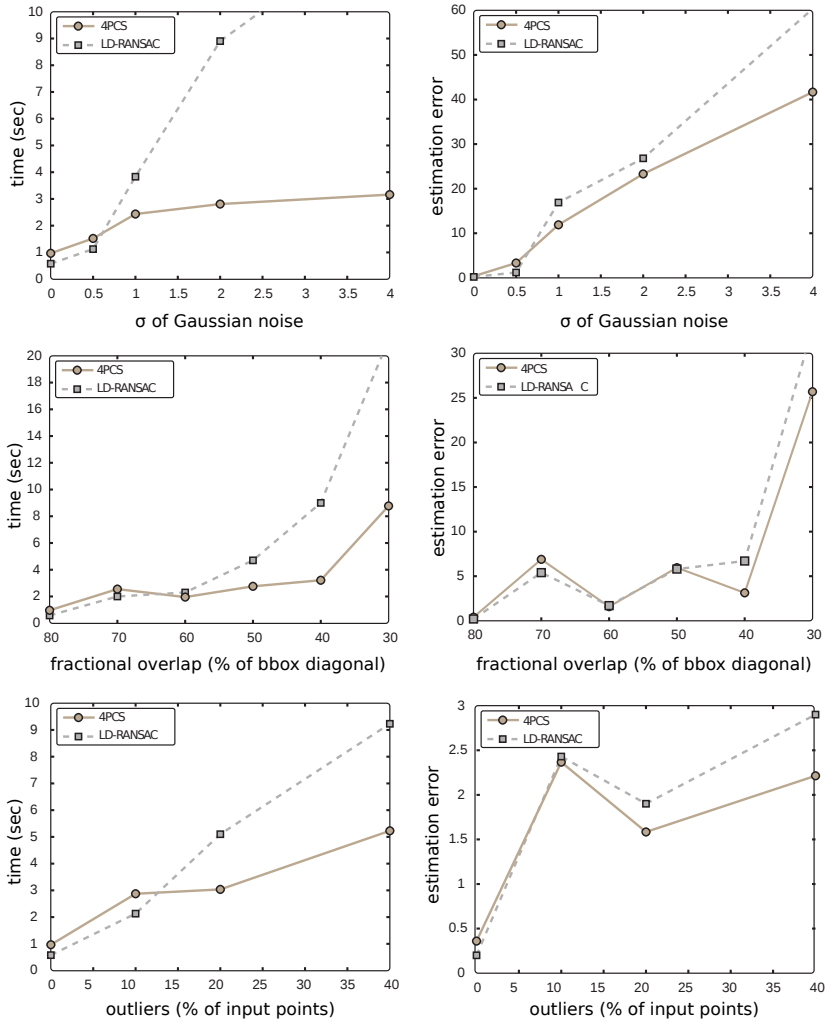

Figure 2.15: Performance and comparison taken from [17], between 4PCS and LD-RANSAC. We can observe that with low overlap ratios, high level of noise or many outliers, 4PCS outperforms LD-RANSAC, in terms of estimated error and computational time.

surface registration, that consist of casting the selection of correspondences in a game-theoretic framework, where a natural selection process allows matching points which satisfy a mutual rigidity constraint to thrive, eliminating all the other correspondences. Other strategies are used: stochastic sampling, classical one-point crossover or simply bit flipping mutation.

A thorough study was presented by [105]. This work reviews the literature concerning evolutionary image registration methods for 3D modeling, including an experimental study. Evolutionary methods are tested against classical ICP methods. One conclusion reached is that the most evolutionary methods outperformed the classical approaches based on ICP. Moreover, with EM a pre-alignment of input surfaces is not necessary. However, as stated by the authors, genetic algorithms present expensive computational times, making these methods inappropriate

when time is a critical factor.

2.3.6 Hybrid methods

Due to the high computational cost required to find an aligning motion between two views, the Searching Strategies aim to find algorithms to approximate the solution without paying the complete cost. However other approximations exist using external information. A recent trend [9, 12] seeks to use sensors from mobile devices in order to help registration. These sensors are commonly available and less expensive, circumventing many of the previous problems. The goal in this case is to propose hybrid 3D registration methods which combine the best features of software and mechanical approaches. Significantly, [94] presented a hybrid algorithm with a hardware part based on smartphone technology that managed to gain access to rotation data that could then be used to search for the remaining translation part of the rigid motion by software means.

The idea of disaggregating the rotation and translation part was not new and can be found, for example in [71]. Once the rotation part of the problem is solved (either by using hardware data as in the former reference or by software means as in the later), what remains is determining the translation vectors that brings the two sets being registered closer together. To solve this problem it is enough to find one single correspondence between one point in each set so that the resulting translation brings the sets close enough for a fine registration to succeed. This yields an immediate $O(n)$ asymptotic cost for this version of the problem.

Actually, when external information is available, hybrid methods become a preferable option for registration. In [98] we developed a hierarchical version of [94] is presented, obtaining good results and outperforming S4PCS method.

2.4 Refinement

The last part of the registration process is the refinement of the alignment achieved by Coarse Matching. This step is also commonly referred to as "Fine matching". The most commonly used method nowadays is Iterative Closest Point (ICP), presented by [24]. This method has become a standard in the research field of registration due to its robustness and reliability. Given an initial coarse registration, the method associates points from two different point clouds by nearest neighbor criterion, uses Mean Squared Distance minimization functions in order to estimate the movement, transforms points according to these functions and iterates until convergence. At the same time that Besl, [121] presented a method following a similar approach. Later, [100] presented several variations of ICP improving the precision of the algorithm, introducing several filtering methods like Normal Space Sampling (NSS), as mentioned previously. Nowadays, there are many improvements of ICP, such as [59, 77, 88, 107], to name a few.

Other approaches are able to solve the same problems as ICP, each with a slightly different focus: examples are *Matching signed distance fields* [80, 81], or *Evolutionary Methods* [36], which are, in most cases, able to solve both the Coarse and Fine Matching problems.

In this thesis we mainly focused on Coarse Registration part because is an open problem nowadays. In Fine Registration topic, although some improvements are presented every year, the majority of the community is focused on Coarse methods, using ICP for Fine Matching.

2.5 Discussion

Every paper studied in this chapter was tested by the authors under different conditions. For this reason, it is very difficult to compare their reported experimental performances. However there are some reviews and benchmarks which implemented and tested the most commonly used methods [28, 29, 52, 68, 103, 104, 113, 114, 118, 123]. Using the results reported in the literature, here we extract some overall conclusions.

In order to evaluate the performance of the different methods, we focus on three main issues: precision, robustness and efficiency. We understand *precision* as how accurate the method is, considering error measurements presented in the experimental results of every paper. *Robustness* is the resilience of the method against outside perturbations, such as noise, occlusion or cluttering. Finally, *efficiency* is measured according to the runtimes provided by the authors, taking into account the data size and the complexity of the algorithm.

Furthermore, we take into account the type of the data used in each proposal. Repositories of processed scanned models like The Stanford Repository[2], AIM@SHAPE Shape Repository[3] or Ajmal Mian's Databases[4] are the most widely used because they provide useful data for the tests. However, these models consist, usually, of only processed data. Besides, there are authors who use scanned data without any pre-processing. This kind of input data provides more realistic situations for the testing of algorithms.

For this discussion we follow the order of the proposed Registration Pipeline (Figure 1.1). We focus on Detectors, Descriptors and Searching Strategies. For a thorough Fine Matching discussion, see [104].

2.5.1 Discussion on Detectors

Detection speeds up computations and, thus, enhances the range of applicability of algorithms. However, if the detection is not done properly, important information might be lost and existing matches overlooked. If a detection algorithm is able to consistently produce a similar output for the same object under different conditions (noise, change of view, etc.) then this problem is minimized. Consequently, we focus on the repeatability of the detected points over all executions.

[2] http://graphics.stanford.edu/data/3Dscanrep/
[3] http://shapes.aimatshape.net/
[4] http://www.csse.uwa.edu.au/~ajmal

A first general conclusion to be drawn is that most of the approaches reviewed use only processed data. Only a few papers present results with real scanned data. Additionally, important differences are observed between these two types of data whenever reported. We believe that obtaining results with real application data represents a mandatory step towards truly practical algorithms.

The methods which achieve higher repeatability results, all with processed data, are **Heat Kernel Features** (HKF) [110], **Harris 3D** [109] and **MeshDoG** [124], which are tested in [28, 29, 103, 104]. According to [29] and [28], HKF achieves a ≈93% of repeatability, retrieving between 9 and 23 feature-points from a point cloud of 10,000 points. Under the same conditions, Harris 3D and MeshDoG obtain ≈83% and ≈87% of repeatability, respectively. Harris 3D demonstrates notable robustness against holes and topological changes in the input surface, while MeshDoG performs when it comes to scaling variances and noise. Furthermore, in [52], HKF is compared, among other methods, with a human ground truth. These experiments, with processed data alone, demonstrate the good performance of HKF, retrieving points that are usually selected by human subjects.

We mention two other methods, **Intrinsic Shape Signatures** (ISS) [126] and **Key-Point Quality** (KPQ) [85], both tested in [103]. Although these methods obtain slightly worse results than those of the methods mentioned above, both are tested using processed and real data. ISS reports ≈70% of key-point repeatability using processed data. The authors demonstrate a good recognition range using ISS to retrieve similar models from a database. KPQ achieves lower results in terms of repeatability (≈58%). With real data taken from scans, KPQ achieves similar repeatability results, yet the performance of ISS decreases considerably (≈30%). In terms of temporal efficiency, however, ISS is much faster than KPQ in all tests.

Using volumetric input data, DoG, Harris3D, and **MSV** are tested in [123], among others. The volumetric-specific-designed method MSV obtains the best results yet proves slower than the others. In terms of time efficiency **Hierarchical Normal Space Sampling** [47] outperforms the runtimes of Normal Space Sampling [100] obtaining a ≈99% of time reduction in the most extreme case, besides its precision improvements. The authors stress the importance of hierarchical data structures in order to speed up the searching of correspondences.

Upon analyzing the reported results, we note that there are methods which are more restrictive than others. Specifically, methods like HKF reports very few key points, which are expected to convey more distinctive shape information. Conversely, other methods such as ISS report many more points in comparison. Although HSF performs better than ISS in most situations, in some others such as low overlapping ratios or missing data, it might be advisable to use more key points. Using fewer feature points in situations where some parts of the shapes are missing might lead to sampling instances which actually prevent the finding of a match. This observation is supported by the results reported in [29], where HKF obtains its worst result when applied to models with holes and shot noise. Consequently, robustness to noise and occlusions should

also be considered when choosing which detector to use. Table 2.1 summarizes all the detectors reviewed in this section.

Table 2.1: Summary of detector-based methods, sorted by publication year.

Year	Method	Description
2001	**Normal Space Sampling** [100]	Selects points with high distinctiveness.
2006	**Maximally Stable Volumes** [51]	Detects stable regions across different scales.
2009	**Heat Kernel-based features** [110]	Selects key-points according to head distribution function.
2009	**MeshDoG** [124]	Uses DoG operator to detect points with maximum value of the Laplacian function.
2009	**Intrinsic Shape Signatures** [126]	Uses the neighborhood's covariance matrix to detect key-points.
2010	**Key-Point Quality** [85]	Uses the neighborhood's covariance matrix and principal curvature to detect key-points.
2011	**Harris3D** [109]	Harris operator for 3D point cloud registration.

2.5.2 Discussion on Descriptors

Descriptors represent a very active field of research in terms of the number of published papers. More precisely, object retrieval methods are, nowadays, one of the most popular topics within Descriptors. According to the literature there are many different approaches, but all of them have the same target: providing a useful representation of the shape around the given point which facilitates searching for correspondences between two shapes, thereby avoiding exhaustive searches.

As discussed in Section 2.2, many of the methods are histogram-based. These approaches are easy to implement and also incur low computational costs. However, whenever a part of the compared surfaces is missing or is extremely perturbed, the resulting histogram might also be largely perturbed. This is produced by the strong dependency of these types of descriptors on local reference frames. This dependency makes these approaches more sensitive to noise and occlusions. It is not clear whether histogram-based methods are able to overcome these kinds of problems. However, some methods such as **ISS** [126], **SHOT** [116] or **Improved SI** [125] show a good resilience to synthetic noise, and obtain promising results in many different situations, as we see in [103, 116, 125]. In [116], the authors present an experimental study which proves that using a unique and unambiguous LRF improves the precision and the accuracy of the descriptor.

In [68] SHOT, Spin-Image, **Shape Contexts** [69] and **FPFH** [101] are tested using real data acquired with different techniques. The latter proves to be the most stable and fastest method throughout the tests. With models possessing a high level of irregularities, SHOT works better, yet it fails with regular surfaces.

Another approach to take into account is **PCA** [38]. Although it is not robust to noise and sensible to errors and occlusions, it is one of the fastest methods. For this reason, there are many other algorithms that use PCA as a local descriptor, reporting a local distribution of the neighborhood around a given point. Examples are [43, 92, 109, 122] to name a few. Using PCA in a local neighborhood provides descriptions less sensitive to noise.

Both in terms of accuracy and time efficiency, the methods reporting best recent results are, **Integral Invariants** [78, 92] and **Heat-Kernel Signature** [110]. Both methods outperform other algorithms in terms of speed and accuracy, with input data sizes around 100,000 points. Due to the complexity of integral computation, Integral Invariants is slower than HKS. However, most methods are tested only with processed data, where the ground truth is clearly known. With real data, the results are quite different. For example, HKS shows high repeatability and distinctiveness with processed models [29], yet with laser-scan data or image-based reconstructions, this method is too selective to report a robust registration [68]. Table 2.2 summarizes all the descriptors reviewed in this paper.

Table 2.2: Summary of descriptor-based methods, sorted by publication year.

Year	Method	Description	Category	Input data	Ref. Frame
1996	**Principal Curvature** [54]	Use the principal curvatures of the surface as a descriptor	Sign.	Points	No
1997	**Point Signature** [37]	Describe a point with the curvature of the surface arround	Sign.	Points	Yes
1997	**Spin-Image** [65]	Histogram of the relative position of the neighbors	Hist.	Mesh	Yes
1998	**PCA** [38]	Principal directions of the shape	Sign.	Points	Yes
2003	**3D Shape Contexts** [69]	Describe a point with the position of certain points of the object	Hist.	Points	No
2003	**Line-based algorithms** [119]	Describe a point using the curves of the surface	Sign.	Points	No
2006	**Integral Invariants** [78]	Descriptor that use the volume below the surface	Sign.	Mesh	Yes
2007	**Point Feature Histograms** [102]	Describe points according to the normal distribution of its neighborhood.	Hist.	Points	Yes
2009	**MeshHOG** [124]	Descriptor based on the gradient information over different scales	Hist.	Mesh	Yes
2009	**Intrinsic Shape Signatures** [126]	Histogram of the relative position of the neighbors	Hist.	Mesh	Yes
2009	**Heat-kernel Signature** [110]	Descriptor based on the heat diffusion over the surface	Sign.	Mesh	No
2010	**SHOT** [116]	Descriptor that encodes histograms of the normals	Hist.	Mesh	Yes
2012	**Scale-Invariant Spin-Images** [43]	Scale-invariant formulation of Spin-Image descriptor	Hist.	Mesh	Yes
2012	**Local-Depth SIFT** [43]	Use the SIFT operator over 2D depth map of the neighbor positions	Hist.	Points	Yes
2012	**Improved Spin-Image** [125]	Encode angle information between normals and neighbors	Hist.	Mesh	Yes
2013	**RoPS** [60]	Uses rotational statistics of the surface to describe points.	Sign.	Mesh	Yes

2.5.3 Discussion on Searching Strategies

There are very few published papers related to Searching Strategies (Table 2.3). The tendency in the literature is to use good detectors and descriptors, retrieving a number of feature points in order to use them in brute-force matching strategies. ICP is then used to refine the alignment. Searching Strategies make the searching step more efficient for an initial pose for ICP, helping spread the computational costs, while making the process less descriptor-dependent.

Concerning **Robust Global Registration** [58], the method achieves good results with processed models up to 68,000 points yet, without strong feature points, the computation time and errors increase noticeably. Similarly, **Evolutionary algorithms** [105] obtain very good results in terms of precision yet they are computationally expensive dealing with large amounts of data. **RANSAM** obtains good results in terms of performance and accuracy. Working with ≈60,000 points the computation time is reasonable. For example, RMS precisions of 1.03mm, consisting of mechanical ground-truth provided with a high-precision turn table, achieves runtime values

around 0.5 seconds. No data is provided, however, on the robustness to noise of the algorithm. We feel that this might well prove to be an issue as algorithms using normal vector information have been reported to present sensibility to noise [104].

Even though no proper experimental comparison of Searching Strategies exists, we consider that the best approach nowadays is **Super 4-points Congruent Set** (S4PCS) [84], because it achieves very good results in many different situations (see Section 2.3 for details). Using a standard laptop computer, the authors deal with huge point clouds achieving very accurate registration outputs with low computation time. Moreover, the authors tested their algorithm with real scanned data achieving more accurate results than others compared to any RANSAC-based method (see Figure 2.15).

Table 2.3: Summary of Searching Strategies, sorted by publication year.

Year	Method	Description
1998	**Algebraic surface model** [115]	Motion estimation using polynomial models.
1998	**RANSAC-based methods** [34]	Find the same 3-point bases between two models that preserves the euclidean distances between them.
2005	**Robust Global Registration** [58]	Find correspondences using a branch-and-bound algorithm.
2006	**RANSAM** [120]	Select points randomly and use distance and angular relationships between points to find a good movement.
2008	**4-point Congruent Set** [17]	Use a coplanar 4-point bases to find the correspondences between two models.
2011	**Evolutionary Methods** [105]	Align two different shapes using evolutionary algorithms without no other assumptions.
2014	**Super 4-point Congruent Set** [84]	Improve 4PCS using efficient data structures.
2016	**Hybrid Methods** [94]	Use external information from sensors to speed up the computation.

2.6 Conclusions

In this chapter we reviewed state-of-the-art methods for point cloud rigid registration and we proposed a pipelined classification in order to organize the available approaches.

Working with synthetic or processed data makes it possible to create ground truth values to test the methods. This stands for a far better controlled testing scenario and allows us to focus on specific algorithmic aspects. However, as we will see in next chapters, testing methods with only synthetic data does not provide a complete understanding of their performance. The majority of the methods are sensitive to noise and outliers, which are inherent of real scanned data. For this reason, experimentation with real data is always a crucial step towards application. We believe the lack of comprehensive studies on real data represents another sign of the intrinsic complexity of the problem.

According to the number of published papers, current trends are focused on detection and description methods, both using the same shape function either to detect or describe the key-points. We noted that, besides registration between two objects, many approaches also deal with the problem of 3D shape retrieval from object databases. Herein lies an important difference because these kinds of methods tend to use fewer feature points, in order to reduce the runtime needed to match the model with all objects in the database. The more data points considered, the

more precise the final result. Consequently, when working with real data perturbed by noise and occlusions and related to real life applications, a large number of points is often necessary and, thus, efficiency becomes a key factor. Another issue that arises is that most of the studied methods focus on one of the steps of the Pipeline only -usually description- and use brute-force in order to determine the final motion. We believe that combining a good descriptor with a sophisticated Searching Strategy such as those reviewed in Section 2.3 would improve the efficiency of the methods. Moreover, this combination has the potential to extend the degree of detail admissible in shape retrieval libraries.

There are few publications focused specifically on improving the Searching Strategies for matching correspondences between feature points from different shapes. We believe that this field of research may yet produce more efficient methods, using advanced algorithms and data structures such as kd-trees, octrees or other GPU-friendly structures in this part of the Registration Pipeline. This fact brings ever closer the registration topic to other fields of research such as computational geometry, where this problem is tackled from a more theoretical point of view. We believe that the inclusion of these resources has the potential to improve the state-of-the-art, particularly in terms of time efficiency. We believe that this possibility together with the good performance, already shown using real data, shows how Searching Strategies are ready to be integrated in a fully practical Registration Pipeline that can deal with a variety of application problems.

ICP and its improvements are the most commonly used refinement methods. The majority of the papers studied in this review used ICP to refine the initial alignments. Although there are other approaches which achieve good results, they incur in high computational costs and are only available for small input data.

It is difficult today to compare the performances of existing registration methods. This is due to the lack of a standardized evaluation methodology as well as commonly accepted benchmarks. These results are presented in different magnitudes, or, in some cases, they go unreported. Although there are some benchmarks [28, 29, 52, 68, 103, 104, 113, 114, 118, 123] that provide comparative studies, they are far from comprehensive and difficult to extend to all methods in the literature. We believe that it is necessary to define generic guidelines to be used in order to test the performances of registration methods. What may be more interesting still might be to define standard conventions for the study and presentation of results. For example, it is important to report the runtime of each part of the process, computer characteristics, data sizes, number of feature points, evaluation measures, etc.

