Bottom-Up Approach to Spatial Datamining

Thèse présentée à la Faculté des sciences économiques

Institut du management de l'information

Université de Neuchâtel

Pour l'obtention du grade de docteur en informatique

Par

Christophe Künzi

Acceptée sur proposition du jury :

Prof. Kilian Stoffel, directeur de thèse, rapporteur

Prof. Jean-Marie Grether, président du jury

Prof. Jacques Savoy, rapporteur

Prof. Galina Bogdanova, rapporteur

Université de Neuchâtel

soutenue le 3 mai 2013

UNIVERSITÉ DE
NEUCHÂTEL

IMPRIMATUR POUR LA THÈSE

Bottom-Up Approach to Spatial Datamining

Christophe KÜNZI

UNIVERSITÉ DE NEUCHÂTEL

FACULTÉ DES SCIENCES ÉCONOMIQUES

La Faculté des sciences économiques,
sur le rapport des membres du jury

Prof. Kilian Stoffel (directeur de thèse, Université de Neuchâtel)
Prof. Jean-Marie Grether (président du jury, Université de Neuchâtel)
Prof. Galina Bogdanova (Bulgarian Academy of Sciences)
Prof. Jacques Savoy (Faculté des sciences, Université de Neuchâtel)

Autorise l'impression de la présente thèse.

Neuchâtel, le 29 mai 2013

Le doyen	Le vice-doyen
Kilian Stoffel	Claude Jeanrenaud

■ Téléphone : +41 32 718 15 00 ■ Fax : +41 32 718 15 01 ■ E-mail : secretariat.seco@unine.ch ■ www.unine.ch

Mots clés : indexage, reconnaissance 3D, nuage des points, scene spatiale, interprétation spatiale.

Keywords: indexing, recognition 3D, point cloud, spatial scene, spatial interpretation

Acknowledgments

I would like to thank my professor Kilian Stoffel for his constant help and guidelines to pass the difficult path of my Ph.D. thesis.

I think that this work is not an individual achievement, but the achievement of the whole group I was a member during five years. Besides the true friendship I had with all of them, I would like specially to thank Paul Cotofrei. His point of view was very interesting in a lot of very diverse subject, I could learn from his universal culture. I would like to thank Claudia Ciorascu. In the first states of the thesis I was motivated by the enthusiasm of Claudia Ciorascu for the research. I was impressed and learned a lot in watching Iulian Ciorascu especially in the way of his usage of Linux amazing possibilities. Furthermore, the friendship of Eric Simon helped me to survive the stress and he showed me how to write better.

I would like to thank my parents, who teach me moral values.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Overview

One of the goals of computer vision research is to design systems that provide human-like visual capabilities such that a certain environment can be sensed and interpreted to take appropriate actions. Among the different forms available to represent such an environment, the 3D point cloud (unstructured collection of points in a three dimensional space) rises a lot of challenging problems. Moreover, the number of 3D data collection drastically increased in recent years, as improvements in the laser scanners technology, together with algorithms for combining multiple range and color images, allowed to accurately digitize any amount of 3D scenes. Because of these developments, some important digitalization projects - like the digital Michelangelo projet [54] or the digitalization of the Pantheon - were achieved. The last project, conducted by the Karman Center[1], generated a 3D digital model (available as a validation data set for our research study) containing more than 620'000'000 points.

If the universe (or unstructured space) is given by all 3D points - generated by the acquisition device, then calibrated, registered, and finally stocked in a spatial database system - then a scene is a limited region of this universe, having a regular geometric form and containing (un)known 3D objects. The interpretation of a scene is defined as learning *which* model is located *where* in the scene. Such an interpretation binds the entities in the scene to the models that we already known. Following the recent trend consisting in applying the AI point of view on Computer Vision problems, we adopt an extended definition of the "interpretation" task (closed to what was denoted as "high-level scene interpretation" [65]): it consists in the construction of a symbolic description including scene elements (objects or higher-level entities) and predicates (class

[1] http://www.digitalpantheon.ch/BDPP0715/BDPP0715.pdf

memberships and spatial relationships between these elements). This extension, which implicitly bear prior knowledge about spatial relations, allows the acquisition of a new kind of knowledge (the *semantic content*), concerning the possible regular patterns of objects spatial distributions. Furthermore, by defining a spatial description language as the set of models and spatial relationships, the shortest description of the scene in this language (in terms of existing objects and spatial relations between) defines the concept of *optimal scene interpretation*. Actually, even if the storage is not a problem anymore and tools for visualizing, streaming and interacting with 3D objects are readily available, there is still a big lack of methods for coding, extracting and sharing the semantic content of 3D media. Therefore, the overall goal addressed by this thesis is the development of a flexible approach (including framework, methodology, processing methods and finally a working system) that could help us to extract the semantic information of a spatial scene. A lot of work related to this idea has been done but most of it was dedicated to geographic information systems (GIS). The increase of collected 3D data urges for developing new technics adapted to these kind of data.

In order to reduce the complexity of the scene interpretation process regarding the large diversity of real-world situations, the framework is based on the following assumptions:

1. the objects of interest are rigid, free-form objects (which exclude statistically defined shapes such as textures, fractals or other objects);

2. a description language, based on a set of predefined models and a set of selected spatial relationships, is defined and encoded as a set of ontologies (denoted the *semantic layer*).

The framework we propose here, denoted RRR [23] (for Represent, Recognize and Retrieve), brings solutions for some important processes concerning the efficient storing and fast and accurate retrieving of 3D points, the augmentation of 3D points with semantic, and the automatic extraction of semantic information.
Stated succinctly, the design of the RRR system involves a three stage processing:

i. Representation - for each basic object type, a compact and meaningful model, based on point cloud, is proposed;

ii. Recognition - the characteristics (spatial, geometrical) extracted from partial point cloud are compared with known models to identify the objects present in the scene;

iii. Retrieval - based on a spatial description language (encoded as a dynamically semantic layer) and using a reasoning engine, a complete scene description (the set of object instances and the set of spatial relationships between these instances) is generated.

Extracting semantic content is generally a difficult problem, but particularly more difficult when the recognition system needs to draw useful inferences from a point cloud, which in itself is not very informative. Among the important issues which are of interest for our thesis we may enumerate

- the *object shape characterization in the presence of noise* - implying the use of outlier robust representation approaches;
- the *size dimension of model database* - implying the selection of efficient indexing techniques or the development of efficient pruning strategies during the matching process;
- the *learning capability* - to avoid the application breakdown in the presence of unknown objects, the system must be able to learn the description of the new object and have the ability to represent and store such modeling information.

1.2 Thesis structure

The next chapter contains a detailed description of various types of representation for spatial information (with a particular interest on shapes). Firstly, the difference between unstructured points and some mathematical description of objects is explained, followed by a description of the data that comes from the laser scanned device (which consist of unstructured points). Thereafter it is shown how to extract information about the neighborhood of these points and how to construct the mesh associated to these points. Even if the mesh representation can be simplified with some algorithms, the objective is to use only simple mathematical representations for spatial information. Of particular interest is the mathematical representation of some simple, basic shapes, denoted complete forms, because an object in a scene usually is a composition of many complete forms. Such a composition can also be represented as shock graphs, which is a kind of skeleton of a shape. Other concepts and approaches discussed in this chapter are the partial representation and dynamic objects.