To sum up, we believe that most parts of the Registration Pipeline presented have reached a point where the existing methods have already been shown to be usable in practical situations, or seem to be quite close to it. Consequently, a reasonable expectation is that, in a near future, it will be possible to present a registration algorithm optimized in terms of all these Pipeline steps. Such a method would have the potential to greatly increase the current areas of application of

point cloud registration as well as the sizes of the data sets used. We believe that the design of such a method and the provision of experimental proof of its ability to work on a variety of real-life situations represent both a challenge and an opportunity for the research communities involved.

In summary, in this chapter we thourougly reviewed the registration problem as well as defined a standard notation to unify all methods involved. We classified the main state-of-the-art algorithms in our pipelined classification and we compared them according the published results. With this knowledge we are able to propose the best method combination to tackle different scenarios.

CHAPTER 3

3D REGISTRATION TOOLBOX

In Chapter 2, we showed how different algorithms can be used at each step of the Registration Pipeline. However, although many papers are presented, there are not many software available and the existing ones are created using different notation and implementation guidelines. This fact make more difficult the integration of algorithms from different steps of the Pipeline and the comparison between them. Thus, our aim is to provide a framework with a common notation which includes as many methods as possible, being a useful tool for the community. The contribution that we present in this chapter is a 3D Registration Toolbox, which consist of a 3D registration software collection and a model database. The first, allows researchers to execute all different steps of the Registration Pipeline independently or in a common workflow. The second, provides different 3D models and precomputed aligning information which allows the researchers to check the results of their algorithms in a controlled scenario. We developed the Toolbox so it is easily expandable, so that researchers can include their methods easily. This modularity increases the complexity of the code, because any method can be added without effecting the rest of the software. We developed this Toolbox aiming to be a useful tool for both Developers, which are researchers that want to add their algorithms in the Pipeline, and Users, who only want to run the methods included. Thus, we provide a common testing environment, making more easy the comparison between all methods.

In this chapter we introduce the structure and main characteristics of our Toolbox. The work presented in this chapter is under preparation to be submitted in a journal in the next months. The code and the documentation can be downloaded at [7].

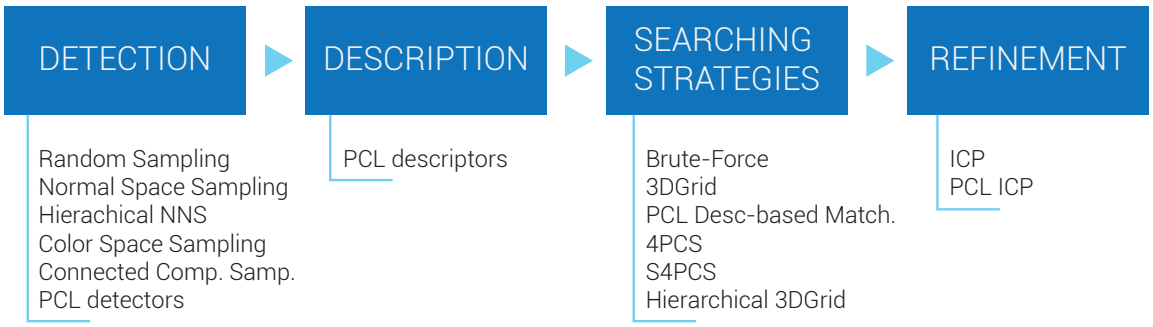

Figure 3.1: Registration Pipeline scheme.

3.1 Toolbox overview

Our 3D Registration Toolbox consist of two main parts: a 3D registration software and a model database. We implemented the Registration Pipeline in C++. This allows Users to test a wide variety of registration methods. We structured this part according to our pipelined categorization (see Figure 3.1) —Detection, Description, Searching Strategies and Refinement— . Besides our own registration algorithms, we included algorithms from the Point Cloud Library (PCL) [15], the 4PCS [17] and Super4PCS [84] methods. PCL provides a lot of implementations from the community and they can be used in any part of the Pipeline. Also, we provide a model database with 6 3D objects (See Figure 3.2), each with different views. With each model, an alignment movement is provided as well as other relevant information such as overlapping or residue between views.

3.2 3D Registration Pipeline

In this section we present our ready-to-run software able to align different views of a certain model using the state-of-the-art methods reviewed in Chapter 2. In order to be as modular as possible, we developed our Pipeline following the S.O.L.I.D. principles [5] which establish a code of practice of programming: **S**ingle-Responsibility, **O**pen-closed, **L**iskov substitution, **I**nterface segregation and **D**ependency Inversion. The key reason to follow SOLID principles is the fact that any combination of Pipeline steps is allowed. For example, depending on the requirements, the user can work with only Detection and Refinement methods, or only using Searching Strategies to compare coarse matching methods. This restriction makes the implementation more complex and a high modularity becomes mandatory. Figure 3.4 shows an overview of the class diagram.

Below we explain the main classes and all the steps of our Pipeline.

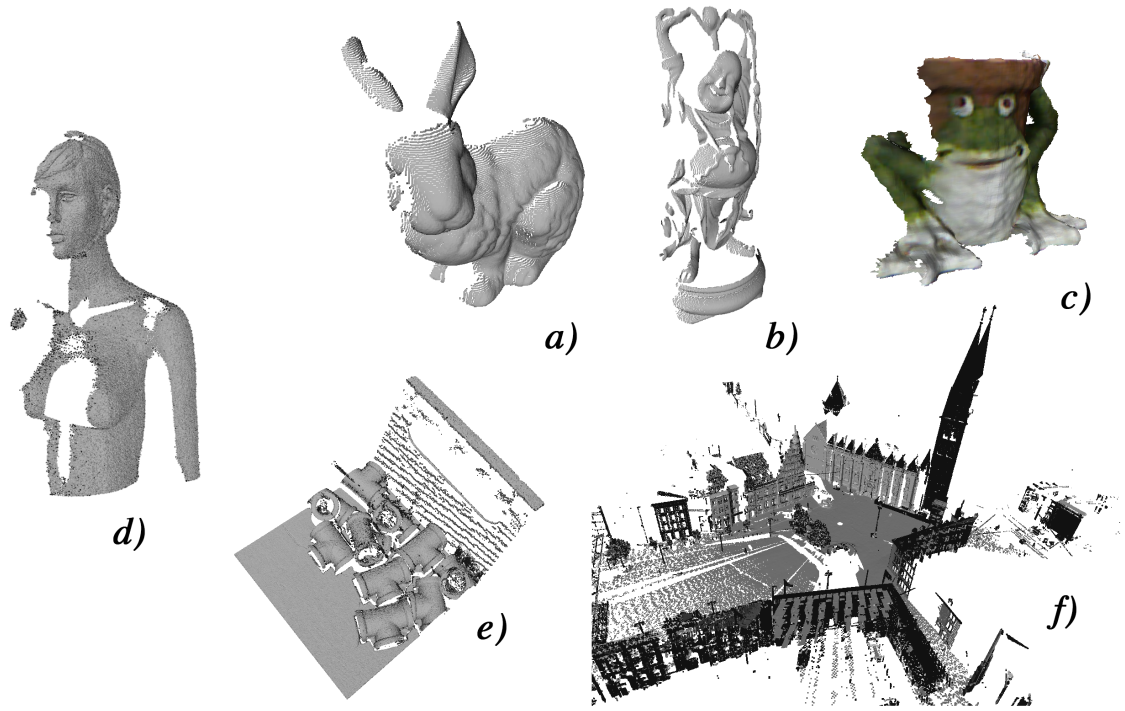

Figure 3.2: Main views of the models in the database: a) Bunny [3], b) Buddha [3], c) Frog [16], d) Bust [95], e) Joints and f) Bremen [4] models.

3.2.1 Class *ElementSet*

Class *ElementSet* is used to store point clouds. That might represent one of the input views or some intermediate point cloud. In order to store a point cloud inside an *ElementSet*, we use a vector of *Point* objects. The access to this vector is done using pointers because we want to keep constant the original point cloud. We create two different access vectors (See Figure 3.3): *allpoints* that contains pointers to each point and *workpoints* which is used across the Pipeline to operate with the point cloud. We use this kind of structure because some methods need only certain points (i.e. keypoints from a Detection step) and other methods needs the complete point cloud (i.e. residue computation). Moreover, with this configuration, no changes are needed depending on whether the user is working only with keypoints or with the entire point cloud. Furthermore, *ElementSet* contains more relevant information about point cloud which can be used during the registration process:

- **xmin, xmax, ymin, ymax, zmin, zmax**: the bounding box of the point cloud.
- **diagonal**: the diagonal of the bounding box.
- **center**: center of the bounding box.

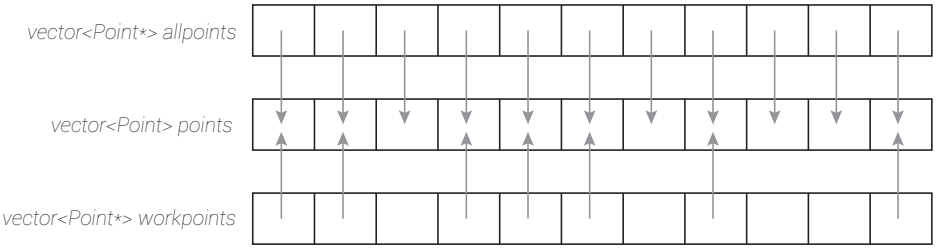

Figure 3.3: Structure of three vectors to store and access to the point cloud in each *ElementSet*.

- **MMD**: Mean Minimum Distance between points. This value is a resolution measure of the model:

$$\mathrm{MMD}(\mathcal{A}) = \frac{\sum(d(a_i, nna_i))}{|\mathcal{A}|}, \forall.\mathcal{A} \tag{3.1}$$

  where nna_i is the nearest neighbor point of a_i. We used this value to define the error thresholds, among other parameters, in order to have a software that does not depends on the input model size.

- **normals**: vector of normals associated with the point cloud.

- **dataStruct**: data structure used for residue computation and nearest neighbor searching.

3.2.2 Class *Data*

Class *Data* encapsulates all information about the model views —*ElementSet*—, movements and parametrization. This object is shared for all Pipeline classes as a pointer. We decided on this implementation in order to avoid unnecessary copies of the same information. Aligning huge models require taking into account memory management. This class includes:

- ElementSet **A**: the target point cloud.

- ElementSet **B**: the source point cloud.

- Motion **coarseTranform**: the aligning transformation obtained after the Searching Strategies step.

- Motion **fineTransform**: the aligning transformation obtained after the Refinement step.

- parameters **params**: contains all the values governing the behavior of the algorithms used, read from the parameters file.

3.2.3 Interface implementation

In order to maximize the modularity and provide an easy way to add more algorithms, we implemented the Pipeline with interface classes, and each Pipeline method is made as an adapter. For external classes —3rd party libraries like PCL or CGAL— we implemented a set of converter classes that transform our data types —points, descriptors— to 3rd party data types, and vice-versa.

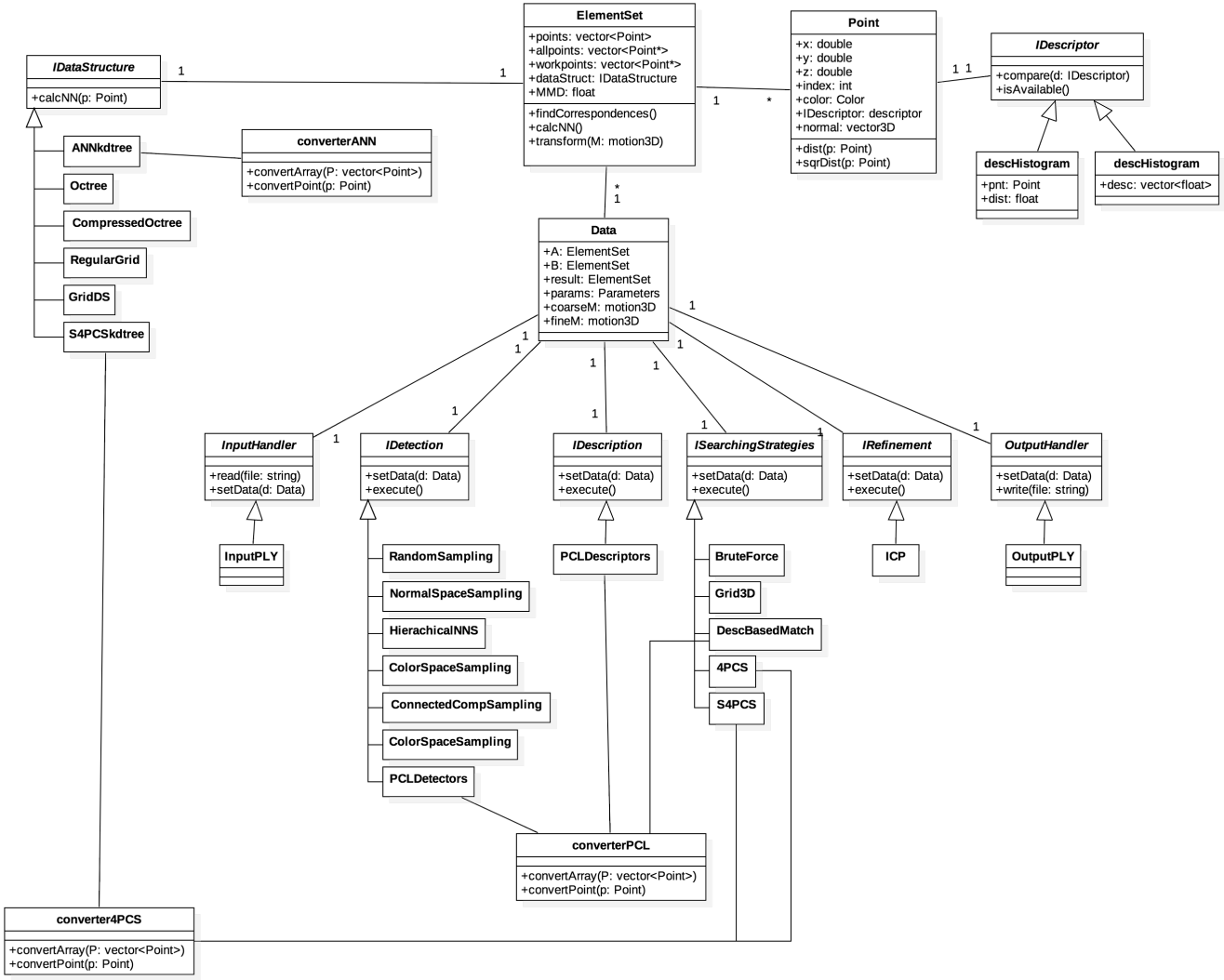

Figure 3.4: Simplified class diagram of our Registration Pipeline. We show the main classes and its relation as well as the available methods in each step of the Pipeline.

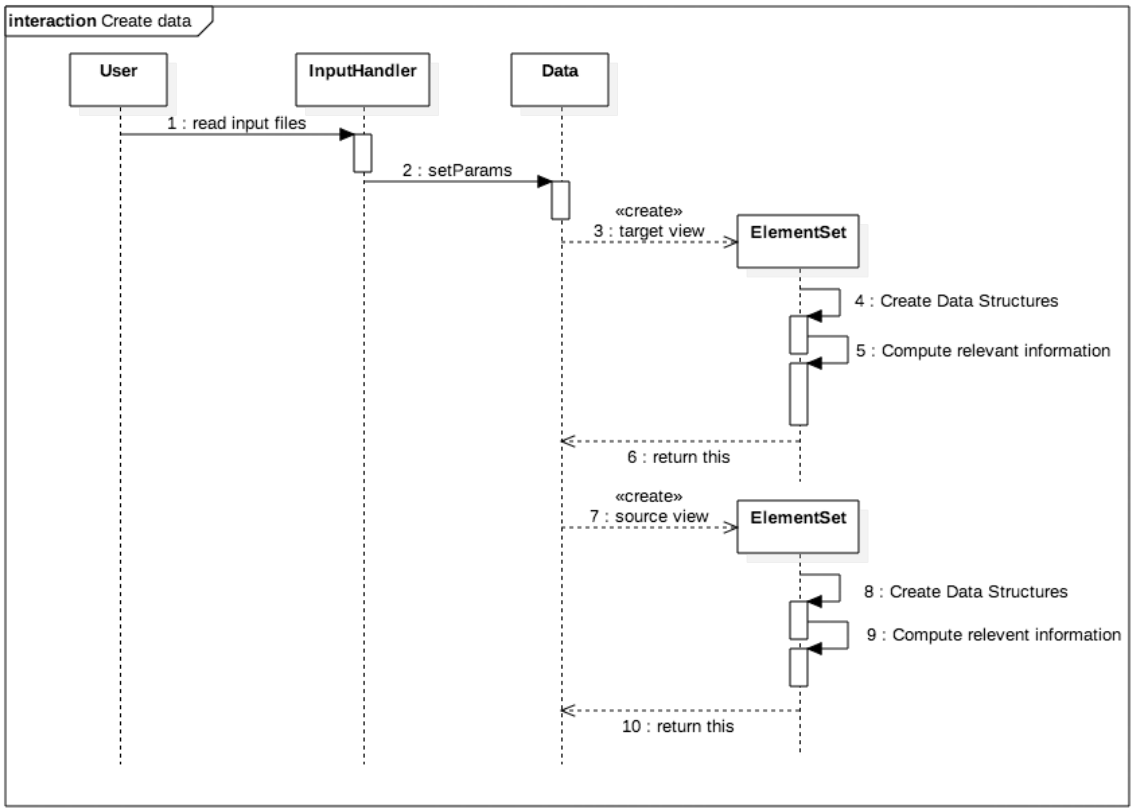

Figure 3.5: Simplified sequence diagram of data creation.

3.2.4 Input data

The first step of our Pipeline consist of preparing all information for the registration process. Besides reading input files —views to be aligned— other geometric information must be computed. Thus, as shown in Figure 3.5, the registration process begins by reading input files using InputHandler classes. In our case, we use the *InputPLY* implementation but other derived classes can be added to read files in different formats.

Because many parameters are needed for all available methods in the Registration Pipeline, we decided to use only one input file where the user can specify their parameters. We decided to use XML language because it is easily extendable and it is popular in the community. See Figure 3.6 for a complete example. In this parameters file the user can define input and output file paths, the method used in each step of the Pipeline and its parameters, and the data structure used for residue computation. When all this information is read, a Data object is created with it, building all data structures needed to store the information and for computing the alignment.

```
<params>
    <files>
        <realData>true</realData>
        <infile>target.ply</infile>
        <infile2>source.ply</infile2>
        <infileTemp>temp.ply</infileTemp>
        <outfile>out.ply</outfile>
        <outres>log.txt</outres>
    </files>
    <methods>
        <detection use="false">
            <method>NormalSpaceSampling</method>
            <properties>
                <nSamples>1000</nSamples>
            </properties>
        </detection>

        <description use="false">
            <method>SHOT</method>
            <properties>
                <radiusNormalFactor>5</radiusNormalFactor>
                <radiusSearchFactor>20</radiusSearchFactor>
                <nNeighbours>100</nNeighbours>
            </properties>
        </description>

        <searchingStrategies use="true">
            <method>Super4PCS</method>
            <properties>
                <nCells>3</nCells>
                <thrsFactor>1</thrsFactor>
            </properties>
        </searchingStrategies>

        <refinement use="true">
            <method>ICP</method>
            <properties></properties>
        </refinement>

    </methods>

    <dataStructure>
        <name>GridDS</name>
        <params>
            <param name="internDS" value="kdtree" />
        </params>
    </dataStructure>

    <generalProperties>
        <percOfPoints>1</percOfPoints>
        <nnErrorFactor>2</nnErrorFactor>
        <percOfNoise>0</percOfNoise>
        <normalizeModels>false</normalizeModels>
    </generalProperties>
</params>
```

Figure 3.6: An example of a parameter file used as input in our Registration Toolbox.

3.2.5 Detection

If the Detection step is used, the system extracts the reference keypoints from the original point cloud using a certain detection method. For this step, we implemented some detectors and also offer those implemented in PCL. A brief description follows:

- **Random Sampling** which randomly picks up points from the original point cloud.
- **Normal Space Sampling** [100] that gathers in buckets the points according to the position of the normals in angular space, and then uniformly selects keypoints across the buckets.
- **Hierarchical Normal Space Sampling** [47] which extends Normal Space Sampling by grouping points hierarchically according to the distribution of their normal vectors. This permits to navigate through the huge search space taking advantage of geometric information and to stop when a sufficiently good initial pose is found.
- **Color Space Sampling** is a new method that we introduce in the Registration Toolbox. It works as NSS but using color coordinates instead of the normal vector. This method is useful for datasets acquired with RGBD cameras, like Microsoft Kinect.
- **Connected Components Sampling** is a contribution that we present in this thesis. It gathers the points in connected components. In this case, the criterion to consider a point in a certain component is not a connecting edge, as in graphs, but the distance being less than a certain threshold. Then, uniform sampling is performed across the components.

Furthermore, there are more detectors available from PCL: **Uniform Sampling**, **Harris3D** [109] and **ISS** [126] among others.

As shown in Figure 3.7, if Detection is used, the system creates a detector object, instantiating the corresponding derived class from IDetection interface. Then a *execute()* method is called and keypoints are computed. Then, the *workpoints* vector is filled with keypoints. From then on, only keypoints are used for the rest of the Pipeline, except for residue computation.

3.2.6 Description

The Description step consist of finding descriptive features of each point in order to use them to find correspondences in the next step of the Pipeline. As we thoroughly reviewed in Chapter 2, different functions can be used as a descriptor. The majority of methods are based on geometric constrains around the points, like Spin Image or SHOT, but other approaches exist [110].

If the Detection step is used, the system only describes the keypoints located in the *workpoints* vector. If not, the whole point cloud is described (*allpoints* vector). Although Description can be used with the entire point cloud, using it with detected keypoints increases the performance and reduces the total registration time. This happens because non-relevant points are discarded and

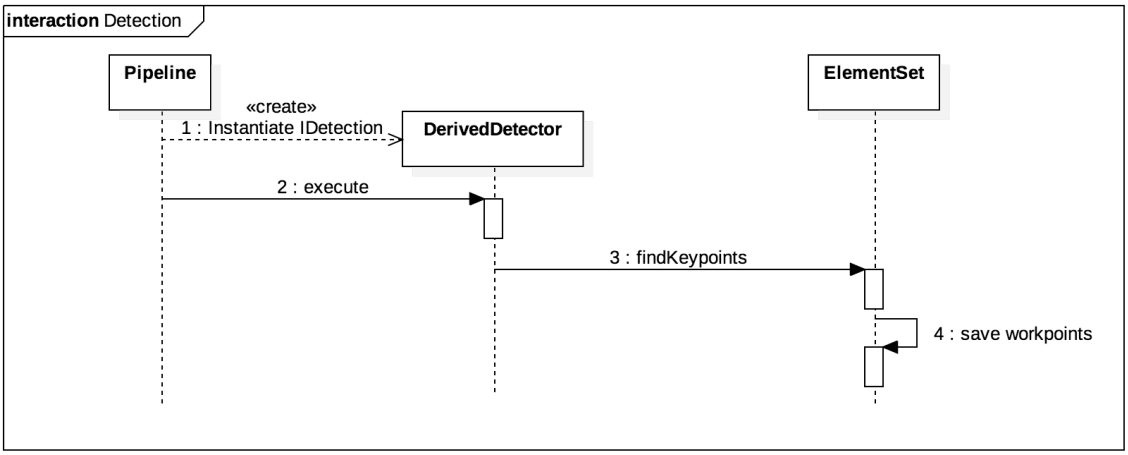

Figure 3.7: Simplified sequence diagram of the Detection step.

no longer used in the subsequent steps. Both steps —Detection and Description— are closely related in terms of feature comparison. Therefore, there are methods that use the same features for both detect and describe points [126].

In our 3D Registration Pipeline, all methods in the Description step are obtained from PCL, most of them reviewed in Chapter 2: **Spin Image** [65], **SHOT** [116], **3D Shape Context** [69], **FPFH** [102] and **Principal Curvature** [54] among others.

Figure 3.8 shows the sequence diagram of the SHOT descriptor method. Here, we use an external implementation of SHOT, from PCL. The process begins creating a SHOT instance of *IDescriptor* class. As we are using an external library, our model need to be converted to PCL-based point cloud using *converterPCL* class. Then, a *PCL::SHOTEstimator* object is created and used to compute SHOT descriptor of each point. Thus, PCL-based descriptors are transformed-back to our data types, and incorporated to the *ElementSet* object.

In this case, any other PCL descriptor can be used. The only work that needs to be done is to implement an adapter class, similar to the SHOT class, which calls the correspondent object from PCL and its *convertPCL* to transform data types. In order to implement a new descriptor, the Developer only needs to create a class that implements the *IDescriptor* interface.

3.2.7 Searching Strategies

Our Pipeline implements four different Searching Strategies. First, a **Brute-Force** approach is available for completeness. Then, the 4-point Congruent Set (**4PCS**) [17] and its improved version Super4PCS (**S4PCS**) [84] are included. We implemented two different Searching Strategies. On one hand, a **Grid3D** from [94] which is a hybrid strategy where the rotation information is given by a hardware solution —An smartphone attached to the scanning system [93]— and the

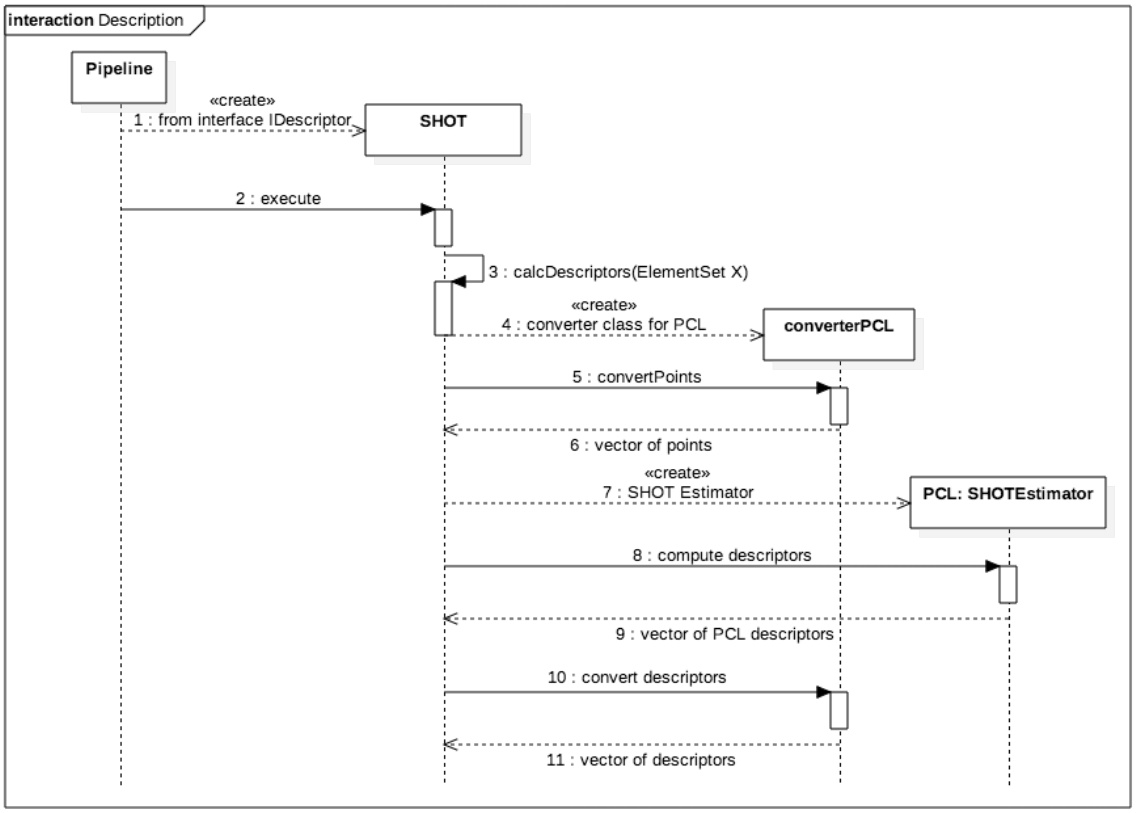

Figure 3.8: Simplified sequence diagram of the SHOT descriptor using PCL.

translation movement is solved with our method. We also introduced a new hierachical version of Grid3D [98]. On the other hand, we implemented a **Descriptor-based Matching** method [61], which is the only that uses descriptor values to find correspondences. Figure 3.9 shows a sequence diagram of our matching algorithm based on descriptor features.

As we detailed in Chapter 1, three correspondences are needed to determine a rigid motion. In order to check if theses correspondences produce a good alignment, the residue between the target view and the moved source view needs to be computed. In order to do this, certain nearest neighbor or range searching data structures can be used. In our toolbox there are different data structures to use. As we show in Figure 3.4, data structures are implemented as the rest of the Pipeline —with derived classes from a common interface—. This configuration allows future extensions with more data structures, as we throughly review in Chapter 4.

3.2.8 Refinement

This final step consist of computing the fine movement which aligns the result from previous step and the target view. As we explain in Chapter 1, the main characteristic of refinement methods

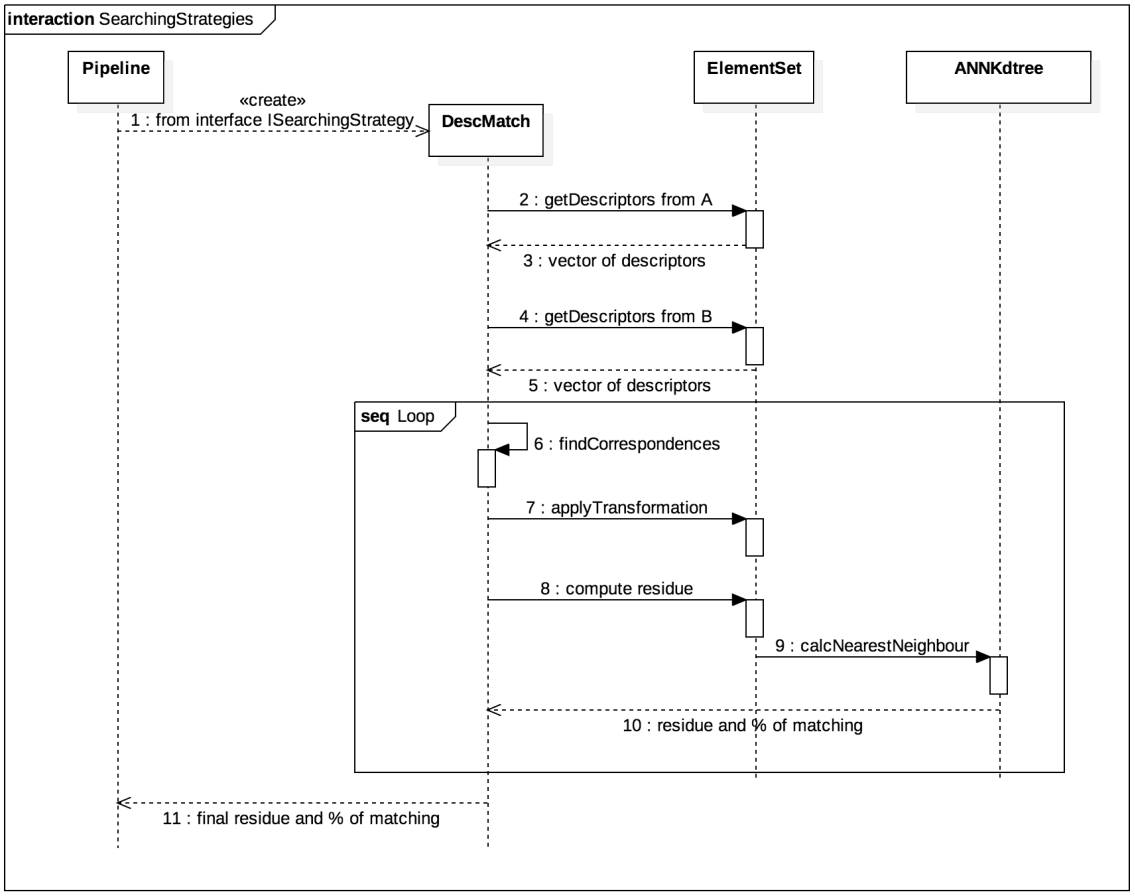

Figure 3.9: Simplified sequence diagram of SHOT descriptor using PCL.

is the ability to achieve a precise alignment from coarsely aligned views. If a coarse alignment is not provided, refinement methods fall in local minima providing wrong registrations [97] In our Pipeline, the Rusinkiewicz implementation of **ICP** [100] is available. However, the user can also use the PCL version, or include any other method, but ICP remains as the most popular refinement method nowadays, not only for its performance, but for its robustness. Figure 3.10 shows a sequence diagram of our Refinement step.

3.3 Public database

Given the diverse origin of the contribution to this research field as well as the divergence in focus between papers dealing with separate parts of the matching Pipeline, most contributions are evaluated with particular datasets that are not accessible to the research community. Although some publicly available datasets do exist [3, 26, 29, 30] and some of them are widely used by the community, not much background information is available for result comparison. Specifically:

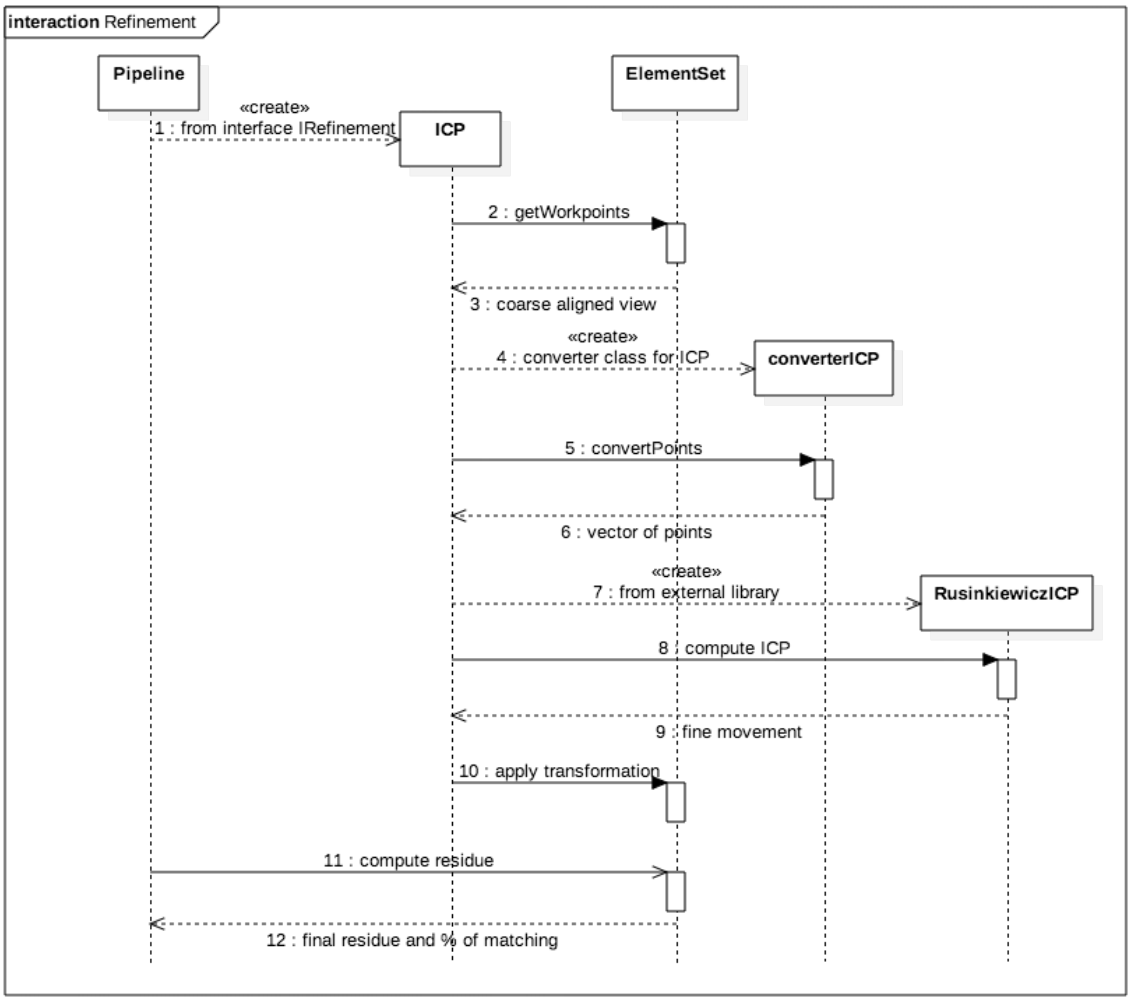

Figure 3.10: Simplified sequence diagram of ICP execution in the Refinement step.

1) No alignment ground truth is provided. 2) Similarly, no assessment on the level of noise or on the final overlap to be achieved between sets is given 3) No intermediate data concerning the different steps of the matching Pipeline is included. For example, researchers developing new search strategies (at the later part of the matching Pipeline) need to either first implement state-of-the-art descriptors or not use them at all. Furthermore, the distinction between synthetic and real data is crucial when evaluating the performance of algorithms, as data with noise or a low degree of overlap presents a much more challenging problem.

As a part of the Toolbox, we include a publicly available database that aims at overcoming these limitations and provide valuable tools for researchers working in the coarse matching field. The main characteristics of our database are:

- It contains datasets targeting different aspects of the matching problems. Special attention

is given to noise and overlap.

- Ground truth data is included as well as measures on the quality of final registration (overlap percentage, residue and pre-aligned data sets).
- The data it contains makes it possible to test different parts of the Pipeline separately. For example, descriptor data is provided as well as output after ICP execution.
- Several practical applications and problems are targeted, so, for example, we include, as well as the usual data where rigid motion needs to be determined, sets where the rotation is provided separately. This makes it possible to test methods that determine the two parts of the motion separately [71].
- Data from real application problems allow to test algorithms in increasingly challenging scenarios.
- It can be accessed online at: http://eia.udg.edu/3dbenchmark

3.3.1 Database description

The database contains several models that cover a variety of scenarios in 3D coarse registration. From low to high complexity, we provide two processed models and four "real application" models. The processed models are modified versions of the well-known Buddha and Bunny models from the Stanford Repository [3]. We include five of the original views for every model as well as data concerning correct alignment results between them. Additionally, we present modified versions of the Bunny dataset with varying levels of Gaussian noise added. Concerning the real datasets, the first one corresponds to 5 views of a frog-shaped flowerpot from [16], scanned with Microsoft Kinect. The second one is the Bust model, which is a reconstructed mannequin using a structured-light system [95]. This model is more challenging due to noise in acquisition and includes the possibility of determining the rotation and the translation part of the motion separately. Then, the Joints model present a particular and challenging problem of interest in industrial settings [27]. The data presented corresponds to an unstructured heap of manufactured parts along with a model part that has to be located (possibly many times over) in the heap. Finally, the most complex one is an scan of a part of the city of Bremen [4]. The two views of this model were acquired with a LIDAR[1] system and cover a massive space including a cathedral.

In the remainder of this section we describe the general layout of the database and all the characteristics shared by all models. In the following subsections we provide specific details on each particular dataset.

All models are provided in *.ply format. Moreover, we added the normal vectors computed at every point. We provide this information in order to offer a common starting point for algorithms

[1] Light Detection And Ranging

that use normals as their main geometric primitive. Such algorithms include descriptor algorithms such as [65] [116] but also algorithms that use Fourier analysis of normal distribution to determine matching [71]. Additionally, all models include:

- Different consecutive views in .*ply format with different overlapping ratios. For each view, we provide the non-aligned view as well as its properly alignment pose. All computations were performed automatically and re-checked manually.
- 4x4 transformation matrices in homogeneous coordinates to align all views.
- Alignment residue computed using the Root Mean Squared Distance (RMSD) criterion.
- Overlap ration for correctly aligned view. This is computed as the percentage of paired points after coarse matching and ICP were successfully run. A point was considered matched if its nearest neighbor in the other set was closer than $2 \times MMD$ where MMD stands for the mean nearest neighbor distance for the set (Mean Minimum Distance).
- All this information is publicly accessible online at *http://eia.udg.edu/3dbenchmark*.

3.3.2 Processed data

Two of the most well-known objects in the literature are the Bunny and Buddha models from the Stanford Repository. Both original datasets consist of several views with smoothed surfaces. These objects also appear to have undergone noise and outlier filtering. This type of data presents less challenging problems than real datasets because are processed and filtered.

The Bunny model is the simplest model with ≈ 37,000 points per view (Figure 3.11). All features are clearly defined, without noise, outliers or symmetries. Pairwise views present decreasing overlap increasing the difficulty of registration. Specifically, views *bun0-bun1* have ≈ 90% of overlap while views *bun3-bun4* have only ≈ 40% (see Table 3.1 for details).

For this model we also include a four-level noisy versions of three different views for more challenging tests (Figure 3.13). Gaussian noise was added to all the views with varying modulus. This modulus of the noise vector associated to each point was randomly chosen but had a limit that changed for each set. Specifically, in the first view this maximum was set to $1 \times MMD$. Values of $2 \times MMD$, $3 \times MMD$ and $4 \times MMD$ were also considered in order to make up the three remaining modified views.