The third chapter is dedicated to the problematic of information storage and information retrieval in a spatial database. The performance of some high-level processes, as the computation of spatial relationships, depends directly on the performance of the indexing methodology. We will show how the indexes are constructed according to a particular kind of data. The performance of each of the indexing structure proposed to store spatial data depends on the time to perform basic operations like: *search*, *minimum*, *maximum*, *predecessor*, *successor*, *insert* and *delete*. To optimize the execution time of these queries, some conditions must be fulfilled: minimize the empty space referenced in these indexes, minimize the overlapping of region and minimize the distance between the indexes of two near regions. A good index structure must not only present optimal performances for the basic operations, but also for some possible extensions of these queries. Another parameter to be considered is that some indexing structures are able to store simple points while others are able to store more complicated objects described by a number of points. Therefore, the decision concerning the choice of the best index for the implementation of our framework is based on a depth analysis of the advantages and disadvantage of the indexing methods. The last section describes the implementation details of some new queries that we introduced as tools for the recognition procedure. These queries are used to extract particular basic 3D shapes (sphere, cylinder, torus) from a scene, on which a

rotation transformation is applied or not, and to return either the interior points from an object or only the hull of the object.

The fourth chapter is dedicated to the description of high-level representation models for spatial scenes. By selecting the ontology as main representation, it is possible to give a semantic meaning to the knowledge extracted from the scene. This approach allows the construction of a semantic layer including the basic knowledge about spatial geometry, the user vision and the high-level knowledge about the spatial scene (e.g. spatial relations). In order to generate a Geometrical Scene Description, conceived as a composition of objects and the spatial relations that exist between them, a spatial ontology is defined (implemented based on OWL-DL language). This complex ontology is composed of two groups of concepts: the upper (reference) group and the lower (user) group. While the reference ontology allows to represent basic (primitive) shapes, their position and some spatial relations, the user ontology allows to define compositions based either on primitive objects or on user-defined objects. To capture the human point of view for a scene description, we propose to introduce qualitative concepts representing contextual spatial relations. This is done by the use of linguistic variables (as $isBig$, $isSmall$, $isAverage$) for which the exact meaning is defined by the user through the properties of the point cloud. The last part of the chapter exemplifies the basics of reasoning about spatial regions (based on RCC-8 calculus) and some description logic used to implement rules inferring spatial knowledge.

The fifth chapter contains information about the implementation of the recognition process, i.e. the algorithms used to construct the ontology representation. After the description of some basic concepts used for recognizing spatial objects (the most important being the concept of local feature size), we concentrate on the different techniques used to find the normal-vectors to a surface represented by a set of points. The estimation of the curvature of a cloud of point is a parameter used by an important number of algorithms for spatial data, as rendering algorithms or shape reconstruction algorithms. The main detection algorithm which is fully detailed is the RANSAC algorithm, which extract basic shapes by randomly drawing minimal sets from the point cloud. We extend this algorithm by two versions designed to incorporate the information contained in the semantic layer. The first version is a global scene extraction algorithm designed to provide a concise description of a spatial scene, using ontologies as representation tool. The second version is an ontology-driven object detection algorithm, which uses at input a filter represented as an ontology description (a list of shapes together with the set of satisfied unary/binary spatial relations) in order to guide the search in the scene.

The sixth chapter presents the complete framework for the RRR system, a flexible and powerful tool allowing the mapping between the point clouds and a mathematical and/or a semantic description. One of the most important aspect for this framework is the way in which a user may interact with the ontology and the spatial scene. Consequently we develop some possible interaction scenarios allowing us to design a functional user interface. In the last part we conduct a number of experiments, on both simulated and real data (3D scan of the Pantheon in Rome), in order to validate the processes that can be realized with our framework.

Finally, the last chapter reviews the most important objectives of our research by resuming the original, proper contributions and presents also some future improvements.

1.3 Publications

Some of the ideas, concepts, approaches and solutions related to the main problematic of this thesis were already presented to the scientific community during the research period devoted to the thesis elaboration. A first proposed system for managing 3D objects using a semantic framework represented as a reference ontology was described in [19], paper presented at SAMT Workshop on Semantic 3D Media, Koblenz, 2008. This article also exemplified how a user can define new complex objects using the reference ontology and how such an object can be retrieved from the spatial database system through a set of predefined queries. A first description of the RRR system designed for optimal scene interpretation was presented in [23], at the Third Int. Conf. of Computer Graphics and Artificial Intelligence, Athens, 2009. The article focused specifically on the problematic of the three system stages (Representation, Recognition and Retrieval), with a particular accent on the appropriate theories used to implement the MDL Principle for scene interpretation. Finally, a synthesis of this project and a more detailed description of the framework for analysing a scene containing a large number of objects represented as unstructured point clouds was presented in [22], at the First Conf. "Digital Presentation and Preservation of Cultural and Scientific Heritage", Veliko Tarnovo, 2011.

2 Representation

The term *shape* has a wide variety of meanings and, implicitly, different types of representations. From a mathematical point of view - and for the purpose of this thesis - shapes can be seen as smooth manifolds in 3 dimensions, which implies that differential and integral calculus may be applied on these forms. The shapes can be further characterized by properties of object representations [14] such as *ambiguity*, *conciseness* and *uniqueness*. Depending on the kind of application, we may be interested only in some of these aspects. As example, for a recognition task we need very precise representations and information about the location of an object in order to distinguish different shapes. At the same time, to cover a large spectrum of applications, as much information as possible related to different shape aspects must be kept.

The starting point for shape representation/shape information is represented by the concept of point cloud.

2.1 Unstructured 3D Point Clouds

A point cloud is defined as a set C of unstructured points (3D points in our case). Having a set of n points, the point cloud is denoted by $C = (s_0, s_1, ..., s_{n-1})$, where each point is determined by m values $p = (x_0, x_1, ..., x_{m-1})$ (beside the basic geometric coordinates, a point may be characterized by various physical values, e.g. the color). These points often come from scanning devices and are generated in an automatic way. In mostly applications, the common problem of a point cloud is the number of points, i.e. the size of the cloud. In order to optimize the treatment of point clouds we need a compact representation for them, which can be achieved by sampling or by reorganizing them in form of meshes. [79]

2.1.1 Sampling

If we consider an object O represented by points and a sample S (i.e. a subset) of these points, then, in order to maintain all important features of the object O, the number of points in S has to be sufficiently dense. Therefore a measurement function that calculates the complexity of an object around each point of O must be defined [24].

One way to measure the complexity of an object is to use the *medial axis* or *topological skeleton*. The medial axis refers to the middle of the object O. By definition, a ball $B \in \mathbb{R}^3$ is denoted a maximal ball for the set O if $B \subseteq O$ and, for any ball D containing B, we have $D \nsubseteq O$ (or, equivalently, is a ball $B \subseteq O$ with the maximal radius that has empty interior). The medial axis can be constructed with the set of all the centers of all maximal balls. Another possibility is to use the *local feature size* which represents, at any point $x \in O$, the distance between x and the medial axis. Based on this concept it is possible to calculate a measure of the complexity of an object denoted *feature size*. In order to have have a good sample, the distance between the medial axis of the sampled object and the real object should be minimized.

The quality of the sample S can impact significantly the estimation of a particular object feature, the normal at the surface in a point $x \in S$ (see the Section 5.2).