The Buddha model is a bit more challenging than the Bunny because of its bigger size (≈ 75,000 points per view). Additionally, it has much smaller details and a higher degree of symmetry. These represent challenges both for Descriptor functions as well as for Searching Strategies. Furthermore, the base of the figure is a rounded pedestal which induces quite a noticeable source of symmetry and hampers normal space analysis.

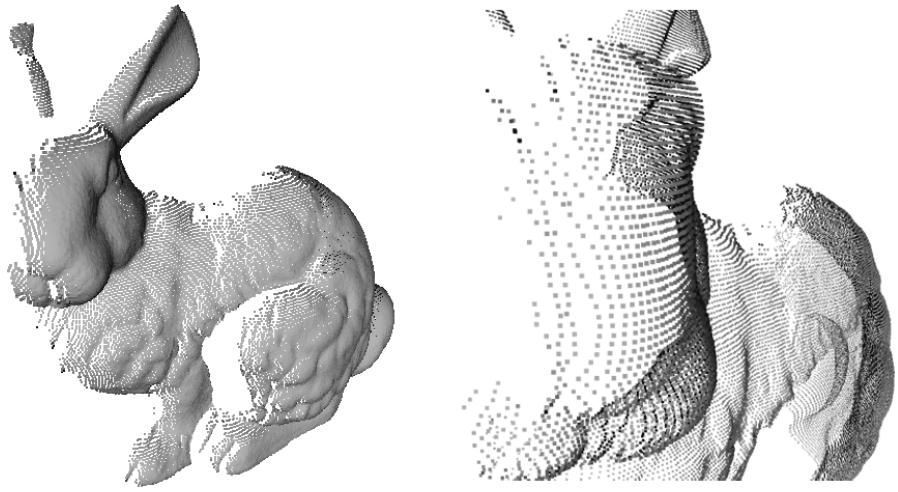

Figure 3.11: Left: *bun1* view. Right: Detail of *bun1* view. Notice that there are no noise or outliers.

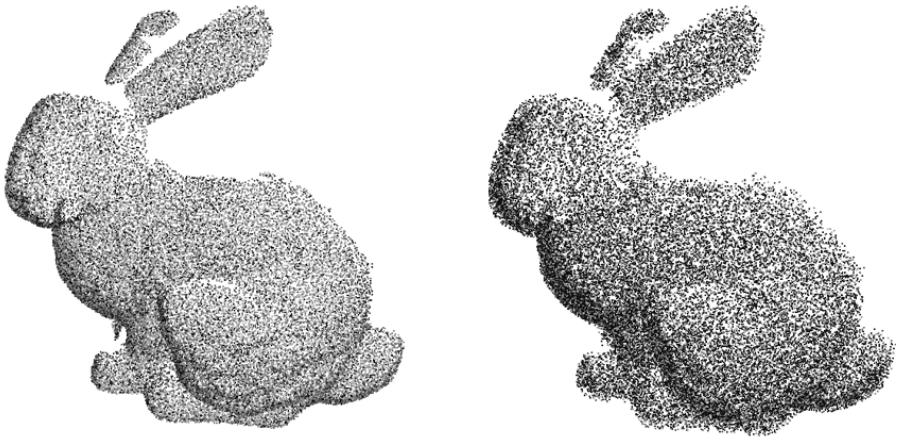

Figure 3.12: Left: *bun0* view with $1 \times MMD$ of Gaussian noise. Right: *bun0* view with $4 \times MMD$ of Gaussian noise.

3.3.3 Real data

Our real scanned data consist of a set of views from four different models that have been acquired using different techniques. The Frog model [16] is a flowerpot scanned with Microsoft Kinect [116]. The Bust model is a real-sized mannequin of a human body, scanned with a 3D structured-light system [93, 95]. The Joints model consists of an unsorted lot of metal joints acquired using a range scan (a laser and a single camera). This particular model, acquired with our own scanning system, was conceived for solving the bin picking problem, where a robot is expected to identify a certain part from a stack of unsorted similar parts. Finally, Bremen model from [4] consist on two views of the city of Bremen, acquired with a Lidar system. Figures 3.14, 3.15, 3.16 and 3.17

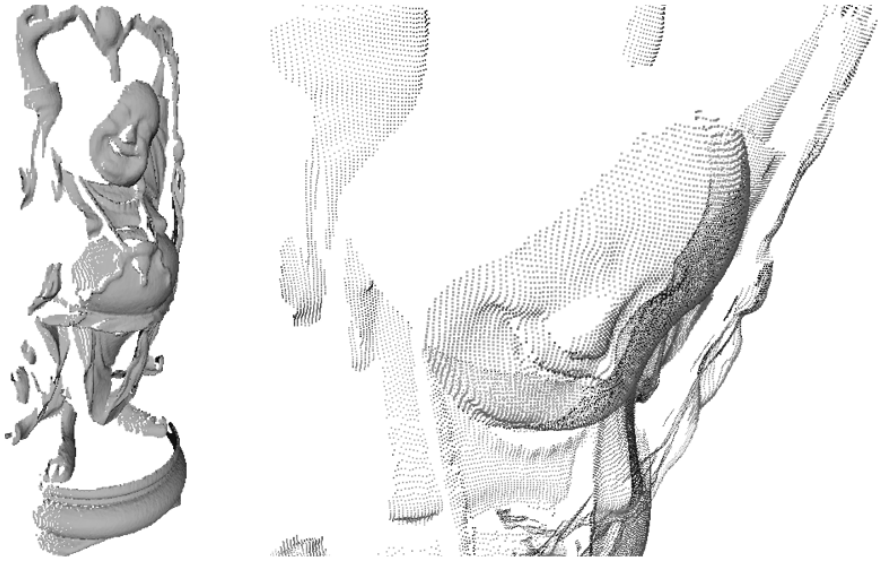

Figure 3.13: Left: *buddha0* view of Buddha model. Right: Detail of *buddha0* view.

Table 3.1: Example of data base results from processed data with a $MMD \approx 5 \times 10^{-4}$ for Bunny yhe model and $MMD \approx 3 \times 10^{-4}3$ for the Buddha model.

	Views	**Residue**	**Ovlp A-B**	**Ovlp B-A**
Bunny	0 - 1	3×10^{-4}	91.66%	89.91%
	1 - 2	3×10^{-4}	48.67%	79.07%
	2 - 3	5×10^{-4}	44.45%	27.53%
	3 - 4	4×10^{-4}	38.45%	48.30%
Buddha	0 - 1	2×10^{-4}	79.74%	85.16%
	1 - 2	2×10^{-4}	80.04%	89.28%
	2 - 3	2×10^{-4}	76.99%	90.84%
	3 - 4	2×10^{-4}	75.64%	79.10%

present example views of these datasets.

These datasets represent a more challenging scenario for any registration method, due to the presence of noise, the low overlapping ratios and the outliers.

The Frog model (see Figure 3.14) was acquired by Tombari et al. to test its SHOT descriptor [116] using a Microsoft Kinect camera. This is an small model (≈ 27,000 points) but with few overlap between views. Using this acquiring system, that mix color image with a depth sensor, color information is also available. There are 5 views for this model.

The Bust model was acquired using a structured light system that consists of projecting a pattern on a real object and tacking a photo of the scene [95]. Then, the differences between the original pattern and the captured one provide the 3D information. The views of this model contain ≈ 450,000 points. There was no post-processing step and the noise comes from acquisition system

Figure 3.14: One view of the Frog model and a detailed visualization.

(Figure 3.15). The overlapping ratio is ≈ 60%, depending on the view. All the characteristics of this model (noise, overlap and number of points) represents a challenging problem for registration algorithms. For this particular case we also provide a single translation version of data, where an approximation of the rotation is already computed using a gyroscope added to the scanning system [93]. This approximation reduces the final computation costs, because only a single correspondence needs to be found. As the estimation of the rotation is noisy due to the nature of the sensor used, this datasets aim at being useful for researchers who tackle the matching problem by determining rotations and translations separately [71]. The data provided allows to compare rotation determination algorithms to the data obtained by the sensor and also to prove the usefulness of robust translation determination algorithm with noisy rotation data.

A different registration dataset in the database is our Joints model (Figure 3.16). This scenario is intended for industrial applications such as quality control. Specifically, it is an instance of the "bin picking" problem, where a robot arm is expected to pick an industrial part from an unstructured heap of possibly defective similar parts. This data was obtained using a range scan composed by a laser and a single camera and presents abundant noise and outliers. We provide this model without any post-processing in order to offer a registration problem as close as possible to real application conditions, as well as an "ideal" model of the piece to be found within the heap. Although, many possible matches are possible the noise and the outliers make it a very difficult task. Providing a scoring function that ranks all possible matching candidates is also an interesting associated problem. Both Bust and Joints models are acquired using our own scanning systems.

The biggest model in the database is the Bremen model [4] (Figure 3.17). This dataset consist of two views of a part of the Bremen city with approximately 2 million points each one, acquired with a Lidar system. This type of scanners, which are designed to work outdoors, are able to cover huge areas with high levels of accuracy. However they are very expensive.

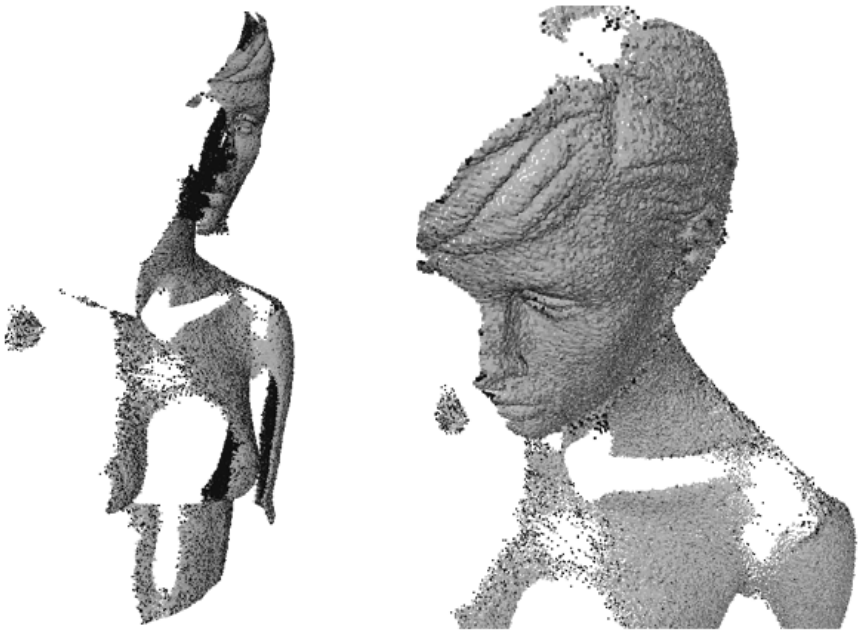

Figure 3.15: Left: *bust0* view of Bust model. Right: Detail of *bust0* view.

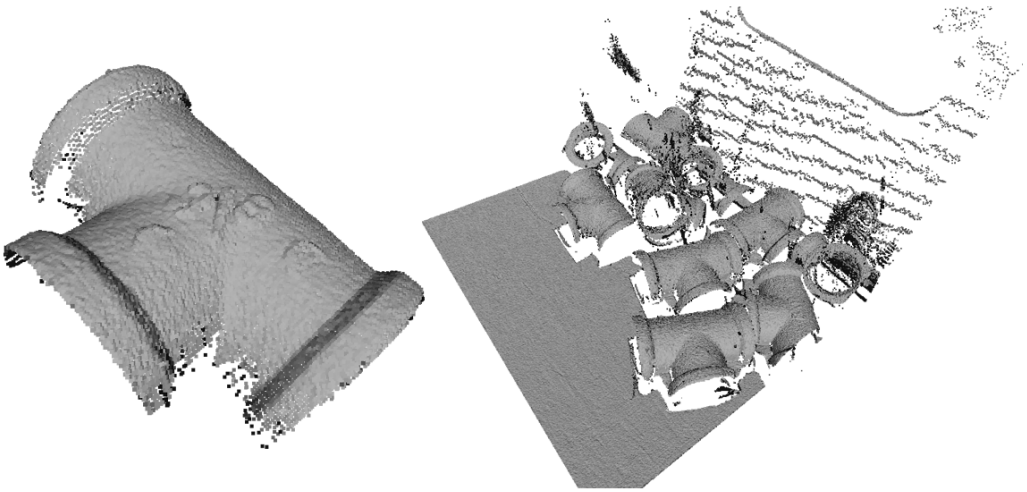

Figure 3.16: Left: Single *joint1* view. Right: Heap of unsorted joints.

3.4 Usage

We built this software focused on being used by two types of end users. On the one hand, our Registration Pipeline can be used without modifications, running the best combination of steps in order to align different views. These *Users* have different options within the software and

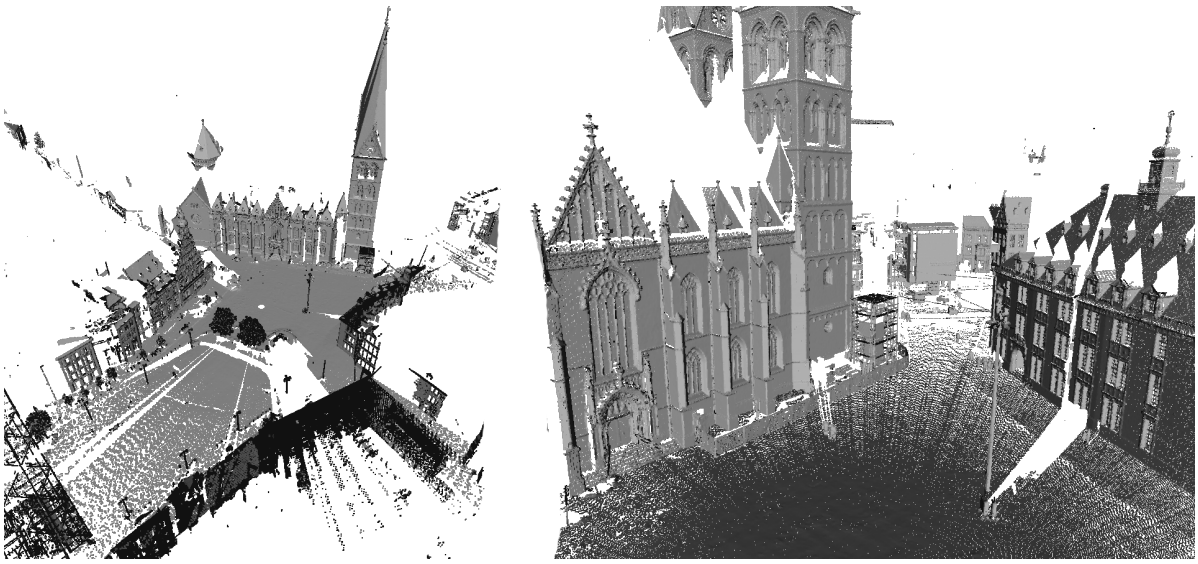

Figure 3.17: Left: one view of the Bremen model. Right: Bremen model in detail.

Table 3.2: Example of the information available in our data base. The aligning information with a $MMD \approx 0.59$ for Bust model and $MMD \approx 0.16$ for Joints model.

	Views	**Residue**	**Ovlp A-B**	**Ovlp B-A**
Bust	0 - 1	0.68	86.34%	53.94%
	1 - 2	0.66	72.28%	57.31%
	2 - 3	0.65	61.16%	60.06%
	3 - 4	0.66	61.69%	76.50%
Joints	heap - 1	0.14	4.27%	86.92%
	heap - 2	0.19	3.73%	73.60%

they can test all algorithms and choose the most appropriate in each case. On the other hand, there is the possibility to add other methods to the Pipeline. The *Developers*, can add their own algorithms easily, and test them alongside with our included methods. In this section we will present two usage cases —for *Users* and *Developers*— of our Registration Pipeline in order to exemplify both possibilities.

3.4.1 Registration of the Bunny model using NSS, 4PCS and ICP

This case presents a *User*'s execution example, using Detection, Searching Strategies and Refinement steps. Specifically, the selected methods are Normal Space Sampling (NSS), in order to filter the original point clouds and obtain the keypoints, 4-point Congruent Set (4PCS) which is used to find correspondences between views and align them and, finally, Iterative Closest Point (ICP)

to refine the coarse movement given by 4PCS. In this case, only one file needs to be modified, because the *User* will execute the Pipeline with the available methods. The parameters file needs to be prepared with the appropriate configuration to set up the Registration Pipeline. In the rest of this section we explain how this is done in detail.

Input and output files

The parameters file begins with input and output file paths definition. Inside the tag `<files>` we define the target view (`<infile>`), the source view (`<infile2>`), the intermediate file, which is used to store the source view after applying a coarse movement from Searching Strategies, the output file (`<outfile>`) where the final registered view is stored and, finally, the results text file (`<outres>`) that stores all other results in text format, like % of paired point and final residue between registered views.

```
<params>
<files>
    <infile>bunTarget.ply</infile>      # Target view
    <infile2>bunSource.ply</infile2>    # Source view
    <infileTemp>temp.ply</infileTemp>   # Intermediate view
    <outfile>out.ply</outfile>          # Final registered view
    <outres>results.txt</outres>        # Other numerical results
</files>
```

Method selection

The tag `<methods>` includes the information for all steps. For each of them, a dedicated tag is needed, where the properties of each method are defined. The method is selected with `<method>` tag and properties are defined with `<properties>` tag. In this particular case, Detection, Searching Strategies and Refinement are explicitly activated with the parameter `use="true"`. As the Description step is not used, this parameter is set to `false`.

```
<methods>
    <detection use="true">
        <method>NormalSpaceSampling</method>
        <properties>
            <nSamples>1000</nSamples>
        </properties>
    </detection>

    <description use="false"></description>
```

```
    <searchingStrategies use="true">
        <method>4PCS</method>
        <properties>
            <thr>0.1</thr>
            <nPoints>1000</nPoints>
            <delta>0.1</delta>
            <overlap>0.3</overlap>
        </properties>
    </searchingStrategies>

    <refinement use="true">
        <method>ICP</method>
        <properties></properties>
    </refinement>
</methods>
```

Data structure for residue computation

Different data structures are available for residue computation and can be selected with `<dataStructure>` tag.

```
<dataStructure>
    <name>GridDS</name>
    <params>
        <param name="internDS" value="kdtree" />
    </params>
</dataStructure>
```

General properties

Finally, general properties can be defined for different purposes. In this case, available properties are the percentage of points to be used for residue computation (`<percOfPoints>`), the error factor for correspondence searching (`<nnErrorFactor>`)[2], the percentage of noise that can be added to input views, for testing purposes (`<percOfNoise>`) and the possibility of normalize the input views size (`<normalizeModels>`).

```
<generalProperties>
    <percOfPoints>1</percOfPoints>
    <nnErrorFactor>2</nnErrorFactor>
```

[2] nnErrorFactor*MMD=correspondence distance threshold to consider a correct matching.

```
    <percOfNoise>0</percOfNoise>
    <normalizeModels>false</normalizeModels>
</generalProperties>
</params>
```

3.4.2 Adding a new registration method

Our Registration Pipeline has been conceived to be easy expandable by the community. We developed our code taking into account future inclusions in any step of the Pipeline. We named these use cases as *Developer* usages. In this section we explain a concrete example of adding a new registration method to our Pipeline. In this case, the *Developer* wants to add their own algorithm named *amazingMethod()* as a Searching Strategy, use ICP to improve the final result and compare its method with the other available algorithms in the Pipeline. As an initial requirement, the *amazingMethod()* needs to use the complete point cloud to compute the coarse transformation. Thus, only the Searching Strategies and Refinement steps are needed.

As explained in Section 3.2, we define each step as an interface (abstract class in C++) which is implemented by each method. Furthermore, as the *amazingMethod()* uses its own data types, a converter class is needed to work with both formats. Thus, the *Developer* has to create an implementation of Searching Strategies interface and a particular class to convert data types.

Converter class

This is a simple class to transform all data types needed to use the *amazingMethod()*, which are, in this case, only the point types. The definition file of *amazingConverter()* follows:

```
class amazingConverter {

public:
    amazingConverter();
    ~amazingConverter();

    vector<amazingPoint> cloud2Amazing(vector<Point*> *P);
    vector<Point*> * amazing2Cloud(vector<amazingPoint> AP);
};
```

Interface implementation

The implementation of Searching Strategies step must satisfy its interface, which has two mandatory methods: *setData()* and *execute()*. The former is used to access to the *Data* object shared overall Pipeline and containing all information —point clouds, descriptors, ...—. The

later executes the *amazingMethod()* from the *Developer*'s code. In this particular case, the *amazingMethod()* needs some parameters to work properly. These parameters can be defined in the parameters file, and recovered using *params* object stored in *Data*. We show below the *execute()* implementation of this case.

```
void ss_amazingMethod::execute(){

    amazingConverter aC;
    vector<amazingPoint> set1 = aC.cloud2Amazing(data->A->getWorkpoints());
    vector<amazingPoint> set2 = aC.cloud2Amazing(data->B->getWorkpoints());

    amazingMethod AM;
    AM.setTargetCloud(set1);
    AM.setSourceCloud(set2);
    AM.setStoppingThrs(data->getParams.ss_StopThrs);
    AM.setDelta(data->getParams.ss_Delta);

    data->coarseTransform = AM.computeAlignment();
}
```

Finally, the Refinement step is performed using ICP, using the approximation given by the *amazingMethod*. The aligned point cloud is stored in a file and the execution results are shown in the screen or stored in a text file.

Using our Registration Pipeline, the *Developer* not only can include its own algorithm, but they can easily compare it with the other available Searching Strategies, and also use other steps of the Pipeline to improve his method.

3.5 Examples of application

The main aim of this toolbox is to be used to compare new matching algorithms, but it can also be used to study other aspects of the coarse matching problem. In this section we illustrate how the presented database can be used to study commonly agreed upon "truths" of coarse matching algorithms.

3.5.1 Application 1: ICP needs a "good enough initial pose" to succeed.

The very distinction between coarse and fine matching algorithms is based on the accepted fact that ICP fails to converge or stalls at a local minimum if the initial pose it is provided with is "not good enough". It is, however, infrequent to see quantifications of what a local minimum looks like or exactly how good the initial pose needs to be.

In this experiment we run ICP with the Bunny and Bust datasets. For each of the two objects, one view was registered against its consecutive view. The original pose stands for the best possible initial alignment. We then perturbed this initial pose by rotating the second view along one of the three axes. We repeated the process independently for axes X,Y and Z and also by rotating along all the axes at the same time. Table 3.3 contains the summary of this experiment. Column *Deg. fail* shows the angle (in degrees) were ICP failed to converge to the global minimum for the first time. The table also shows how in some executions, ICP fell in local minimum (indicated in the table by an asterisk) while in other executions ICP was not able to converge. Notice that, once again, we were able to observe differences in behavior for processed and real data. Specifically, the Bunny dataset was much more robust to the perturbation of the initial pose than the Bust dataset. We also observe, how the resilience against this type of rotational perturbation increases when the total overlap between views is higher. As showed in Table 3.1, the overlap between *bun0* and *bun1* is higher ($\approx 91\%$) than *bun1* and *bun2* ($\approx 48\%$).

Table 3.3: Table of ICP test. An asterisk indicates that the algorithm stalled at a local minimum. The dash indicates that the ICP was not able to find any solution.

	Views	**Rot. Axis**	**Deg. fail**	**Ovlp B-A**
Bunny	0 - 1	X	50°	14.24%*
		Y	70°	12.78%*
		Z	40°	17.45%*
		XYZ	25°	9.98%*
	1 - 2	X	40°	Fail
		Y	60°	Fail
		Z	30°	4.21%*
		XYZ	20°	4.12%*
Buddha	0 - 1	X	20°	Fail
		Y	10°	Fail
		Z	10°	Fail
		XYZ	2°	Fail
	1 - 2	X	15°	Fail
		Y	15°	Fail
		Z	15°	Fail
		XYZ	3°	Fail

3.5.2 Application 2: Descriptor performance drops in the presence of noise

It is often claimed that noise in data negatively affects the behavior of shape descriptors. In this experiment we aimed at determining how much noise in data it takes to get well-established descriptor to fail. Specifically, we chose the well-established SHOT method [116]. We then run a search based on coupling points according to their descriptor value:

- A three point basis in the first set was chosen.

- Each point in the basis was tentatively matched to k neighbors in order of decreasing descriptor similarity.
- Once three correspondences were determined, distances between the points in the two basis were checked for consistency.
- If the two basis presented similar distances, then a rigid motion between the two sets was computed.
- ICP was used to complete the matching process. The percentage of matched points (also referred to as overlap percentage) and the residue between the two sets was computed.

In order to test the effect of noise, we use the sets with increasing quantity of noise described in Section 3.3.2. Table 3.4 presents the results obtained.

Table 3.4: Results of registration process with SHOT descriptor without ICP refinement. Timeout was set at 15 hours. The overlap presented is the best obtained (at the end of execution or at timeout). An asterisk indicates that the overlap obtained was not the best possible and, thus, the algorithm stalled at a local minimum.

Noise	**Residue**	**Ovlp A-B**	**k**
-	7×10^{-4}	97.32%	500
$1 \times MMD$	5×10^{-4}	94.58%	500
$2 \times MMD$	1×10^{-3}	36.72%*	500
$3 \times MMD$	1×10^{-3}	21.59%*	500
$4 \times MMD$	1×10^{-3}	22.10%*	500

The results show how in the absence of noise the descriptors-based search is able to find correspondences very quickly while achieving the total degree of overlap. Descriptors manage to discriminate points very well and we only need to consider a low number of possible correspondences k in order to obtain the best possible matching. As soon as noise is added to the data the behavior of the search suffers. For the less noisy set, the algorithm still manages to find the correct matching but needs to consider many more correspondences. For the remaining sets, containing more noise, the algorithm was allowed to run for 15 hours before being stopped. During all that time even when considering a very high number of correspondences, only local minimum were reached and the algorithm was unable to output the correct alignment for any of the three sets.

3.6 Experiments using the 3D Registration Toolbox

In order to test our Toolbox as a useful resource for the community, we performed some experiments to assess the behavior of our proposal and also test some of the reviewed registration

methods. This section is a real example of the Toolbox usage. We focused on testing Shape Descriptor functions, which is one of the most active areas in the literature.

A key factor for shape descriptors is their resiliency to some of the most commonly appearing scanning artifacts. Although extensive works [29] studying the performance of shape descriptors exist, we feel that an in-depth study on the effects of these artifacts for state-of-the-art descriptors for rigid registration has not yet been conducted. Consequently, in this section we present a comparison of 4 state-of-the-art shape descriptors. We use real and synthetic data in order to assess the effect of different levels of noise and different ratios of overlap in descriptor performance. We tested four different descriptors which are extensively used in the literature: SHOT[116], Spin Image[65], Fast Point Feature Histogram (FPFH) [101] and 3D Shape Context (3DSC) [69]. We use these approaches over four different models, both synthetic and real data. The synthetic data models are the Bunny, Buddha models from the Stanford Repository [3], which allows us to gain further insight in the problem by controlling variables such as the amount of noise included, and the real data are Bust and Joints models, from our own scanning system [97], which makes it possible to show the behavior of algorithms in a realistic setting.

3.6.1 Methodology

The state of the art of 3D registration is mostly focused on detection/description. Local shape descriptors provide a numerical representations of the shape of the object around surface points. This already challenging situation is made even more difficult by the objects being represented as a discrete set of points. These points might contain noise and, thus, fail to precisely represent the object they belong to. Consequently, descriptors that are robust to small quantities of noise in points are sought. Additionally, any matching algorithm typically has to deal with the two objects being matched not being exactly identical but instead presenting a certain area of overlap. This is the case, for example, of the problem of reconstructing an object from several views. Descriptors that are able to focus on significant common areas are desirable. Finally, scanners sometimes produce spurious "outlier" points that do not really represent any part of the objects but nevertheless complicate registration. This is the case for example for metallic objects causing laser scanners to output incongruous points due to reflections.

We have considered a widely used descriptor as is Spin Image along with three more recent descriptors. SHOT, is an example of a descriptor favoring local reference frames that shows a remarkable resiliency to noise [116]. FPFH is representative of histogram-based descriptors that obtains good results with sets of overlap as low as 45% [101]. 3DSC is a descriptor with a totally different approach as not only points in local neighborhoods are used. We believe that this selection covers the different approaches that have proven successful in one way or another in the recent development of shape descriptors.

3.6.2 Experimental evaluation

The descriptor-based coarse matching algorithm used in this evaluation is the following. Given two input point clouds $\mathcal{A}$ and $\mathcal{B}$, we select a *wide base* [17] $B_{\mathcal{A}}$ made up of 3 points from set $\mathcal{A}$. For each point in $B_{\mathcal{A}}$, we search for point correspondences in $\mathcal{B}$. Possible correspondences are sorted in terms of descriptor values. In order to discard obviously wrong bases from set $\mathcal{B}$, the distances between base points are checked. The bases that pass this test, are used to compute an alignment movement. Then, the percentage of paired points and the residue are computed to evaluate the quality of the coarse alignment. Finally, ICP is applied in order to refine the movement and complete the registration process. All tests were performed using a PC Intel Core i7-2600 CPU 195 3.40GHz with 8 cores and 16GB of RAM on an Ubuntu 14.04 LTS operating system.

Some details on the design decisions for this algorithm follow. First of all, using wide bases makes the search more robust than using randomized selection [17]. In order to increase the quality of the bases $B_{\mathcal{A}}$, we select their points from a reduced group of keypoints obtained using a Intrinsic Shape Signature (ISS) detector [126]. The number of descriptor-sorted correspondences for each point is set to be, $k = 50$ for processed data, and $k = 100$ for real models, after preliminary testing. Finally, and in order to circumvent having chosen bases falling in non-overlap areas, the whole search is repeated for $r = 10$ different bases and only the best result is reported. The computational cost of the algorithm is thus, $O(r\,k^3\,m\,log\,n)$. With processed models the algorithm is able to find solutions after exploring few candidates (small k), therefore the computation time is rather small ($\bar{t} = 3min$). However, for more complicated objects, the algorithm tends to explore all possible candidates and, thus, the running time increases greatly ($timeout = 20hours$). For each model, we compute the Mean Minimum Distance (MMD) between its points in order to fairly apply random noise for the tests.

Processed Data

For this datasets we performed extensive tests using different levels of noise and overlap. For each model we considered two views. For each of them we produced 8 noisy copies. We added Gaussian noise with varying intensity. For each point, we added random noise up to $n * MMD$ with n taking eight values ($n \in \{0.2, 0.5, 0.9, 1, 1.5, 2, 5, 10\}$). We registered each model with noisy copies of itself and with noisy consecutive view. In the first case, only noise affects registration while in the second lower overlap is an added factor. Figures 3.18 and 3.19 show the results separately for every model. Continuous lines correspond to registrations of the same view while dashed lines stand for registrations of consecutive views. X-axes contain the level of noise in each set while Y-axes contain the percentage of matched points of the first view over the second view, after coarse matching (and before ICP). Finally, red dots represent the cases where the approximation is not good enough for ICP to succeed. As the main aim of coarse matching algorithms is for this not to happen, we consider a lower "number of red dots" to be a measure of algorithm success.

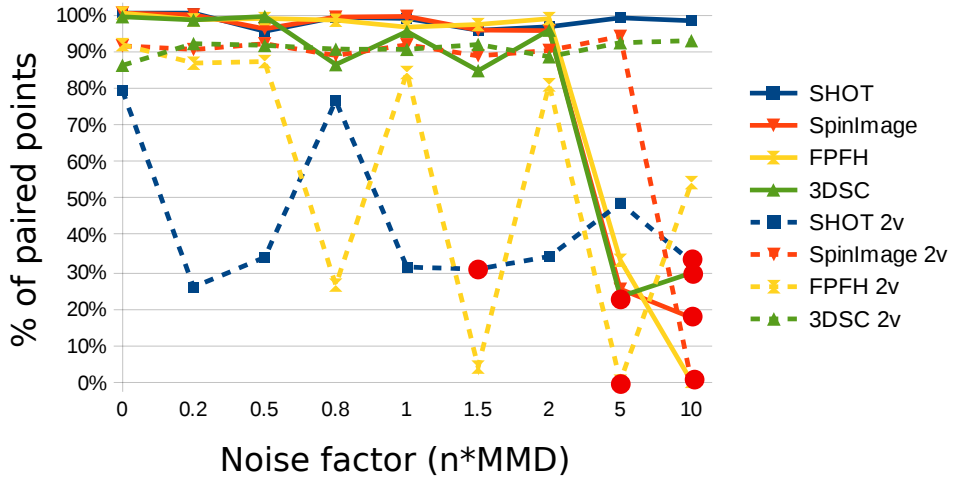

Figure 3.18: Results with Bunny model.

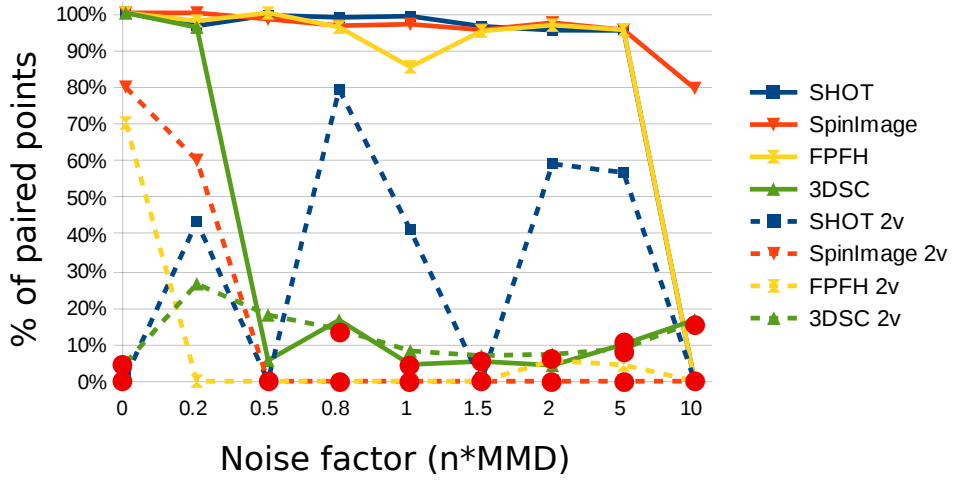

Figure 3.19: Results with Buddha model.

We observed that the overlap between models plays a more crucial role than noise. All descriptors performed well even under high amounts of noise when objects were registered to (noisy) copies of themselves (see full lines in Figures 3.18 and 3.19. Regarding whether or not ICP was able to converge to the optimal solution after the descriptor based algorithms, SHOT succeeded in all 27 registrations, FPFH failed in 1 of them, Spin Image in 2 and 3DSC in 5. However, when different views of the same object were considered the performance decreased. Lower overlap decreases the areas in the two objects that are actually useful for matching.

Consequently, descriptors must do a better job at singling out areas that do actually correspond to matching parts of the two objects. Regardless of this, the random choice of the bases $B_{\mathcal{A}}$ plays a significant role. Whenever $B_{\mathcal{A}}$ is chosen to contain a point outside the overlap region the matching is doomed to fail. Consequently, the figures should be interpreted in terms of general tendencies and not particular performances. These factors produce the unstable results depicted in figures 3.18 and 3.19 on the dashed lines. Specifically, SHOT failed to produce usable approximations for ICP in 7 of the 27 cases, 3DSC in 8 and FPFH and Spin Image in 9. The higher number of problems were detected in the Buddha model were 22 of the 36 registration run by all descriptors failed. To sum up, SHOT obtained the best results in both tests. Even with the problems detected for noise and low overlap most approximations (160/202) were enough for ICP refinement. Regarding data, complex objects with symmetries such as the Buddha proved to be the most challenging.

Real Data

In order to test the descriptor methods in realistic scenarios we performed a set of experiments where pairs of views of the Head and Bust models were registered. The Joints model case is substantially different because, instead of align two different views, we try to identify a certain shape in a unsorted heap. In these tests, noise is inherent to the scanning process. Table 3.5 shows the overlap between the views, the overlap after coarse registration and whether or not ICP succeeded. For the Joints model we provide both overlapping ratios to notice the size difference of this particular case.

The performance of descriptor methods with real data was worse than with synthetic data. Positive results were not achieved for all models. Although we set timeout to be 20 hours and increased the number of candidate to $k = 100$, in some cases, descriptors are not able to find a good approximation or just they do not find any result. In any case, these times are too large to be considered usable in real applications. Concerning descriptor comparison, best results were obtained by 3DSC and SHOT. FPFH obtained lower pairing ratio after coarse matching but succeeded in their main goal of producing an alignment that ICP could use to produce the correct fine matching, but only in Head model, which is smaller than Bust.

3.6.3 Experiment conclusions

In this section we analyzed the performance of 4 state-of-the-art descriptors when dealing with common scanning artifacts. In general terms, SHOT proved to be the most stable for synthetic data (failing only in 7 out of 54 registrations). In these cases, overlapping between views becomes more important than noise, because with single-view tests, almost all methods succeed. Additionally, objects presenting symmetries such as the Buddha proved to be more challenging to register as seen by the failure rate of 22 of its 36 2-view synthetic registrations. Concerning real data, 3DSC and SHOT obtained the best initial approximation. Here, the intrinsic noise from the scanning system becomes more problematic. Descriptors need bigger searching radii, and

Table 3.5: Real data results. Original overlap is given by two values: $\mathcal{A}$ respect to $\mathcal{B}$ and vice-versa. The paired points are only $\mathcal{A}$ over $\mathcal{B}$.

	Method	**Orig. Ovlp**	**Pair. Pnts**	**ICP**
Head	SHOT	84% / 54%	33.18%	✓
	SP		2.13%	✗
	FPFH		5.75%	✓
	3DSC		75.15%	✓
Bust	SHOT	86% / 53%	23.12%	✓
	SP		3.29%	✗
	FPFH		Fail	✗
	3DSC		50.47%	✓
Joints	SHOT	4% / 73%	31.24%	✓
	SP		Fail	✗
	FPFH		Fail	✗
	3DSC		Fail	✗

more candidates are needed to achieve good alignment results. The need of good methods able to deal with raw data in a reasonable amount of time has been shown to be an open problem for the community as only two descriptors were able to output results usable for ICP within the 20 hours of time limit set.

CHAPTER 4

GRIDDS, A HYBRID DATA STRUCTURE FOR RESIDUE COMPUTATION IN POINT SET MATCHING

Dedicated data structures can be used in order to speed up residue computation. However, the comparison of asymptotic cost and runtime performance may lead to discrepances. For example, should two data structures with the same asymptotic cost perform similarly? If a data structure takes longer to set up, how much does this hamper its overall performance? Should one data structure present a really bad worst-case running time and how often does this worst-case occur? Our aim in this chapter is to study both the asymptotic cost and runtime performance of **Compressed Octrees** [20, 106], **KDtrees** [22, 55], their Box-decomposition variant (**BDtrees**)[22], and **Regular Grids**[47]. We present detailed experiments on these data structures, both using synthetic and real data in Section 4.2. Then, we present a new data structure named GridDS that combines the best features of Regular Grid while also being able to take advantage of the strengths of other data structures.

4.1 Data structures

For this study we selected four different data structures, with different implementations, either for their interest as range searching or nearest neighbor data structures or because they were used to solve the residue computation in practice: a) KDtree [55] is a well-known nearest-neighbor data structure which is additionally used in two publically available implementations for point-set matching algorithms [84, 100] with implementations at [1, 2], b) their Box-decomposition variant called BDtree [22], c) the Regular Grid [47] and d) the well-known Octree [57] and its compressed version [20].

In the rest of this section we briefly introduce these data structures providing the expected

asymptotic costs for building the data structure and computing the residue.

4.1.1 KDtree

The KDtree [55] is a binary tree where each node represents a hyperplane dividing 3D space. In each level, a division is made according to one of the 3D axes. The points in the parent node are sorted according to the selected coordinate and the splitting value is used as the division point organizing the points in the left and right son nodes. For a detailed explanation on splitting values see [11]. This operation is repeated with all coordinate axes, until leaf nodes (i.e. nodes with less than a previously fixed number of points) are obtained.

KDtrees are commonly used for residue computation in point-set matching algorithms [17, 84, 100]. In this section we tested two different implementations: the Approximate Nearest Neighbor Library (ANN) [10] from [21], because it is widely used in the research area and a improved version from Super4PCS method [84], because it has been deeply tailored for the matching problem and is used in the fastest matching algorithm known to date.

Concerning the subdivision rules used by the analysed KDtrees, Super4PCS uses the median splitting rule that keeps the height of the tree logarithmic, while ANN uses the sliding midpoint rule that focuses on avoiding subdivisions with very high aspect ratio.

Building cost

The cost of building the data structure for a set of points $\mathcal{A}$ with $|\mathcal{A}| = n$ can be viewed as the cost of performing n insertions of points in the data structure. Additionally, as the points are previously known, they can be pre-sorted in each coordinate and binary search can be used for splitting computations. Consequently, the cost of each of these insertion operations is that of traversing the tree to the leaf node containing the new point and possibly splitting it once into new nodes (if the subdivision threshold was surpassed by the addition of the new point). Thus the construction cost is $O(nH)$ where H stands for the height of the tree. The height of the tree can be seen to be $O(\log n)$ [10, 21] by using the median splitting rule. Thus, the cost of building the structure for our particular problem is $O(n \log n)$.