2.1.2 Mesh

Usually the cloud of points is visually difficult to interpret. It is possible to add some information about the structure of the surface by constructing a mesh. The set of elements which could be triangles or quadrilaterals (in two dimensions), tetrahedra or hexahedra (in three dimensions) is called a mesh.

Triangle mesh

If the triangulation is done in 3D then this procedure is called a *tetrahedralization*. This will add a neighbourhood relation to the points, which is helpful for the recognition of objects. The tetrahedralization procedure creates a partition of the 3D input space. Formally the tetrahedralization consists of a finite set T of triangles having the following properties [53] :

1. every side of a triangle in T is common to an even number of triangles of T.

2. there is no finite decomposition of the triangles T into other triangles forming a set T' such that the (1) is true

For the same input different tetrahedralization are possible, as we can see in Figure 2.1. Any triangle of the tetrahedralization is called simplice [13]. An optimal triangulation is one which is the best according to the selected criterion (e.g. minimizing measures of the size, shape, or number of simplices).

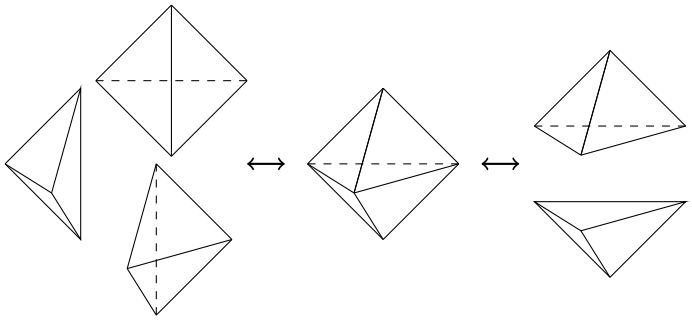

Figure 2.1: Two different tetrahedralizations

Voronoi diagram

The Voronoi diagram is a structure that is built on the neighborhood of a set of points. The Voronoi diagram is composed by cells V_p. Each cell defines the domain in which all points have no other closer point than those in p. If two cells are touching, we say they are neighbors. By building the Voronoi diagram, we can build the mesh.

Mesh construction

We can realize the construction of a mesh starting either with all the vertices of the input, or by adding extra vertices denoted *Steiner points*. These extra points can sometimes lead to better triangulations. There are basically two different types of triangulation mesh: *structured meshes* and *unstructured meshes*. The difference between them is that a structured mesh has for any vertex a fixed number of neighbors, therefore it is simpler.

Here we assume that we have a point cloud as input for the tetrahedralization process. The mesh construction is a two step procedure. In a first step, a tetrahedralization solution is performed; in a second step, this solution will be optimized. The optimal solution needs to be performed depending on the *type of triangulation* and on the *optimality criterion*. A very common criteria for mesh construction is to avoid sharp elements. But the optimality criteria could depend on other criteria related to tetrahedra measures, as the sum, the maximum or the minimum of the angles, the edge lengths, or the area of all triangles.

We know that a tetrahedralization with n vertices has at most $\binom{n-2}{2} = \frac{(n-2)(n-3)}{2}$ triangles. In the case of a strictly convex polyhedron the number of tetrahedra is at most $2n-7$. And any tetrahedralization has at least $n-3$ tetrahedra. For example, if $n=4$ there is only one tetrahedralization with 1 tetrahedra - here the minimum is also the maximum i.e. $n-3 = 2n-7 = 1$ (see Figure 2.2).

Sometimes in a point cloud there are non-convex forms and some of them are not tetrahedralizable. This is proved by exhibiting a polyhedron where no decomposition in tetrahedrons exists [53], as the famous example of the Schönhardt's polyhedron. But it was proved that some of them can be made tetrahedralizable if Steiner points are added. It has been shown by construction that $\Omega(n^2)$ is the lower bound for the number of

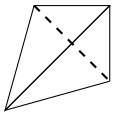

Figure 2.2: Tetrahedralization with 4 points

Steiner points that are needed for tetrahedralization, leading to the following theorem: *Any polyhedron can be triangulated with* $O(n^2)$ *Steiner points and* $O(n^2)$ *tetrahedra.* The proof can be found in [13].

Mesh refinement

We can distinguish three kind of mesh refinements : *global mesh refinement*, *semi-local mesh refinement* and *local mesh refinement*. In global mesh refinement *all* the elements in the mesh are refined to obtain a finer mesh. The local mesh refinement will use a test on the elements of the mesh: if an element corresponds to some criterion, the algorithm will refine the element into a set of finer elements. In the literature, this process is also called adaptive local mesh refinement, because the refinement happens locally.

Mesh optimization

Another interesting operation on tetrahedralization meshes is the simplification of the mesh. It is called mesh optimization or decimation. The algorithm used for optimization visits all the vertexes in the triangle mesh. For each vertex a measure will determine whether the vertex can be deleted. The criterion used for removing the vertex depends on the application in which the tetrahedralization is used. The process of optimization removes information. The question is what will be the approximation error acceptable for the new mesh. On way of having an optimal mesh is by computing interactively new meshes until no more global optimization can be done (process denoted *incremental mesh reduction*). From an abstract point of view, there are basically two different kinds of decimation:

1. building a new mesh without inhering the topology from the original one, and
2. obtaining the new mesh by modifying the original one without changing the topology.

For the incremental mesh reduction, any algorithm has at least three components: a *topological operator*, a *distance measure* and a *fairness criterion.* The topological operator is used to modify locally the mesh, the distance measure is used to check that the maximum tolerance is not violated and the fairness criterion evaluates the quality of the current mesh [48]. We describe the topological operations in the following paragraph.

Topological operations. There are two simple operations, defined by two operators: *vertex-removal* and *edge-collapse*. The vertex-removal is an operator that consists of two

action, *removing a vertex* v and *re-triangulation*. This operation changes the number of triangles from n to $n-2$. To know how to re-triangulate a local edge-swapping optimization is conducted.

The edge-collapse operator uses an edge $p-q$ and collapse both vertexes into one vertex r (computed using a local heuristic). As with the vertex-removal, this operator will remove two triangles. We can also define a third operation by combining these two simple operators in a single one, called the *half-edge collapse*. In this case a undirected edge $p-q$ can be seen as a combination of two directed parts, a directed half $p \rightarrow q$ and another $q \rightarrow p$. Collapsing the half-edge $p \rightarrow q$ implies to pull the vertex p to q and to remove the triangles that are becoming edges. This operation has the advantage to be a simple operation, with no need for further local optimization. This operator does not create new vertexes. The Figures 2.3, 2.4 and 2.5 present the different operators.