Residue computation cost

In order to compute the residue between two point sets $\mathcal{A}$ and $\mathcal{B}$ with $|\mathcal{A}| = n$ and $|\mathcal{B}| = m$ we need to search for the nearest neighbor in $\mathcal{A}$ for every point in $\mathcal{B}$ and then discard those that are further than ε. Consequently, we need to perform m nearest neighbor searches in $\mathcal{A}$. The cost of each of these searches can be shown to be $O(\log n)$ [56]. Intuitively, we need to traverse the tree to where the query point would be inserted and then carefully check that the nearest neighbor does not lie in a neighboring node (see [56] for details). Consequently the cost of computing a residue is $O(m \log n)$.

4.1.2 BDtree

The BDtree or Box Decomposition tree [22] is a data structure based on a hierarchical decomposition of space, similar to KDtree. BDtree subdivides the space into regions defined by axis-aligned hyper rectangles that are fat, meaning that the ratio between the longest and shortest sides is bounded. We used the implemented version from the Approximate Nearest neighbor Library [10].

Building cost

The BDtree has the same building cost than the KDtree ($O(n \log n)$), where the height of the tree is shown to be $O(\log n)$. Although the process is more complicated than for KDtrees, as relations between cells (defined as "stickiness") and the operations they undergo during construction (named "splits" and "shrinks") are somewhat more complex, the asymptotic computation costs remain the same. In all performed tests, BDtree building time was approximately twice as slow than that of the KDtree.

Residue computation cost

Similarly to KDtrees, the cost of residue computation stands for m calls to the nearest neighbor search and thus is $O(m \log n)$.

4.1.3 Regular Grid

This data structure, introduced in [47] and also used in [94], stands for a regular 3D grid that divides an axis aligned bounding box into regular cells. Each cell contains a set of unsorted points. The structure is created by considering the section of each coordinate axis contained in the bounding box and dividing it into k intervals where $k = \lfloor(\sqrt{\lceil 3 \rceil} n)\rfloor$. Then the cartesian product of these intervals is considered and a set of k^3 regular rectangular prism cells is obtained. The goal is to have roughly as many cells as points in the set so each point would correspond to one cell in the best possible distribution of points.

Building cost

Each point is inserted into its corresponding cell, where each cell contains an unsorted list of points. Consequently the insertion cost of a point in a cell is $O(1)$ by just adding it at the end of the list of the cell it belongs to. The cost of locating a cell is also constant as the grid allows direct access to all its cells. Consequently inserting a point in the data structure takes $O(1)$ and the cost of building the data structure for n points is $O(n)$.

Residue computation cost

In order to perform residue computations with this data structure, we perform range searches at distance ε and iterate over the output (if any) until we find the closest point. This data structure is simple to implement and build, but this simplicity has the downside that the regular subdivision of space created by the structure might not work well in point distributions that present large concentrations of points.

Even with the assumption that any disc of radius ε overlaps at most a constant number of cells (an assumption that we feel is reasonable as ultimately ε represents a measure of imprecision in data), the number of points contained in these cells might be very large. Actually, if we consider a distribution with 8 points in vertices of a cube and the rest of them inside a disc of radius ε placed at the center of the cube, this value reaches $O(n)$. Hence, we define the "load factor F"' of the structure as the average number of points in non-empty cells. Consequently, the cost of computing the residue for a set of m points is $O(mF)$ which, as F is $O(n)$ totals to $O(mn)$.

Notice how, asymptotically, this data structure is not better than brute force. However, the $O(n)$ bound for F is not met in practice and the data structure performs well as we will see in Section 4.2.

4.1.4 Compressed Octree

The Octree [57] is a splitting-space data structure. Built as an 8-ary tree, the root node is associated to an axis aligned bounding box of the whole point set. Then the root node is divided into 8 sons of equal volume maintaining the aspect ratio. Each of these "octants" is subsequently subdivided in eight constant-aspect-ratio equal volume sons until nodes which contain less points than a predefined subdivision threshold are obtained. These final nodes are known as leaf nodes. Octrees, thus, produce a hierarchical regular-aspect-ratio decomposition of space. The main problem with Octrees is that, if two points are really close it might take an arbitrarily long branch to separate them inside the subdivision. This can result in massively unbalanced data structures.

In order to solve these unbalancing problems, Compressed Octrees were introduced [20]. Basically the cases in which branches longer than a fixed length are needed to separate few points are sidestepped by storing only the final nodes in the branch. Then, it is necessary to keep track of the size of each node in the Octree (this information was previously implicit in the situation of the node in the tree) but we obtain trees that have at most $O(n)$ nodes.

For these experiments we used the octree implementation from Point Cloud Library (PCL) [15] and our own implementation of a Compressed Octree. Due to the big performance differences between both, where our compressed Octree is many times faster, we decided to not include the PCL-Octree in the final results.

Building cost

The cost is, as in the case of ordinary Octrees, $O(nH)$, where H is the height of the tree. In this case H is $O(n)$ so the total construction cost becomes $O(n^2)$

Residue computation cost

In order to compute the residue we need, also, to look for points in set $\mathcal{A}$ within a ε range from all the points in set $\mathcal{B}$. Thus, the cost of residue computation is $O(mH)$ with H being linear, so it becomes $O(mn)$. As we saw with the regular Grid, this structure is asymptotically as bad as brute force. However, we will show in Section 4.2 that in practice the worst case is not attained and that the performance is actually quite close to that of the best data structures.

4.2 Experiments and results

We present runtime experiments both with a controlled synthetic scenario and with real data. All tests were performed using a PC Intel Core i7-2600 CPU 3.40GHz with 8 cores and 16GB of RAM on an Ubuntu 14.04 LTS operating system. We performed a certain number of executions in order to avoid being mislead by outlier results. Specifically, set $\mathcal{A}$ was considered to be static and the residue was computed from many rigidly transformed copies of set $\mathcal{B}$. Thus, data structures were built only for set $\mathcal{A}$ and the query points were selected to be all the points in each of the rigid transformations of $\mathcal{B}$. This section contains data of over 100,000 residue computations (each of which is made up of tens of thousands of range searching or nearest neighbor operations) executed over 9 different 3D models, consisting in 2 misaligned views per model. Al runtime results are presented in seconds.

A possible naive strategy to solve the problem would be Brute-Force, i.e. in order to find the nearest neighbor of set $\mathcal{A}$ all the points in $\mathcal{A}$ are traversed and the minimum distance is kept. However, its huge computational cost ($O(nm)$ for residue computation) makes it unacceptable in practice. For example: with the Stanford Bunny shown in Figure 4.1 with 40,000 points, Brute-Force computations take about 88 seconds. Data structures with the same asymptotic cost like the Regular Grid or the Compressed Octree complete the same task in 0.09 and 0.22 seconds, respectively.

4.2.1 Synthetic experiments

Synthetic experiments provide a scenario were most of the parameters that are unknown in practice can be defined to our convenience. In this case, we aimed at testing the performance of data structures when working with point sets with irregular density, i.e. large sparse parts as well as very dense areas. We run series of experiments changing parameters such as the model size (in terms of coordinate axes), the number of points and the point density distribution.

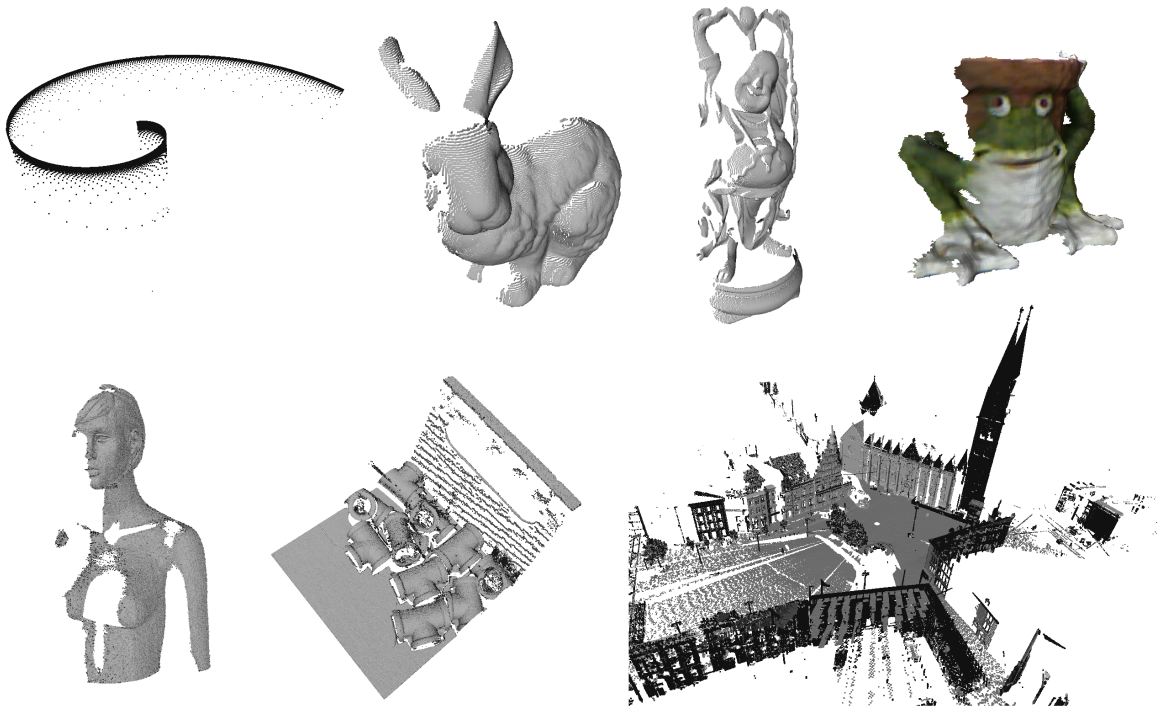

Figure 4.1: Synthetic Spiral (top-left), Bunny, Buddha, Frog, Bust (bottom-left), Joints and Bremen models.

Specifically, we built 3 different types of Spiral models. An example of such a surface (with logarithmic descent along the z axis) is depicted in Figure 4.1 and has the following parametric equation:

$$\begin{array}{ccc} (0,2\pi)\times\mathbb{R}^{+} & \longrightarrow & \mathbb{R}^{3} \\ (t,x) & & (t\sin(t),t\cos(t),\dfrac{K}{\log(x)}) \end{array}$$

All the sets used in this section were generated with our own automatic Point Cloud Generator available at [8]. Point clouds of 70,000 points were sampled over every spiral surface. For each data structure and Spiral version, we performed 100 executions with the spiral model as set $\mathcal{A}$ and copies of itself rotated around the z axis (in equally spaced intervals of increasing angle) as set $\mathcal{B}$. This results in a residue computation where no noise or missing data is present, but each point has many potential nearest neighbors. This scenario is expected to be specially detrimental for data structures that compute the residue using range searching. Figure 4.2 and Table 4.1 show the mean time for residue computation tests.

The S4PCS implementation of the KDtree performs best of all, especially compared with the Regular Grid, where its performance is, as expected, worse than that of any KDtree. This can be explained since the areas of the spirals with large concentrations of points result in grid

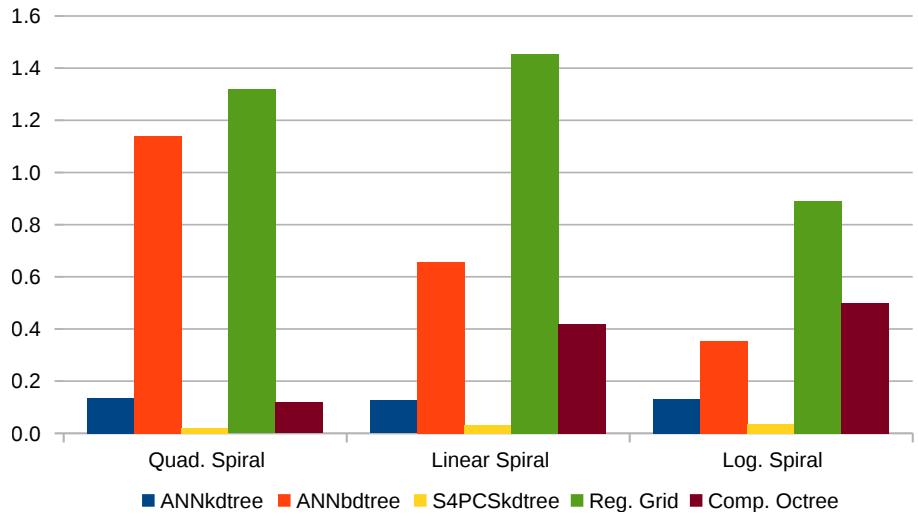

Figure 4.2: Runtimes (in seconds) for the three spiral models. The first follows a quadratic descent along the z axis, the second a linear descent and the third a logarithmic descent.

cells with many points. Besides, the BDtree underperforms somehow unexpectedly. Possibly the bounded aspect ratio property associated with its nodes fails at the same areas of high point concentration. It is important to note how, even though in this case the data structure with best performance is also one of those with best asymptotic costs, the BDtree (which has the same asymptotic cost) is the worst in terms of runtime performance. This exemplifies the differences between asymptotic cost and runtime performance.

4.2.2 Real data experiments

Although synthetic experiments have shown challenging situations for all the studied data structures, it is not clear whether these situations appear in practice. In order to test the relative importance of these worst-case scenarios, we present results obtained when working with real data from the 3D point set matching problem. We used different kinds of models [97] shown in Figure 4.1. The widely used **Bunny** and **Buddha** models from the Stanford repository [3] were obtained with a range scan, with ≈ 40,000 and ≈ 80,000 points per view, respectively. These two models are the two easiest to align as they present high ratio of overlap between views and lack of noise. The **Frog** model from [16] with ≈ 26,000 points has less overlapping than Stanford models. The mannequin **Bust** [95] with ≈ 35,0000 points per view has low overlapping and acquisition noise since contains raw data acquired by an structured light scanner. The bin-picking model of metal **Joints** was obtained with a range scanner [97] having ≈ 520,000 points for set $\mathcal{A}$. This set represents a large heap of unsorted metal pieces. However, set $\mathcal{B}$ (which stands for a single metal piece to be found in the heap) has only ≈ 25,000 points. These issues make this a specially challenging problem of interest in industrial applications. Finally, in order to encompass

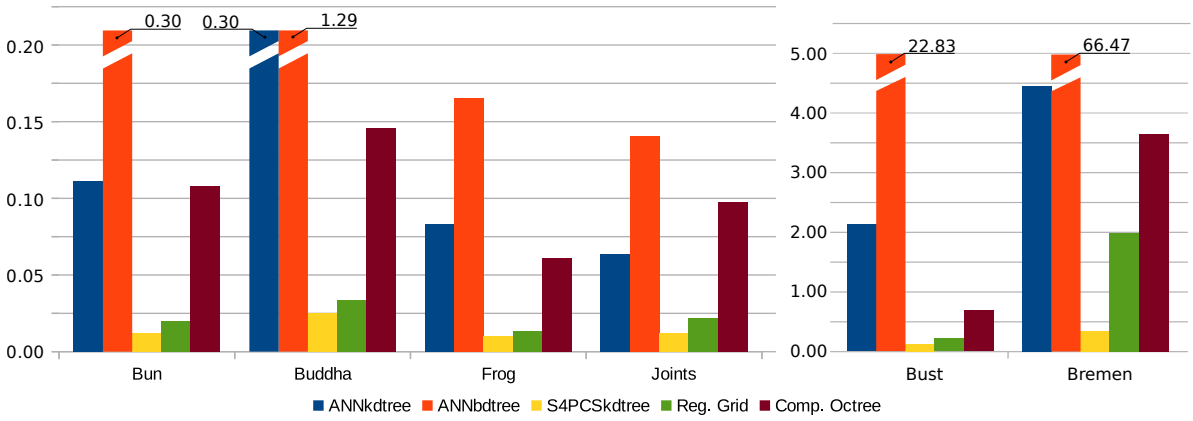

Figure 4.3: Mean running time of residue computation for all real point clouds and methods.

a widerange of situations, we used the **Bremen** model, acquired with a Lidar system. This model is the biggest one, with ≈ 2,000,000 points, with high rates of occlusion and noise.

For all models, given two misaligned views of them, we generated many transformation matrices which try to align both views. We extracted these matrices from the widely used 4PCS registration method [17]. Then, each transformation is applied and the residue and overlapping are computed using all data structures.

Figure 4.3 and Table 4.1 show the average execution time of residue computation for each model. We observed that the best performances are obtained for S4PCSkdtree and Regular Grid. BDtree is clearly the worst data structure with all models.

Notice that data structures with the same asymptotic cost perform differently. For example, ANNkdtree and ANNbdtree obtain very different results. Once again, the S4PCSkdtree implementation of KDtrees obtains the fastest results. This data structure unites the best asymptotic cost with an implementation that is fine tuned to the problem and thus performs best overall.

Nevertheless, the Regular Grid, which is among the worse candidates in terms of asymptotic cost, obtains very good performances in all scenarios. In this case, the direct access to smaller parts of the model accelerates the neighbor searching, obtaining the second best runtime in all tests. Although a sequential search is carried in Regular Grid cells, the points are distributed fairly evenly among the cells. Concerning the models, it is important to notice that their runtimes vary in terms of the number of points of the model but also due to the presence of noise and occlusions. To cite a representative example, the runtime of a single nearest neighbor search for the regular grid are 4.58e-7 s. and 1.0e-6 s. for the noisy Bust and Bremen models, respectively, while it goes down to 5.6e-7 s. for the Bunny model. This shows the importance of considering models with different characteristics when analyzing the behavior of algorithms within the context of point cloud matching.

Finally, in order to tell the full story about the data structures performace, we observed a better performance of KDtree when the sets were better aligned. Figure 4.4 depicts curves for the

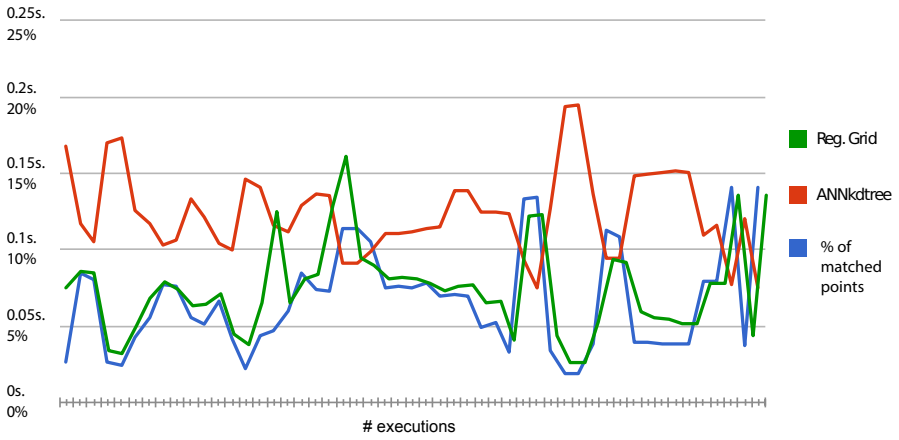

Figure 4.4: Relation between % of matched points and computation time of ANNkdtree and Regular Grid. Notice that, while Regular Grid curve mathces the % of matched points curve, the ANNkdtree curve does the exact opposite.

time and matching ratio of ANNkdtree and Regular Grid, associated with the 263 executions corresponding to the Bunny model. The figure shows how the ANNkdtree runtimes present an inverse behaviour while Regular Grid follow the curve of the matching results. So, it seems that KDtree is able to compute residues faster for sets with large correspondence ratio. This can be useful in the final stages of the matching process and for fine matching applications [100].

4.3 GridDS, a hybrid data structure

Section 4.2 showed that Regular Grid obtained very good results when used with real data. At the same time, this structure was shown to under-perform in sets with severe variations in density. Consequently, we decided to combine the Regular Grid with other "inner" data structures in those areas where its performance deteriorates. The result is a new hybrid data structure that takes advantage of the characteristics of the Regular Grid and of whatever inner data structure was used. The possibility of hybrid data structures was suggested in [53] and for this proposal we have developed and implemented the idea as well as studied its runtime performance. We have implemented and studied Regular Grids combined with all the data structures considered throughout this section.

The GridDS is made up of a Regular Grid where cells that get a large number of points (larger than a certain threshold) are associated to another inner data structure. Such inner structure could be a simple list of unsorted points or any of the data structures reviewed in Section 4.1, among some others present in the literature (see Figure 4.5). We believe that the direct access provided by the Grid works very well for parts of the set where the points are sparsely distributed, while the inner data structure provides much better access at parts of the set where points are more tightly packed. Furthermore, the grid structure avoids exploring unnecessary parts

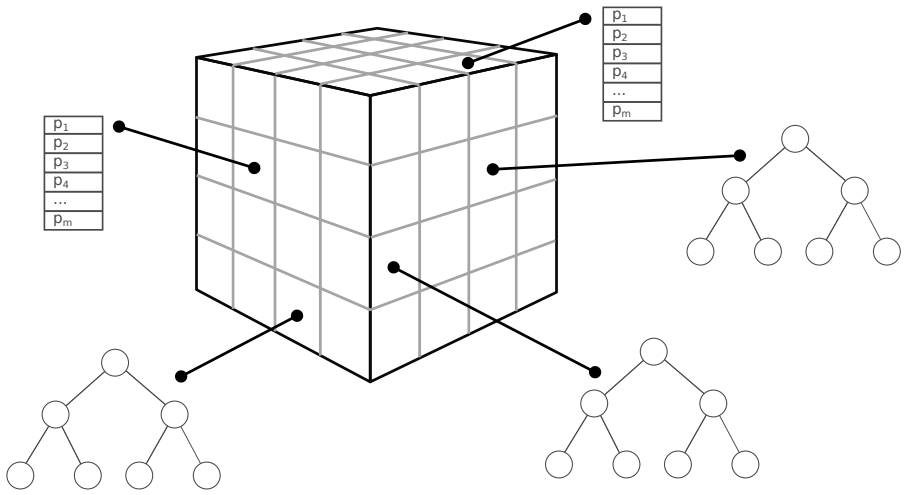

Figure 4.5: GridDS representation.

of the model, speeding up the nearest neigbour searching. As we show in the Section 4.3.3, our proposal outperforms all the data structures reviewed in Section 4.1. Source code for the GridDS is available at [6]

4.3.1 Building cost

In order to build the data structure we first need to build a Regular Grid able to store any other inner data structure. From now on, we will name this inner data structure as *InnerDS*. Additionally, we need to keep track of the cells that have more points than the InnerDS construction threshold. This can easily be done while building the regular grid so the cost of this step remains $O(n)$. Then, we need to build the InnerDS for the cells that need them. If we note r to be the number of such cells, the cost of this is $\sum_1^r n_i \log(n_i)$ where n_i is the number of points in the i-th cell where we need to build, for example, a KDtree. As each point in the original set belongs to at most one cell, we have that $n_i \leq n \ \forall i$, and as the logarithm is a monotonically increasing function we also have $\log(n_i) \leq \log(n) \ \forall i$ (consequently $\sum_1^r n_i \log(n_i) \leq \log(n)(\sum_1^r n_i)$ using again that each point belongs to at most one of these cells where we need to build an InnerDS. Hence, we obtain that the cost of building the structure is $O(n \log n)$.

4.3.2 Automatic thresholding

The distribution and density of points and error thresholdings play an important role in the searching process. In any data structure, there are some parameters to be adjusted in order to adapt the data structure to each model and reach the best performance. In our case, there are two main parameters to analyze: point threshold to decide to use an InnerDS (*pointThrs*) and the

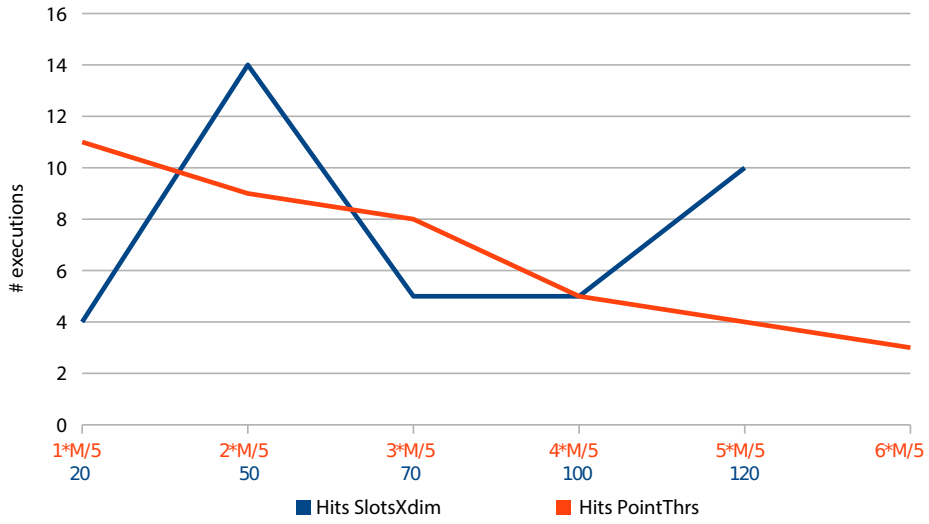

Figure 4.6: Results of our thresholding tests. We tested different value combination of *pointThrs* (from $1 * M/5$ to $6 * M/5$) and *slotsXdim* (from 20 to 120).

number of cells of the 3D Regular Grid (*#cells*). As we use a Regular Grid, we defined the slots per dimension (*slotsXdim*) as our base parameter, where $\#cells = slotsXdim^3$. The goal is, as explained in Section 4.1.3, to have roughly as many cells as points. Thus, we developed a set of experiments in order to find the best parameter combination.

Figure 4.6 shows a graphical summary of the thresholding tests, which consists in using our experimental setup explained in Section 4.2, tuning the *pointThrs* and *slotsXdim* parameters. For each model, we performed all residue computation tests with different parameter combinations. *slotsXdim* takes values from 20 to 120 and *pointThrs* varyies from $1 * M/5$ to $6 * M/5$ (30 average residue computation tests per model). We define M as the mean number of points of not-empty cells, which give us a general occupation measure. Thus, setting *PointThrs* with multiple values of $M/5$ we can control, approximately, the percentage of cells without an InnerDS. For example, if $M = 50\%$ of cell occupancy, setting $PointThrs = 4 * M/5$ means that 40% of cells do not contain an InnerDS.

For each model, we selected the 8 best average runtimes from these 30 tests. Figure 4.6 counts, for both *slotsXdim* and *pointThrs*, the number of times that each parameter value appears in these 8 best results.

Obtained results indicate that *slotsXdim* parameter is dependent of model size since the best runtimes increase as increases the model size. Thus, we decided to link this parameter to the number of points n, having $slotsXdim = \sqrt{\lceil 3 \rceil n}$. Regarding *pointThrs*, and according to the results, we set up $PointThrs = 2 * M/5$. With this strategy we achieve a setup that is able to adapt to the varying needs of each model producing very good practical results automatically and without the need for any user intervention.

4.3.2.1 Residue computation cost

First, the algorithm needs to find the cells affected by the range search operation of radius ε. For all affected cells, if they contain an InnerDS, the algorithm runs a nearest neighbor search. If not, the algorithm checks all the points in the cell to find the nearest neighbor as in a Regular Grid. In this case, we can assume that cells not containing an InnerDS have at most a constant number of points (by definition). Consequently, if we assume that at most a constant number of cells are touched by the ε radius, then the cost of finding the nearest neighbor of a given point a distance smaller than ε is the same cost than that of finding a nearest neighbor in an InnerDS. Additionally, its residue computation runtime is as good as that of the InnerDS. Consequently, by incurring only in a $\log n$ penalty in construction time we obtain a data structure that is as fast as the InnerDS in residue computation time. Additionally, Section 4.3.3 will show how, the direct access to the parts of the sets where the points are regularly distributed, while not appearing in this asymptotic cost discussion, results in a noticeable improvement on its runtime performance.

4.3.3 GridDS results

In Figure 4.7, and in deeper detail in Table 4.1, we show the runtimes for each model and data structures when is used as InnerDS. First, focusing on synthetic data, the performance of GridDS suffers from the Regular Grid behavior reported in Section 4.2.1. However note the improvement compared to single ANN data structures and single Compressed Octree, although similar results are achieved compared to S4PCSkdtree.

In real scenarios GridDS obtains the best results, outperforming all data structures. For ANN data structures we achieve between 72% (ANNkdtree) and 99% (ANNbdtree) of runtime improvement. With Compressed Octree, the time reduction is similar, up to 81% in the best case (Bremen model). Even with S4PCSkdtree, GridDS achieve from 2% to 20% of reduction, depending on the model.

This illustrates the effectiveness of GridDS strategy, dividing the space in order to quickly focus the nearest neighbor search and accelerate residue computation.

4.4 Conclusions

We have presented a study analyzing the asymptotic costs and runtimes of several data structures used for the computation of residues for point cloud matching. Results showed how, in some cases, data structures that do not have good asymptotic costs can outperform others that do (see Figure 4.3). Additionally, structures with the same asymptotic cost sometimes obtain significantly different results. This shows the need for using both types of analysis when evaluating data structures. Furthermore, we observed small variations in the behavior of data structures as well as code optimization are important factors for runtime performance. This is best illustrated by the differences observed (see figure 4.3) between the general-purpose ANNkdtree, and the much

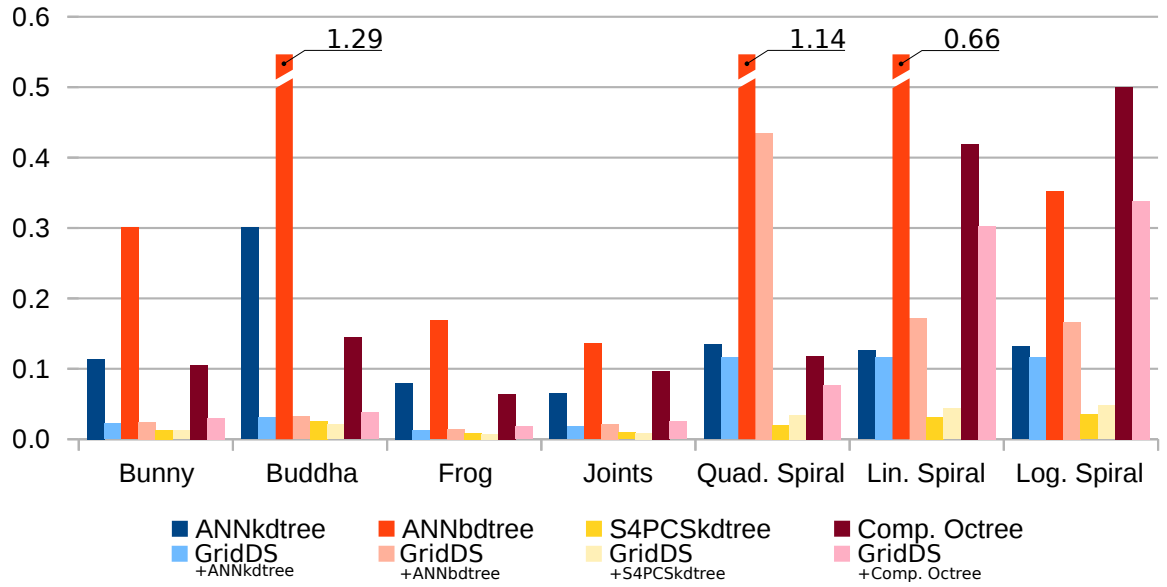

Figure 4.7: Residue computation time comparison. Each bar in dark color depicts the runtime of each data structure. The light bars show the runtimes using a GridDS with each data structure as InnerDS.

more code-optimized and problem-tailored S4PCSkdtree. The comparatively good performance of Regular Grids, which are second overall for real life data sets and take only 36% longer on average than the best data structure studied (the S4PCSkdtree) drove us to develop a new hybrid data structure aiming at taking advantage of this good performance of Regular Grids while avoiding their potential pitfalls as shown in Figure 4.7.

The subsequent data structure (GridDS) uses a Regular Grid for the parts of the model where direct access to points is an advantage and reverts to an inner data structure for the rest. Results show how this hybrid approach manages to improve the computation times of whatever data structure the regular grid is being combined with. Specifically, GridDS improves on average the Compressed Octree in a 74%, the ANNbdtree in a 93%, the ANNkdtree in a 84% and even manages to improve the runtime performance of the S4PCSkdtree by 12%. To the best of our knowledge, this improvement in the run-time performance of what is currently the fastest data structure for residue computation in point cloud matching problems yields the best results to date. Additionally, the fact the that GridDS can be combined with different inner data structures has the potential to improve the runtimes of existing and future applications without compromising the properties that the structures used in them might have.

Table 4.1: Results of all experiments. For each model, both synthetic and real data, we show the number of residue computation executions (*#exec*), the execution time of each single data structure and the execution time of the same data structure as an InnerDS inside the GridDS. The results in green show the fastest runtimes.

	#exec	**ANNkdtree**	**GridDS**	**ANNbdtree**	**GridDS**	**S4PCSkdtree**	**GridDS**	**Comp. Octree**	**GridDS**
Bunny	393	0.114	0.022	0.301	0.025	0.013	0.012	0.106	0.030
Buddha	528	0.301	0.031	1.298	0.032	0.026	0.021	0.144	0.038
Frog	263	0.080	0.013	0.168	0.014	0.009	0.008	0.064	0.018
Bust	8966	2.105	0.140	22.838	0.147	0.143	0.112	0.672	0.182
Joints	514	0.065	0.018	0.136	0.022	0.010	0.009	0.097	0.025
Bremen	12365	4.468	0.640	66.475	0.792	0.377	0.369	3.636	0.691
Quad. Spiral	100	0.135	0.116	1.139	0.435	0.020	0.034	0.118	0.077
Linear Spiral	100	0.126	0.117	0.657	0.172	0.031	0.043	0.419	0.303
Log. Spiral	100	0.132	0.117	0.353	0.166	0.035	0.048	0.500	0.339

CHAPTER 5

CONCLUSIONS

We conclude this thesis with the satisfaction of having fulfilled all proposed objectives. We thoroughly reviewed the 3D registration problem (Chapter 2), especially focusing on coarse matching. We proposed a Registration Pipeline in order to organize and classify the main existing methods. We tested them in different scenarios (Chapter 3) in order to prove its performance not only with synthetic data, but with real data, which is much more difficult. Furthermore, we proposed a standardized notation for the concepts used regarding point cloud matching problem in all the disciplines that tackle it. Then, we developed a 3D Registration Toolbox (Chapter 3) consisting both of registration software and a model database, which is freely available online. Our software is implemented in C++ and is prepared for executing all the steps of the Registration Pipeline independently. We developed this software following the S.O.L.I.D. principles, which allow the Developers to expand the software with their own algorithms easily, and compare them to state-of-the-art methods. Our model database complements the software providing not only the input data (point cloud models) but also useful data (ground truth alignment, residue values, descriptor values...) for algorithm comparison. Moreover we developed a hybrid data structure for residue computation (Chapter 4) and we demonstrate that our proposal overcomes the state-of-the-art data structures in terms of speed. We also presented a new hybrid hardware/software searching strategy that performs very well in all scenarios, both synthetic and real, besides two new Detection methods.

This thesis has been a team effort and I have been able to work with other members of my research group as well as with coauthors from other institutions and colleagues from other research centers that I had the good fortune to visit. However, in this section I want to give a personal view on what I see when I look back on the work done in these past few years. During the development of this thesis, I acquired not only a deep knowledge in the field, but I understand

the evolution of this topic. I want to note some facts that I learned during this thesis: In the course of my research I found out that ICP is generally used as a refinement method and it can be considered as an standard. I noticed that there are many publications every year on Descriptors but they still having problems to deal with real data, with high noise rates. Searching Strategies are not so popular in the literature but I learned that, in order to deal with difficult or huge datasets, the best option is having a good Searching Strategy, because Descriptors are not powerful enough and also very time consuming. I also found out that the new trend in registration topic is using information from external sensors to help the Coarse Matching part, like in new virtual and augmented reality products or smartphone applications. Finally, I conclude that, although many advances are presented every year, in order to have an accurate and detailed registration result, huge computation costs are still needed. However, GPU's approaches are opening an encourage path to solve these problems.

5.1 Contributions

The main contributions of the present thesis can be summarized as follows:

- Proposing a standardized notation for 3D registration problems in order to unify definitions coming from different research communities. This work is published in [49].
- Proposing a Registration Pipeline so that existing methods can be classified according to it. This work is published in [49].
- Developing a new 3D Registration Toolbox, including two new Detection methods (Color Space Sampling and Connected Components Sampling) and one new Searching Strategy (Hierarchical Grid3D). This work is under preparation to be submitted in a journal.
- Providing experimental evaluation about some of the state-of-the-art algorithms for each of the steps of the Pipeline. This work is published in [97].
- Developing a new data structure for residue computation as well as a new hybrid hardware/software search strategy. This work is submitted for publication in a journal.

5.2 Future work

With the experience acquired these years I clearly identify two tendencies in the research field of 3D registration: On one hand, I believe that the future of the registration topic goes by using GPU capabilities to solve the alignment process. Although a lot of methods are presented, the new acquisition devices provides more resolution and precision, and a faster computation becomes mandatory. On the other hand, 3D registration is becoming popular in the commercial world do to the application designed for smartphones. The the lack of computation power of these devices

is supplied using external information from embedded hardware, which is an interesting trend. I think that in next years we can see this technology as a fundamental part of the new virtual reality entertainment systems.

Following the research developed in this thesis we want to mention a list of topics that could be developed to continue our work. We propose the following improvements for our Registration Toolbox:

- Provide a graphical interface to enhance the user experience, with a 3D viewer and an easy-to-use parametrization system.
- Provide support for different input formats, like *.obj or *.3ds files to name a few.
- Add more methods in each step of the Pipeline and also more objects in the Database.
- Add GPU support in order to work with these types of algorithms.
- Provide a complete User/Developer manual and an API documentation.
- Develop a GPU version of our GridDS data structure in order explore each cell simultaneously, reducing the searching time.

BIBLIOGRAPHY

[1] *http://geometry.cs.ucl.ac.uk/projects/2014/super4pcs/*.

[2] *http://gfx.cs.princeton.edu/proj/trimesh2/*.

[3] *http://graphics.stanford.edu/data/3dscanrep*.

[4] *http://lgg.epfl.ch/~ichim/registration_tutorial_ram_2015/*.

[5] *https://en.wikipedia.org/wiki/solid_(object-oriented_design)*.

[6] *https://github.com/ferranroure/gridds*.

[7] *https://github.com/ferranroure/registrationpipeline*.

[8] *https://gitlab.com/froure/pointcloudgenerator*.

[9] *http://structure.io/*.

[10] *https://www.cs.umd.edu/~mount/ann/*.

[11] *https://www.cs.umd.edu/~mount/ann/files/1.1.2/annmanual_1.1.pdf*.

[12] *https://www.google.com/atap/projecttango/#project*.

[13] *https://www.microsoft.com/microsoft-hololens/*.

[14] *https://www.oculus.com/*.

[15] *http://www.pointclouds.org*.

[16] *http://www.vision.deis.unibo.it/research/80-shot*.

[17] D. AIGER, N. J. MITRA, AND D. COHEN-OR, *4-points congruent sets for robust pairwise surface registration*, in ACM Transactions on Graphics, vol. 27, 2008, p. 85.

[18] A. ALBARELLI, E. RODOLA, AND A. TORSELLO, *A game-theoretic approach to fine surface registration without initial motion estimation*, in Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, IEEE, 2010, pp. 430–437.

[19] M. ALEXA, *Recent advances in mesh morphing*, in Computer graphics forum, vol. 21, 2002, pp. 173–198.

[20] S. ALURU AND F. E. SEVILGEN, *Dynamic compressed hyperoctrees with application to the n-body problem*, in Foundations of Software Technology and Theoretical Computer Science, Springer, 1999, pp. 21–33.

[21] S. ARYA AND D. M. MOUNT, *Approximate nearest neighbor queries in fixed dimensions.*, in SODA, vol. 93, 1993, pp. 271–280.

[22] S. ARYA, D. M. MOUNT, N. S. NETANYAHU, R. SILVERMAN, AND A. Y. WU, *An optimal algorithm for approximate nearest neighbor searching fixed dimensions*, Journal of the ACM (JACM), 45 (1998), pp. 891–923.

[23] S. BAILEY, *Principal component analysis with noisy and/or missing data*, Publications of the Astronomical Society of the Pacific, 124 (2012), pp. 1015–1023.

[24] P. J. BESL AND N. D. MCKAY, *A method for registration of 3-d shapes*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 14 (1992), pp. 239–256.

[25] H. BLUM ET AL., *A transformation for extracting new descriptors of shape*, Models for the perception of speech and visual form, 19 (1967), pp. 362–380.

[26] F. BOGO, J. ROMERO, M. LOPER, AND M. J. BLACK, *FAUST: Dataset and evaluation for 3D mesh registration*, in Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Piscataway, NJ, USA, June 2014, IEEE.

[27] F. BOUGHORBEL, Y. ZHANG, S. KANG, U. CHIDAMBARAM, B. ABIDI, A. KOSCHAN, AND M. ABIDI, *Laser ranging and video imaging for bin picking*, Assembly Automation, 23 (2003), pp. 53–59.