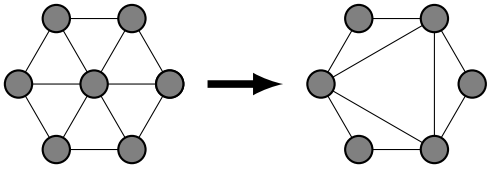

Figure 2.3: Topological operator: vertex-removal

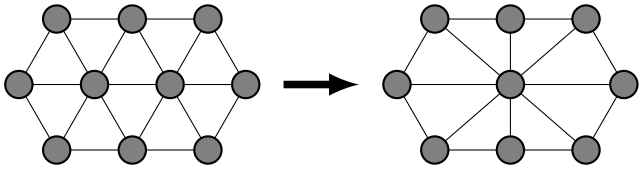

Figure 2.4: Topological operator: edge-collapse

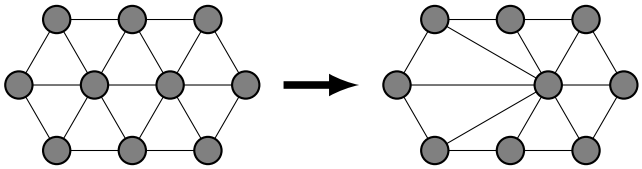

Figure 2.5: Topological operator: half-edge collapse

The Fairness Function. The fairness function is used to estimate the quality of a mesh. It can be used to formalize the optimization or reduction mesh problem. We can consider the problem of constructing a surface S whose maximum distance to the given set points P does not violate a given tolerance : $||S-P||_\infty \leq \epsilon$. A fairness function was developed in [46]: $F(S) = \sum_{p \in S} \alpha E(p) + \beta R(p) + \gamma S(p)$. It can be adjusted by changing the weights α, β and γ. By increasing α we increase the weight of the *local*

approximation error, meaning that we want to preserve more details; by increasing β we increase the weight of the *local distortion* which can be estimated by the ratio of inner sphere's radius to the largest edge. This leads to a reduced mesh, because a lower number of triangles implies a better fairness function. And by increasing γ we increase the weight of the local curvature which is estimated by the angle between two triangles.

2.2 Complete forms

In this part we will present the different types of representation of a three-dimensional object. As we showed before, an object can be viewed as a cloud of points $C = (s_0, s_1, ..., s_{n-1})$, containing n points, each point being defined by m values $p = (x_0, x_1, ..., x_{m-1})$. In the previous section we have presented the tetrahedralization process which constructs a finite set T of triangles with specific properties. Some other formal presentations of 3D objects are given in the following sections.

2.2.1 Static k-dimensional objects

A particular case of geometrical representations is the geometry of static objects, meaning that the description of those objects will not vary or change over time.

Parametric forms

In this representation the parameters are used to define forms. As example, a surface can be represented as a smooth vector function $s : \mathbb{R}^2 \to \mathbb{R}^3$, defined by three one-dimensional functions taking two arguments u and v, each of them being defined on the interval $[0, 1]$. A common parametric equation is the NURBS (nonuniform rational B-Spline), a generalization of the B-Spline.

$$S(u, v) = \begin{bmatrix} x = f(u, v) \\ y = g(u, v) \\ z = h(u, v) \end{bmatrix}$$

Figure 2.6: Parametric form for 3D surface

It is important to notice that the natural quadratic objects (the spheres, the cylinders and the cones) have an exact representation as NURBS.

Implicit surfaces

Another form representation is the implicit surface, defined as the set of zeroes for a shape-specific function $f(x, y, z)$. The equation $f(x, y, z) = 0$ describes the surface whereas $f(x, y, z) < 0$ describes the interior of the shape. It is similar to an iso-contour.

This kind of representation can be used with the MLS (moving least square method). The surface is specified by the moving least-squares solution to a set of constraints that

force the function to respect given values at the sample points and also force its upward gradient to match the assigned normals at the sample points.

Superquadrics

The superquadrics are a family of geometric shapes defined by formulas that are similar of ellipsoids and other quadratics, except that the squaring operations are replaced by arbitrary powers. In general they have the form: $|x|^r + |y|^s + |z|^t \leq 1$. One variation of this form considers the same power for all dimensions, i.e. $r = s = t$. This particular form can be defined as $|x|^u + |y|^u + |z|^u \leq 1$. The variations of u defines different shapes [44]:

- for $u < 1$: a pointy octahedron with concave faces and sharp edges;
- for $u = 1$: a regular octahedron;
- for $1 < u < 2$: an octahedron with convex faces, blunt edges and blunt corners;
- for $u = 2$: a sphere;
- for $u > 2$: a cube with rounded edges and corners;
- for $u \to \infty$: a cube.

General free-form objects can be created in the same way. However specific local features can not be captured using this approach.

Generalized Cylinder

Another example is the generalized cylinder, which can be seen as a deformed cylinder. The idea of this representation is to define the middle of the deformed cylinder as a curve $A(s)$. This middle curve (denoted the axis of the shape) is the central-spine of the generalized cylinder. The cross-section $C(s, \theta)$ defines the distance from any point of the axis to the border of the shape. This function will define the boundaries of the primitive along the axis. Different generalized cylinder defined by those two functions $gc(A, C)$ can be combined to define complex objects.

2.2.2 Dynamic k-dimensional objects

Some objects can't be described by the previous models because they are either deformable, or they change over time. The human body is an example of such an object which is composed of different moving parts. In the next sections we will give some examples of these objects and ways to model them.

Articulated objects

The human body or his hand are examples of articulated objects [1]. They can be decompose into rigid parts connected by joints. We call any rigid part of an articulated object a *component*. Once we have decomposed the articulated objects in its components and joints, we can model them with either the generalized cylinder and/or the superquadrics approach.

Deformable objects

Another even more complicated category of object are the deformable objects. This kind of object are also characterized by their material properties. This class of object contains for example a *beating heart* or *stalks blowing in the wind*. There are roughly two main techniques to represent this objects. The first one consists of using finite element models (FEM) to discretize the object into small connected elements, called *finite elements*. The spatial configuration of any system is described by its (finite) degrees of freedom (DOF). Generally, the degrees of freedom is represented by a column vector, called a *DOF vector* or a *state vector*.

The second technique is the *mass-spring mesh*, which represents the object as a set of point masses connected by elastic links.

2.3 Partial representations

In a general setting where the detection of objects is the main goal, we have to examine some partial representations of objects. There are two approaches to simplify the partial representation of objects: *Curves* and *Skeletons*. Even if these representations are not complete, they have the particularity to simplify the data for the recognition process. In the following subsections, we will briefly describe these partial representations.

2.3.1 Curve

One particular technique to describe an object is based on its features. This means that for each object we are looking to find particular curves. An example of a curve matching algorithm is given in [91]. One very well-known short-coming for this method is its difficulties with the representation of the interior of the shape.

2.3.2 Skeleton/Shock graphs

Non-formally, the skeleton refers to the "central-spine" of the shape. In the literature, the concept of skeleton has different mathematical definitions (straight skeleton, morphological skeleton, skeleton by influence zones) and is computed by various algorithms.

The skeleton of a point cloud S is defined by the set of all centers of the maximal balls[1]. After the creation of the skeleton (using the thinning algorithm which successively

[1] A disk/ball B is maximal for in set S if $B \subseteq S$ and if, for every disk/ball $D \supseteq B$, we have $D \not\subseteq S$

erodes the borders of the shape), the shock graph can be extracted. This is done by creating nodes and branches corresponding to the skeleton topology. The shock graph is a simpler form used to make the comparison between two objects.

2.4 Conclusions

In this chapter we saw some techniques for representing spatial information. Each of these representations has its advantages and disadvantages. As we will see in the following chapter, the partial representations are helpful for recognition algorithms, by improving their performance for storing and comparing objects. The complete forms are more precise, but they are not easily applicable to point clouds. The mesh approaches are particularly well suited for visualization purposes.