[28] E. BOYER, A. BRONSTEIN, M. BRONSTEIN, B. BUSTOS, T. DAROM, R. HORAUD, I. HOTZ, Y. KELLER, J. KEUSTERMANS, A. KOVNATSKY, ET AL., *Shrec 2011: robust feature detection and description benchmark*, arXiv preprint arXiv:1102.4258, (2011).

[29] A. E. A. BRONSTEIN, *Shrec 2010: robust feature detection and description benchmark*, Eurographics Workshop on 3D Object Retrieval, 2 (2010), p. 6.

[30] A. M. BRONSTEIN, M. M. BRONSTEIN, AND R. KIMMEL, *Numerical geometry of non-rigid shapes*, Springer, 2008.

[31] B. BUSTOS, D. A. KEIM, D. SAUPE, T. SCHRECK, AND D. V. VRANIĆ, *Feature-based similarity search in 3d object databases*, ACM Computing Surveys (CSUR), 37 (2005), pp. 345–387.

[32] O. CARMICHAEL, D. HUBER, AND M. HEBERT, *Large data sets and confusing scenes in 3-d surface matching and recognition*, in IEEE International Conference on 3D Digital Imaging and Modeling, 1999, pp. 358–367.

[33] F. CAZALS AND M. POUGET, *Estimating differential quantities using polynomial fitting of osculating jets*, Computer Aided Geometric Design, 22 (2005), pp. 121–146.

[34] C.-S. CHEN, Y.-P. HUNG, AND J.-B. CHENG, *Ransac-based darces: A new approach to fast automatic registration of partially overlapping range images*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 21 (1999), pp. 1229–1234.

[35] H. CHEN AND B. BHANU, *3d free-form object recognition in range images using local surface patches*, Pattern Recognition Letters, 28 (2007), pp. 1252–1262.

[36] C. K. CHOW, H. T. TSUI, AND T. LEE, *Surface registration using a dynamic genetic algorithm*, Pattern Recognition, 37 (2004), pp. 105–117.

[37] C. S. CHUA AND R. JARVIS, *Point signatures: A new representation for 3d object recognition*, International Journal of Computer Vision, 25 (1997), pp. 63–85.

[38] D. H. CHUNG, I. D. YUN, AND S. U. LEE, *Registration of multiple-range views using the reverse-calibration technique*, Pattern Recognition, 31 (1998), pp. 457–464.

[39] D. COHEN-STEINER AND J.-M. MORVAN, *Restricted delaunay triangulations and normal cycle*, in ACM Annual Symposium on Computational Geometry, 2003, pp. 312–321.

[40] N. D. CORNEA, M. F. DEMIRCI, D. SILVER, A. SHOKOUFANDEH, S. J. DICKINSON, AND P. B. KANTOR, *3d object retrieval using many-to-many matching of curve skeletons*, in International Conference on Shape Modeling and Applications, IEEE, 2005, pp. 366–371.

[41] N. D. CORNEA, D. SILVER, AND P. MIN, *Curve-skeleton properties, applications, and algorithms*, IEEE Transactions on Visualization and Computer Graphics, 13 (2007), pp. 530–548.

[42] A. CZUMAJ, C. SOHLER, AND M. ZIEGLER, *Property testing in computational geometry*, in Algorithms-ESA 2000, Springer, 2000, pp. 155–166.

[43] T. DAROM AND Y. KELLER, *Scale-invariant features for 3-d mesh models*, IEEE Transactions on Image Processing, 21 (2012), pp. 2758–2769.

[44] J. DE REU, G. PLETS, G. VERHOEVEN, P. DE SMEDT, M. BATS, B. CHERRETTÉ, W. DE MAEYER, J. DECONYNCK, D. HERREMANS, P. LALOO, ET AL., *Towards a three-dimensional cost-effective registration of the archaeological heritage*, Journal of Archaeological Science, 40 (2013), pp. 1108–1121.

[45] T. K. DEY, J. GIESEN, AND S. GOSWAMI, *Shape segmentation and matching with flow discretization*, in Algorithms and Data Structures, Springer, 2003, pp. 25–36.

[46] T. K. DEY, K. LI, C. LUO, P. RANJAN, I. SAFA, AND Y. WANG, *Persistent heat signature for pose-oblivious matching of incomplete models*, in Computer Graphics Forum, vol. 29, 2010, pp. 1545–1554.

[47] Y. DÍEZ, J. MARTÍ, AND J. SALVI, *Hierarchical normal space sampling to speed up point cloud coarse matching*, Pattern Recognition Letters, 33 (2012), pp. 2127 – 2133.

[48] Y. DÍEZ, A. OLIVER, X. LLADÓ, J. FREIXENET, J. MARTÍ, J. C. VILANOVA, AND R. MARTI, *Revisiting intensity-based image registration applied to mammography*, Information Technology in Biomedicine, IEEE Transactions on, 15 (2011), pp. 716–725.

[49] Y. DÍEZ, F. ROURE, X. LLADÓ, AND J. SALVI, *A qualitative review on 3d coarse registration methods*, ACM Computing Surveys (CSUR), 47 (2015), p. 45.

[50] M. P. DO CARMO, *Differential geometry of curves and surfaces*, vol. 2, 1976.

[51] M. DONOSER AND H. BISCHOF, *3d segmentation by maximally stable volumes (msvs)*, in Pattern Recognition, 2006. ICPR 2006. 18th International Conference on, vol. 1, 2006, pp. 63–66.

[52] H. DUTAGACI, C. P. CHEUNG, AND A. GODIL, *Evaluation of 3d interest point detection techniques via human-generated ground truth*, The Visual Computer, 28 (2012), pp. 901–917.

[53] C. ERICSON, *Real-time collision detection*, CRC Press, 2004.

[54] J. FELDMAR AND N. AYACHE, *Rigid, affine and locally affine registration of free-form surfaces*, International Journal of Computer Vision, 18 (1996), pp. 99–119.

[55] J. H. FRIEDMAN, F. BASKETT, AND L. J. SHUSTEK, *An algorithm for finding nearest neighbors*, IEEE Transactions on computers, (1975), pp. 1000–1006.

[56] J. H. FRIEDMAN, J. L. BENTLEY, AND R. A. FINKEL, *An algorithm for finding best matches in logarithmic expected time*, ACM Transactions on Mathematical Software (TOMS), 3 (1977), pp. 209–226.

[57] K. FUJIMURA, H. TORIYA, K. YAMAGUCHI, AND T. KUNII, *Oct-tree algorithms for solid modeling*, Springer, 1983.

[58] N. GELFAND, N. J. MITRA, L. J. GUIBAS, AND H. POTTMANN, *Robust global registration*, in Eurographics Symposium on Geometry Processing, 2005, pp. 197–206.

[59] A. GRUEN AND D. AKCA, *Least squares 3d surface and curve matching*, ISPRS Journal of Photogrammetry and Remote Sensing, 59 (2005), pp. 151–174.

[60] Y. GUO, F. A. SOHEL, M. BENNAMOUN, J. WAN, AND M. LU, *Rops: A local feature descriptor for 3d rigid objects based on rotational projection statistics*, in Communications, Signal Processing, and their Applications (ICCSPA), 2013 1st International Conference on, 2013, pp. 1–6.

[61] D. HOLZ, A. E. ICHIM, F. TOMBARI, R. B. RUSU, AND S. BEHNKE, *Registration with the point cloud library: A modular framework for aligning in 3-d*, IEEE Robotics & Automation Magazine, 22 (2015), pp. 110–124.

[62] H. HOPPE, T. DEROSE, T. DUCHAMP, J. MCDONALD, AND W. STUETZLE, *Surface reconstruction from unorganized points*, vol. 26, ACM, 1992.

[63] Q.-X. HUANG, B. ADAMS, M. WICKE, AND L. J. GUIBAS, *Non-rigid registration under isometric deformations*, in Computer Graphics Forum, vol. 27, 2008, pp. 1449–1457.

[64] N. IYER, S. JAYANTI, K. LOU, Y. KALYANARAMAN, AND K. RAMANI, *Three-dimensional shape searching: state-of-the-art review and future trends*, Computer-Aided Design, 37 (2005), pp. 509–530.

[65] A. E. JOHNSON, *Spin-images: A representation for 3-D surface matching*, PhD thesis, Citeseer, 1997.

[66] A. E. JOHNSON AND M. HEBERT, *Using spin images for efficient object recognition in cluttered 3d scenes*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 21 (1999), pp. 433–449.

[67] K. KHOSHELHAM AND S. O. ELBERINK, *Accuracy and resolution of kinect depth data for indoor mapping applications*, Sensors, 12 (2012), pp. 1437–1454.

[68] H. KIM AND A. HILTON, *Evaluation of 3d feature descriptors for multi-modal data registration*, in International Conference on 3DTV, IEEE, 2013, pp. 119–126.

[69] M. KÖRTGEN, G.-J. PARK, M. NOVOTNI, AND R. KLEIN, *3d shape matching with 3d shape contexts*, in Central European Seminar on Comp. Graph., vol. 3, 2003, p. 5.

[70] S. KUMAR, M. SALLAM, AND D. GOLDGOF, *Matching point features under small nonrigid motion*, Pattern Recognition, 34 (2001), pp. 2353–2365.

[71] R. L. LARKINS, M. J. CREE, AND A. A. DORRINGTON, *Verification of multi-view point-cloud registration for spherical harmonic cross-correlation*, in Proceedings of the 27th Conference on Image and Vision Computing New Zealand, ACM, 2012, pp. 358–363.

[72] X. LI AND I. GUSKOV, *Multi-scale features for approximate alignment of point-based surfaces*, in Eurographics Symposium on Geometry Processing, vol. 217, 2005.

[73] Y. LI, A. DAI, L. GUIBAS, AND M. NIESSNER, *Database-assisted object retrieval for real-time 3d reconstruction*, in Computer Graphics Forum, vol. 34, Wiley Online Library, 2015, pp. 435–446.

[74] Z. LIAN, A. GODIL, B. BUSTOS, M. DAOUDI, J. HERMANS, S. KAWAMURA, Y. KURITA, G. LAVOUÉ, H. VAN NGUYEN, R. OHBUCHI, ET AL., *A comparison of methods for non-rigid 3d shape retrieval*, Pattern Recognition, (2012).

[75] Y.-S. LIU AND K. RAMANI, *Robust principal axes determination for point-based shapes using least median of squares*, Computer-Aided Design, 41 (2009), pp. 293–305.

[76] W. L. D. LUI, T. J. J. TANG, T. DRUMMOND, AND W. H. LI, *Robust egomotion estimation using icp in inverse depth coordinates*, in IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2012, pp. 1671–1678.

[77] A. MAKADIA, A. PATTERSON, AND K. DANIILIDIS, *Fully automatic registration of 3d point clouds*, in Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on, vol. 1, 2006, pp. 1297–1304.

[78] S. MANAY, B.-W. HONG, A. YEZZI, AND S. SOATTO, *Integral invariant signatures*, European Conf. on Computer Vision, (2004), pp. 87–99.

[79] C. MARTINEZ, R. BOCA, B. ZHANG, H. CHEN, AND S. NIDAMARTHI, *Automated bin picking system for randomly located industrial parts*, in 2015 IEEE International Conference on Technologies for Practical Robot Applications (TePRA), IEEE, 2015, pp. 1–6.

[80] T. MASUDA, *Generation of geometric model by registration and integration of multiple range images*, in IEEE International Conference on 3D Digital Imaging and Modeling, 2001, pp. 254–261.

[81] T. MASUDA, *Registration and integration of multiple range images by matching signed distance fields for object shape modeling*, Computer Vision and Image Understanding, 87 (2002), pp. 51–65.

[82] T. MASUDA, K. SAKAUE, AND N. YOKOYA, *Registration and integration of multiple range images for 3-d model construction*, in IEEE International Conference on Pattern Recognition, vol. 1, 1996, pp. 879–883.

[83] J. MATAS, O. CHUM, M. URBAN, AND T. PAJDLA, *Robust wide-baseline stereo from maximally stable extremal regions*, Image and vision computing, 22 (2004), pp. 761–767.

[84] N. MELLADO, D. AIGER, AND N. J. MITRA, *Super 4pcs fast global pointcloud registration via smart indexing*, in Computer Graphics Forum, vol. 33, Wiley Online Library, 2014, pp. 205–215.

[85] A. MIAN, M. BENNAMOUN, AND R. OWENS, *On the repeatability and quality of keypoints for local feature-based 3d object retrieval from cluttered scenes*, International Journal of Computer Vision, 89 (2010), pp. 348–361.

[86] N. J. MITRA AND A. NGUYEN, *Estimating surface normals in noisy point cloud data*, in Proceedings of the nineteenth annual symposium on Computational geometry, ACM, 2003, pp. 322–328.

[87] L. NAJMAN, M. COUPRIE, ET AL., *Quasilinear algorithm for the component tree*, in Proceedings of SPIE, vol. 5300, 2004, pp. 98–107.

[88] A. NUCHTER, K. LINGEMANN, AND J. HERTZBERG, *Cached kd tree search for icp algorithms*, in 3-D Digital Imaging and Modeling, 2007. 3DIM'07. Sixth International Conference on, 2007, pp. 419–426.

[89] M. OVSJANIKOV, Q. MÉRIGOT, F. MÉMOLI, AND L. GUIBAS, *One point isometric matching with the heat kernel*, in Computer Graphics Forum, vol. 29, 2010, pp. 1555–1564.

[90] K. PEARSON, *Liii. on lines and planes of closest fit to systems of points in space*, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2 (1901), pp. 559–572.

[91] I. R. PORTEOUS, *Geometric differentiation: for the intelligence of curves and surfaces*, Cambridge University Press, 2001.

[92] H. POTTMANN, J. WALLNER, Q.-X. HUANG, AND Y.-L. YANG, *Integral invariants for robust geometry processing*, Computer Aided Geometric Design, 26 (2009), pp. 37–60.

[93] T. PRIBANIC, Y. DIEZ, S. FERNANDEZ, AND J. SALVI, *An efficient method for surface registration.*, in VISAPP (1), 2013, pp. 500–503.

[94] T. PRIBANIĆ, Y. DIEZ, F. ROURE, AND J. SALVI, *An efficient surface registration using smartphone*, Machine vision and applications, 27 (2016), pp. 559–576.

[95] T. PRIBANIĆ, S. MRVOŠ, AND J. SALVI, *Efficient multiple phase shift patterns for dense 3d acquisition in structured light scanning*, Image and Vision Computing, 28 (2010), pp. 1255–1266.

[96] D. RON, *Property testing*, COMBINATORIAL OPTIMIZATION-DORDRECHT-, 9 (2001), pp. 597–643.

[97] F. ROURE, Y. DÍEZ, X. LLADÓ, J. FOREST, T. PRIBANIC, AND J. SALVI, *An experimental benchmark for point set coarse registration*, in Int. Conf. on Computer Vision Theory and Applications, 2015.

[98] F. ROURE, X. LLADÓ, J. SALVI, T. PRIBANIC, AND Y. DIEZ, *Hierarchical techniques to improve hybrid point cloud registration*, in Int. Conf. on Computer Vision Theory and Applications, 2015.

[99] S. RUIZ-CORREA, L. G. SHAPIRO, AND M. MELIA, *A new signature-based method for efficient 3-d object recognition*, in IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, 2001, pp. I–769.

[100] S. RUSINKIEWICZ AND M. LEVOY, *Efficient variants of the icp algorithm*, in IEEE International Conference on 3D Digital Imaging and Modeling, 2001, pp. 145–152.

[101] R. B. RUSU, N. BLODOW, AND M. BEETZ, *Fast point feature histograms (fpfh) for 3d registration*, in Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, 2009, pp. 3212–3217.

[102] R. B. RUSU, N. BLODOW, Z. C. MARTON, AND M. BEETZ, *Aligning point cloud views using persistent feature histograms*, in Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, 2008, pp. 3384–3391.

[103] S. SALTI, F. TOMBARI, AND L. D. STEFANO, *A performance evaluation of 3d keypoint detectors*, in IEEE International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission, 2011, pp. 236–243.

[104] J. SALVI, C. MATABOSCH, D. FOFI, AND J. FOREST, *A review of recent range image registration methods with accuracy evaluation*, Image and Vision Computing, 25 (2007), pp. 578–596.

[105] J. SANTAMARÍA, O. CORDÓN, AND S. DAMAS, *A comparative study of state-of-the-art evolutionary image registration methods for 3d modeling*, Computer Vision and Image Understanding, 115 (2011), pp. 1340–1354.

[106] R. SCHNABEL AND R. KLEIN, *Octree-based point-cloud compression.*, in SPBG, 2006, pp. 111–120.

[107] G. C. SHARP, S. W. LEE, AND D. K. WEHE, *Icp registration using invariant features*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 24 (2002), pp. 90–102.

[108] P. SHILANE, P. MIN, M. KAZHDAN, AND T. FUNKHOUSER, *The princeton shape benchmark*, in Shape Modeling Applications, 2004. Proceedings, 2004, pp. 167–178.

[109] I. SIPIRAN AND B. BUSTOS, *Harris 3d: a robust extension of the harris operator for interest point detection on 3d meshes*, The Visual Computer, 27 (2011), pp. 963–976.

[110] J. SUN, M. OVSJANIKOV, AND L. GUIBAS, *A concise and provably informative multi-scale signature based on heat diffusion*, in Computer Graphics Forum, vol. 28, 2009, pp. 1383–1392.

[111] H. SUNDAR, D. SILVER, N. GAGVANI, AND S. DICKINSON, *Skeleton based shape matching and retrieval*, in Shape Modeling International, IEEE, 2003, pp. 130–139.

[112] A. TAGLIASACCHI, H. ZHANG, AND D. COHEN-OR, *Curve skeleton extraction from incomplete point cloud*, in ACM Transactions on Graphics (TOG), vol. 28, ACM, 2009, p. 71.

[113] G. K. TAM, Z.-Q. CHENG, Y.-K. LAI, F. C. LANGBEIN, Y. LIU, D. MARSHALL, R. R. MARTIN, X.-F. SUN, AND P. L. ROSIN, *Registration of 3d point clouds and meshes: a survey from rigid to nonrigid*, IEEE transactions on visualization and computer graphics, 19 (2013), pp. 1199–1217.

[114] J. W. TANGELDER AND R. C. VELTKAMP, *A survey of content based 3d shape retrieval methods*, in IEEE Shape Modeling Applications, 2004, pp. 145–156.

[115] J.-P. TAREL, H. CIVI, AND D. B. COOPER, *Pose estimation of free-form 3d objects without point matching using algebraic surface models*, in IEEE Workshop on Model-Based 3D Image Analysis, 1998, pp. 13–21.

[116] F. TOMBARI, S. SALTI, AND L. DI STEFANO, *Unique signatures of histograms for local surface description*, European Conf. on Computer Vision, (2010), pp. 356–369.

[117] G. TURK AND M. LEVOY, *Zippered polygon meshes from range images*, in ACM Transactions on Graphics, 1994, pp. 311–318.

[118] O. VAN KAICK, H. ZHANG, G. HAMARNEH, AND D. COHEN-OR, *A survey on shape correspondence*, in Computer Graphics Forum, vol. 30, 2011, pp. 1681–1707.

[119] J. VANDEN WYNGAERD AND L. VAN GOOL, *Automatic crude patch registration: toward automatic 3d model building*, Computer Vision and Image Understanding, 87 (2002), pp. 8–26.

[120] S. WINKELBACH, S. MOLKENSTRUCK, AND F. M. WAHL, *Low-cost laser range scanner and fast surface registration approach*, in Pattern Recognition, 2006, pp. 718–728.

[121] C. YANG AND G. MEDIONI, *Object modelling by registration of multiple range images*, Image and Vision Computing, 10 (1992), pp. 145–155.

[122] Y.-L. YANG, Y.-K. LAI, S.-M. HU, AND H. POTTMANN, *Robust principal curvatures on multiple scales*, in ACM International Conference Proceeding Series, vol. 256, 2006, pp. 223–226.

[123] T.-H. YU, O. J. WOODFORD, AND R. CIPOLLA, *A performance evaluation of volumetric 3d interest point detectors*, International Journal of Computer Vision, (2013), pp. 1–18.

[124] A. ZAHARESCU, E. BOYER, K. VARANASI, AND R. HORAUD, *Surface feature detection and description with applications to mesh matching*, in IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 373–380.

[125] Z. ZHANG, S. H. ONG, AND K. FOONG, *Improved spin images for 3d surface matching using signed angles*, in IEEE International Conference on Image Processing, 2012, pp. 537–540.

[126] Y. ZHONG, *Intrinsic shape signatures: A shape descriptor for 3d object recognition*, in IEEE Int. Conf. on Computer Vision Workshops, 2009, pp. 689–696.