3 Indexing and Queries

3.1 Indexing

In recent years, databases had to adapt their structure to multi-dimensional data. This is particularly true for the indexing mechanism used in database. One type of data that has caused issues is multidimensional or spatial data, represented by items characterized by 2-D, 3-D or higher dimensional space coordinates. A spatial database system (SDS) is a general purpose software that manages the storage and retrieval of spatial data, ensures the consistency of data, and provides the corresponding tools. There exist mainly two kind of structures to be indexed in a SDS, namely multi-dimensional points and multi-dimensional structures.

The efficiency of the queries for retrieving data is based on their spatial characteristics and depends on a structure called an *indexing structure*. Even if the index optimizes the retrieval, the performance of the index depends also on the data set properties, as:

- *The number of spatial objects per unit of space.* If the distribution of objects in space is irregular then this could result in some denser areas and causing overlaps in the indexing structure.
- *The size of objects.* Large objects cover a large amount of space, which can result in a denser space usage.
- *Database Size.*
- *Frequency of updates.*

An index for multi-dimensional data provides means for fast retrieval of points defined by location in a multi-dimensional space. The creation of an index is one of the most

important components of a DBMS (DataBase Management System). The DMBS is a complete system allowing among others to interact with the index through a query system.

Usually, spatial data is large in size and can be complex in structure (polygons and volumetric objects). Furthermore spatial data structures may present relations between them (such as *intersection*, *adjacent to*, *contain in*, etc..). Efficient processing of queries manipulating spatial relationships, called *spatial queries*, relies upon auxiliary indexing structures. Due to the normally large volume of spatial data, it is highly inefficient to pre-compute and store spatial relationships for all the data objects (although there are some proposals in the literature that store pre-computed spatial relationships [59]). Instead, spatial relationships are most of the time dynamically calculated during the query process. In order to find spatial objects efficiently, it is essential to have an efficient spatial indexing structure.

There is a large number of spatial indexes proposed in literature. A complete description of the different index structures, together with their strengths and weaknesses, can be found in the review paper of Ooi *and all* [72]. Therefore, to keep an unitary view of the various classes of index structures we decided to follow the taxonomy proposed by the authors.

The underlying indexing structure must support efficient spatial operations such as locating objects in the neighbours of a given object or identifying objects in a defined region. Data structures of various types, such as B-trees [9, 21], ISAM indexes, hashing and binary trees [47], have been used to assure efficient database operations (access, insertion, deletion). These structures allow to index data based only on primary keys. In order to index data based on secondary keys, inverted indexes are used, but they are not suited for a database where range searches are common operations. For multi-dimensional structures, grid-files [68], multi-dimensional B-trees [52, 76, 89], kd-trees [11] and quad-trees [33] where proposed to index multi-attribute data.

The point indexing structures allow to index spatial objects defined as points in a multi-dimensional space. Other techniques used to extend point indexes to accommodate spatial objects can be categorized into three classes of indexes, as proposed in [92]:

- *Object mapping index* - in this approach, an object defined by n-vertices from a k-dimensional space is mapped into points in a nk-dimensional space or as single-value objects in the original k-dimensional space.

- *Object duplication index* - in this approach, the object identifier is stored in multiple data spaces covering the whole object.

- *Object bounding index* - in this approach, data objects are partitioned into different groups using hierarchical grouping technique.

Each approach has its own strengths and weakness, which directly affects the performance. Based on its basic structure, spatial indexes may be classified in different categories. In the next sections the most common indexes will be explained in more details.

3.1.1 Binary-Tree indexing techniques

By definition, a binary tree is a tree where each node has at most two child nodes, usually designated as "left" and "right". The first node of the tree is denoted the "root" node (the ancestor of all nodes). Any node in the tree can be reached by a path starting at root node. For implementing indexing, we use the binary search tree, which is a binary tree data structure having the following properties:

- The left subtree of a node contains all nodes those keys are less than the node's key.
- The right subtree of a node contains all nodes those keys are greater than the node's key.
- Both subtrees (left and right) are binary search trees.

The general idea is to repetitively partition the data space into subspaces. The partitioning is done when the maximum capacity of a node is reached. The binary search trees commonly implement the following set of operations: *search*, *minimum*, *maximum*, *predecessor*, *successor*, *insert* and *delete*.

kd-Tree

A *kd*-Tree is a binary search tree in which every node is a k-dimensional point [11]. The sub-nodes which are non-leafs represent a splitting hyperplane that divides the space into two parts, known as subspaces. For point clouds the nodes represent 3D plans. It has been showed that the deletion cost of an element in the *kd*-tree is high. To reduce the cost of deletion, a non-homogeneous *kd*-Tree was proposed [10]. Many other variants of the *kd*-Tree have been proposed in the literature [7, 28, 85, 94] in order to improve the performance of the *kd*-Tree. These variants try to solve some issues such as clustering, searching, storage efficiency and balancing. In the Figure 3.1 one can see an example of the *kd*-Tree structure and the corresponding subspaces divisions.

The grid-file

A typical file structure is designed to manage a storage allocation on disk in units of fixed size, called disk blocks, pages, or buckets, depending on the level of description. A bucket is a reference to a spatial region which is a part of the space containing the objects in this subspace. For storing the data, we consider these buckets to be unlimited. Each bucket has normally a capacity c between 10 and 1000 units, and for each of these buckets the data is stored in a large linearly ordered list.

The idea for a grid file [69] is to construct a n-dimensional grid, where n represents the number of different attributes. For spatial data we construct a 3-dimensional grid. The grid-file is an extension - allowing more flexibility and better performance - of the fixed-grid (a grid having the n dimensions equally divided, adapted for uniformly distributed data).

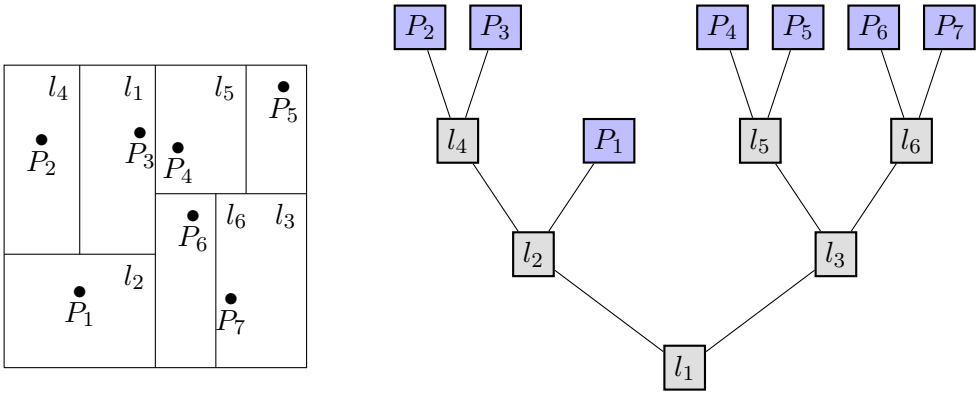

Figure 3.1: A *kd*-Tree with $k = 2$ and the corresponding space division and points. The $l_1, ..., l_6$ are corresponding to the space divisions and the $P_1, ..., P_7$ are the points in the area. representation.

The grid-file decomposes the index into two parts. The first part consists of a n-dimensional grid directory, where each directory points to an actual bucket containing the data to retrieve.

3.1.2 B^+-Tree based indexing techniques

The difference between a binary search tree and a B-Tree is that the B-Tree is balanced. In a B^+-Tree, in contrast to a B-Tree, all records are stored at the leaf level of the tree, whereas keys are stored in interior nodes. The B^+-Trees have been used for systems implying a large number of query retrieval. This data structure is characterised by height-balance, which makes it ideal for disk operations where data transfer is the major bottleneck. It has become a basis for many the newer indexing structures. However, B^+-Trees are not adapted to index three-dimensional or higher dimensional objects. To overcome this problem R-Trees were introduced as a multi-dimensional generalization of the B-Trees that preserve height-balance of the tree structure [37]. This structure uses bounding rectangles in the non-leaf nodes covering all rectangles in the lower nodes. It is quite common for several covering rectangles to overlap in internal nodes. The R^+-Tree is an improved R-Tree, which uses a criterion that ensures the quadratic cost in the insertion and splitting algorithms (Quadratic Cost Algorithm [37]). The improvement is particularly significant when both the query rectangles and data rectangles are small and when the data is not uniformly distributed. It's somehow a compromise between the R-Tree and the *kd*-Tree.

The goal of this approach is to avoid overlapping of the internal nodes by splitting an object over multiple leaves. The buddy-tree, proposed by [93], is also a compromise, but this time between the R-Trees and the grid-files. Based on the binary space partitioning trees [36], it alleviates the overlapping bounding rectangle problem of the R-Tree and the "dead space" problems of the R^+-Trees.

The TV-Trees were proposed to avoid the search in overlapping areas [55]. It uses the classification principle. To classify an object, few features are used firstly; and if further refined is needed, more and more features are gradually used.

The X-Tree is an extension of the R-Tree which uses the concept of super-nodes (nodes containing more than one point). The splitting algorithm creates these super-nodes if the number of points is high and the risk of overlapping is big.

Two of the main problems of the B^+-Tree based approaches are overlapping and empty space. The overlapping is defined as the percentage of volume that is covered by more than one bloc in the tree. The problem of overlapping is increasing if the dimension is high and is shared by all the indexing structures based on B^+-Trees. In the Figure 3.2, we can see the point cloud representation of the Pantheon and the corresponding minimum bounding region (in this case hyper-rectangles). For each level of minimum bounding box, we changed the color between green and red. The index should also minimize the empty space to be more efficient.

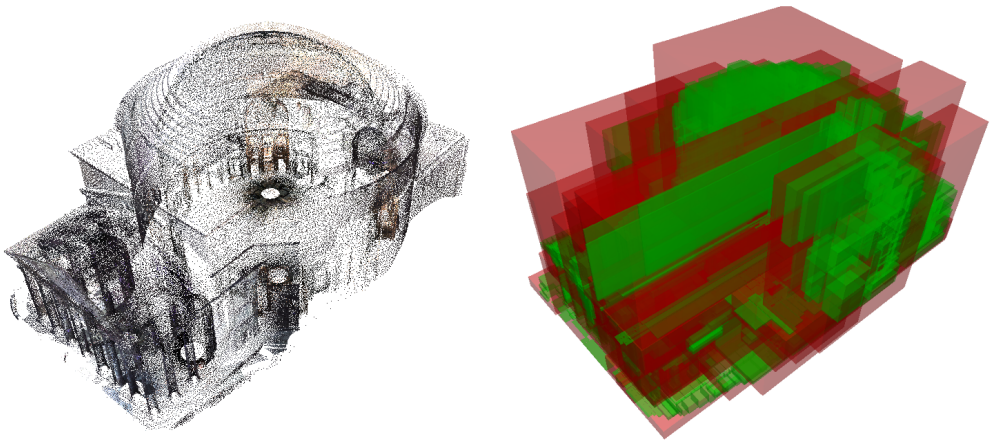

Figure 3.2: The point cloud representation of the Pantheon and the corresponding minimum bounding regions in the X-Tree representation

KDB-Tree

The KDB-Tree [85] is a combination of a kd-Tree and a B-Tree. It partitions search spaces in a manner similar to a kd-Tree: a search space is divided into two subspaces based on comparison with some element in the whole domain. Like as a B-Tree, it consists of a collection of pages and a variable root ID that gives the page key (ID) of the root page. There are two types of pages in the KDB-Tree: region pages, which contain a collection of (region, key) pairs, and point pages, which contain a collection of (point, key) pairs.

hB-Tree

This structure is based on the *KDB*-tree (the space is split using hyper-planes). The internal structure is a *kd*-Tree. But it allows data space to be holey. To create the holey nodes, several leaves of the *kd*-Tree may refer to the same space. In the Figure 3.3 you can see an example of a holey brick.

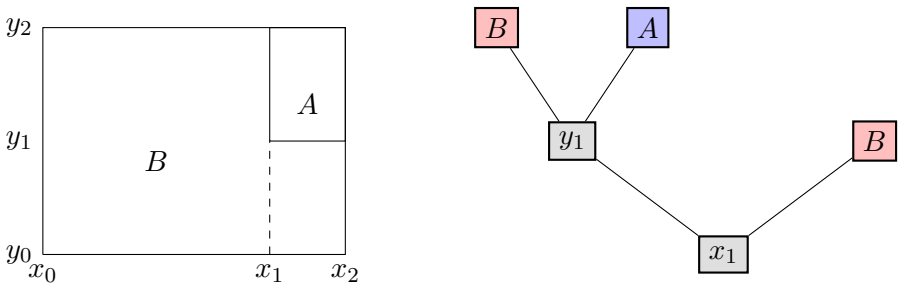

Figure 3.3: A representation of a holey brick: on the left the graphical representation and on the right the *kd*-tree representation

The *hB*-Tree generalizes the *kd*-Tree because the attribute and the comparison value is stored in a *kd*-Tree node. Splitting a node may require the participation of more than one attribute [57, 58].

sdk-Tree

The *sdk*-Tree stands for spatial *kd*-Tree [71]. One problem in the standard *kd*-Tree comes from indexing non-zero sized objects. These kind of objects could overlap in two distinct subspaces. The overlapping of the object on two subspaces implies for a *kd*-Tree to be indexed in both sub-nodes representing the subspace. The *sdk*-Tree also introduces a virtual subspace for each original subspace. The objective is to include all objects only in one of the two virtual subspaces, to avoid objects division and identifiers duplication. This approach increases the performances for the queries in such an index.

BD-Tree

The *BD*-Tree [70] is a variant of the *kd*-Tree. It allows a more dynamic partitioning of the space. A variable-length string, denoted *discriminator zone* (DZ) is stored in each non-leaf nodes of the tree. This variable describes (uniquely) the left subspace while the right subspace is its complement. The differences between a *BD*-Tree and a *kd*-tree are related to the form of data space (not limited to hyper-rectangle for *BD*-Tree), the size of partition components (equally for *BD*-Tree) and the use of DZ expression.

LSD-**Tree**

The *LSD*-Tree (Local Split Decision Tree) is a binary tree proposed in [38]. If the internal structure is the same as for the *kd*-Tree, the improvement is given by the free choice of the split position which is locally optimal. In each node the dimension used for splitting and the split value are kept. Moreover, the splitting strategy can be adapted to the application (which is not the case for all the index structures).

3.1.3 Quad-tree based structures

The difference between binary-trees and quad-trees is that the quad-trees have exactly four children in each node. These nodes can be seen as four quadrants labeled in order NW, NE, SW and SE. They can be differentiated on the following bases: the type of data represented (in our case 3-dimensional points), and the decomposition process. The decomposition can be executed into equal parts on each level or depending on the input.

A variant of quad-tree is the quad-CIF-tree [88], where "CIF" stands for Caltech Intermediate Form. The goal is to locate/extract rapidly the collection of all objects that intersects a given rectangle. It was proposed for representing a set of small rectangles for VLSI applications. A region is repeatedly subdivided into four equal-sized quadrants until blocks are obtained that do not contain rectangles. When one division is made, all the rectangles that are passing through the lines that decomposes the node are placed in the node. It is organized in a way similar to the region quad-tree, with the assumption that no two rectangles overlap.

There are two main differences between a quad-CIF-tree and a quad-tree:

1. In a quad-CIF-tree, the data can be contained both in leaf nodes and in non-leaf nodes.

2. The quad-CIF-tree is used to store rectangles rather than points.

3.1.4 Cell methods based on dynamic hashing

Consider a collection of records, each one identified by a key. This collection is stored in a dynamic file (on which two methods - insert and delete - may be applied). The dynamic hashing methods are based on hash tables, where the key idea is to transform a key of a record into an address. Usually, the hash tables have two main disadvantages. Firstly, hashing cannot support sequential processing of a file according to the natural order of the keys. If we want a sequential process, we need to sort the data, which demands $O(nlog(n))$ operations. Secondly, the traditional hash tables are not extensible, because the size of the hash table is dependent of the hash function. If we use a size too low, it may be necessary to rehash: create a new table and a hash function and relocate the records.

Both extensible hashing [32] and linear hashing [56, 51] lend themselves to an adaptable cell method for organizing k-dimensional objects. A cell in this case can also be called bucket and is a directory of records.

The principle of extensible hashing is to create a hashing function that points to a large address space. We define a partition of this space of m elements. It implies that a leaf of the partition contains all the keys k with $\alpha_{i-1} \leq h(k) \leq \alpha_i$. If one leaf is full, we can re-organise the partition without rehashing. This can be done by a small shift and relocation of only the records that are affected by the shift.

The linear hashing creates at the beginning (level 0) N buckets. The addresses of the buckets are defined as $address(level, key) = hash(key) \cdot modulo(N * 2^{level})$. We define a variable named *split*. If we have $h_i(k) > split$ then the key k is put in $h_{i+1}(k)$. The *split* variable is incremented by 1 at the end of the expansion operation. If the *split* variable reaches $N \cdot 2^{level}$, then the *level* variable is incremented by 1, and the *split* variable is reset to 0.

The grid file [41, 67] and the EXtensible CELL (EXCELL) method [98] are extensions of dynamic hashed organizations incorporating a multi-dimensional file organization for multi-attribute point data.

3.1.5 Spatial Objects Ordering

The actual DBMS are efficient for one-dimensional data by implementing one-dimensional indexes. Therefore, if an object multi-dimensional can be transformed into one-dimensional object, the same indexes can be applied directly. For this reason a mapping functions has to be defined to preserve the proximity between data well enough in order to yield reasonably good spatial search.

Ordering based on space-filling curves.

A particular category of methods for mapping k-dimensional space onto one-dimensional objects in its native space are those who use the Peano space-filling curves [80]. It was proposed in [75]. The basic idea consists to transform each k-dimensional object into a set of line segments. The transformation allocates to each spatial location a number. One of the first order defined is the Z-order curve introduced in [74] (the name 'Z' comes from the shape drawn on the space). In the Figure 3.4, we can see some of the different curve representations. As a compromise between B-Trees and the use of space-filling curves, the UB-Tree (Universal B-Tree) [8] is a variant of the B-Tree, where the keys idea is to use ordered regions.

We need to apply two transformations to a k-dimensional object to compute the address in the space defined by a Peano, Z-order or an Hilbert curve [40]. The first transformation will transform the space of the k-dimensional objects in the space of the curve. The second transformation maps the address from the space curve to a one-dimensional space. The computations of the Hilbert curve involves rotation and reflection in its basic pattern. And even if the necessary computation is greater than for the Peano curve, the Hilbert curve was shown to produce better result for indexing and extracting data.

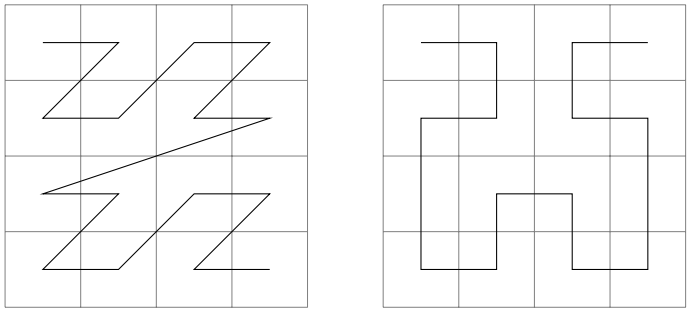

Figure 3.4: Space orders: on the left the Z-order and on the right the Hilbert-curve

3.1.6 Summary

To summarize this section, we have to remark that no structure can claim to be the best indexing structure in all situation. In order to select the best indexing structure, benchmarking is necessary. Data sets and methods to measure the performance must be put in place. The following parameters on data distribution are likely to affect the performance of the proposes indexes :

- The number of spatial objects per space unit: the irregularity distribution of objects in space, meaning that some areas are more dense than the others, may reduce the performance.
- The size of the objects: large objects can lead to a denser space occupation. In distributions where there are many large objects, the performance of indexes based on object bounding and object duplication might be highly affected.

The performance tests we performed on some of the most efficient indexing structures, based on the point cloud representation of the Pantheon as data set, clearly indicated the X-Tree as the best choice for an efficient implementation of a 3-D spatial database.

3.2 The Concept of Spatial Query Language

3.2.1 Introduction

Once the data has been stored into an efficient indexing structure, we need some languages to extract information from it. It provides the interaction with the database and it is required for any DBMS (Database Management System).

The SQL langage has become a standard for all queries in database systems. Even if SQL is a very good approach for retrieving data in general, the basic implementation of SQL langage doesn't support geographic or spatial location. For 3-dimensional data, it is not sufficient to filter the information through conditional predicates. Some extension to the SQL langage have to be implemented to improve the search in spatial data, for example find all the points in a certain area.

A query language for 3D data must be able to extract data for a location or a shape and must be be able to use topological operators. In the following section we will give a brief overview on the different languages proposed to access specific spatial data. We will see the advantages and disadvantages of these languages and see some proposed improvements.

3.2.2 Topological operator

In this section, we will see the different operators that can be used to express relations between objects in a spatial scene. There is a wide range of operators that describe relations between objects. An object can be just next to another (i.e. they *meet*). Among other topological operators between two objects we may enumerate: *disjoint*, *contains*, *inside*, *equal*, *covers*, *coveredBy* and *overlap*. If the two objects are identified by the two regions A and B, the semantic of these operators are:

- *disjoint*: no part of the region A is touching or is in region B.
- *equal*: the region A has exactly the same description as the region B.
- *overlap*: some part of the region A is included in the region B.
- *contains*: the region A is completely inside the region B.

An extension of these relations has been proposed by Reis [84]. Initially, these topological operators were defined to express relations between lines. But they can also be used to express queries looking for objects for which the relations hold.

3.2.3 Spatial Query Language

The traditional database query languages, such as (SQL) [15], Quel [97] or QBE (Query By Example) [103], are not adapted to processing spatial data because they do not consider spatial attributes. Therefore new query languages had to be developed. As underlined in [29], spatial data has additional properties (geometrical and graphical), which the user must be able to use in a query language. A spatial query language must include spatial relationships/operators (see [86]). The extensions of existing (relational) query languages to spatial data are based on the idea that: Relational Query Language + Spatial Relationships = Spatial Query Language. Some examples are GEO-QUEL [12], Query by Pictorial Example [16], (a Query-by-Example extension), or a QUEL extension of image processing [30]. But the most significant and numerous extensions are based on SQL. In the following sections, we will review some of them.

GEOQL

GEOQL fully preserves the SQL structure and adds to the standard definition of SQL the concept of geometry in terms of the bounding lines of spatial objects, spatial operators between geographic objects, and windows. The windows are defined by a coordinates

system. The GEOQL language decomposes the queries into a set of sub-queries. This implies that the queries that are non-spatial can be executed with the standard SQL language. The spatial queries are then sent to the process dedicated to spatial retrieval [73].

PSQL

PSQL is a query language for pictorial databases, conceived as an SQL extension tailored to raster image processing. The PSQL can use pictorial domains representing geographical points or regions. Operators can be used on these pictorial domains. The difference with a traditional SQL is the specification of a **one-at** clause as we can see in Figure 3.5.

PSQL defines relation tables linking a picture with a relation. Each spatial relation is extended by an attribute "loc". The reference of the spatial relation is done in the **on** clause, and represents a pointer to a picture or a list of pictures [86]. The reference in the **at** clause may be followed by spatial operator such as *covering*, *covered-by*, *overlapping* or *disjoint*.

```
select <attribute-target-list>
from <relation-list>
on <picture-list>
at <area-specification>
where <qualification>
```

Figure 3.5: PSQL query form

KGIS

The KGIS is a SQL-based query language for the Geographic Information System [73], able to handle spatial and non-spatial attributes. The spatial informations are stored in one layer. Therefore there exist two views in the KGIS: one gets the spatial and non-spatial information and another is used to represent geographical features in a separate relational table.

The geographical features have additional spatial attributes such as *size*, *shape* and *location*, and spatial relationships to other geographic features, such as *proximity*, *adjacency* and *direction*.

In order to extend the queries for spatial data, the language uses the attribute MAP. Spatial attributes that are currently supported include also AREA, PERIMETER and LENGTH. It makes the language able to treat spatial and non-spatial attributes in a single context. They appear to the user to be stored explicitly in the database. An example of a query in the KGIS looks as follows [42]:

```
SELECT ID, MAP, PERIMETER, OWNERNAME FROM PARCELS WHERE AREA > 10;
```

The location data for geographical features are maintained in a single database referred to as a GEOVIEW.

TIGRIS GIS

The query language for TIGRIS GIS [39] uses an object-oriented approach based on SQL principles. The spatial extensions are defined by Boolean topological operators (*intersect*, *adjacent*, *boundaries* or *union*) or by spatial attributes (*area* or *centroid*).

Spatial SQL

The spatial SQL [29] consists of two components, a query language to describe what information to retrieve and a presentation language to specify how to display query results. The spatial SQL is not changing the basics of the SQL langage for non-spatial data. It creates spatial domains going from 0 to 3 dimensions.

Therefore the characteristics of the spatial SQL is represented by some SQL language extensions for spatial data and the introduction of GPL (Graphical Presentation Language), used to examine and view the output of the queries. For spatial information, we use Spatial SQL commands and give GPL instructions to examine and view the results. More particularly, the GPL is used to give the parameters of the display environment. An example of a spatial SQL query and GPL looks like:

```
SET LEGEND black
  COLOR dashed
  PATTERN
FOR SELECT road.geometry
FROM road, town
WHERE town.name = "Orono" and
  road.name = "Grove Street" and
  road.geometry INSIDE town.geometry;
```

The first part of the query (before the clause FOR) contains the GPL information and the second part contains the spatial query. We can see that the spatial SQL uses spatial relations - in this case INSIDE - but all the common topological operators are authorized: *disjoint, meet, overlap, inside/contains, covers/coveredBy*, and *equal*.

3.2.4 Operation queries

In the following we will describe a number of queries for three-dimensional spatial data. Some of the most important classes of operation queries are: *point queries*, *range queries*, *nearest neighbour queries*, *distance scans*, *intersection queries*, *containment queries* and *spatial join queries*. For spatial recognition, it is interesting to add more particular queries. These new queries are extracting points that have the form of 3-dimensional shapes. For example, in traditional queries, the range query - formally defined as an operation that retrieves all records where some value is between an upper and lower bound (b_{low}, b_{high}) - extracts all the points in a given rectangle (for 2-dimensions) or in a given box (for 3-dimensions). More sophisticated range queries will allow to extract points corresponding to some regions defined by shapes. For these regions, two kind of

queries must be implemented: a query to extract the interior of a shape and a query to extract the ϵ-band around the surface of the shape. To optimize the performance of the query engine, we created queries which correspond to basic 3-dimensional shapes.

For our research we proposed to extend the common classes of spatial queries with the following operations:

- rotated range query: the same as the range query, but we may apply some rotations - the records are retrieved from a rotated box given α and β $(\alpha, \beta, b_{low}, b_{high})$.
- sphere query: the query is defined by a point $q \in \mathbb{R}^d$ and a $radius \in \mathbb{R}$ - the records are retrieved from a sphere $(radius, center(x, y, z))$.
- sized sphere query: the query is similar to the sphere query, but we retrieve the points in the ϵ-band of the hull of the sphere - the parameters are (ϵ, $radius$, $center(x, y, z)$).
- cylinder query: the query is able to retrieve all the point in a given cylinder defined by a center, a radius, a height and two angles α and β to rotate the cylinder - the parameters are $(\alpha, \beta, radius, height, center(x, y, z))$.
- sized cylinder query: only the points in the ϵ-band hull of the cylinder are retrieved, the parameters are $(\alpha, \beta, \epsilon, radius, height, center(x, y, z))$.
- torus query: we retrieve all the points inside a torus defined by a center, the radius to the middle of torus, the radius of the torus and a rotation - the parameters are (α, β, $radius$, $axis\_distance$, $width$, $center(x, y, z)$).
- sized torus query: we retrieve all the points in the ϵ-band hull of a torus defined by a center, the radius to the middle of torus, the radius of the torus and a rotation (α, β, ϵ, $radius$, $axis\_distance$, $width$, $center(x, y, z)$).

These queries may be implemented on the spatial indexes defined in 3.1. Even if the indexes are able to store k-dimensional points (point access methods) or more generally k-dimensional spatial objects (spatial access methods), for these queries the indexes are storing only 3-dimensional points (point cloud). To extract the points that are defined by queries, we need to compute the Boolean function answering if the query is touching a given space (see minimum bounding region in 3.1.2, or the subregions in 3.1.1), and the Boolean function answering if a point is in the query or not.

Graphically it is more simple to exemplify in a 2D space. In the Figures 3.6 we can see that the range query is defined by the space $Q1$ which cuts the boxes $R3$ and $R5$. Therefore for retrieving the points we extract these boxes and look more particularly for the points in these boxes.

In the context of an X-Tree index, the implementation of these functions depends on the type of the tree node: non-leaf nodes (does't contain points) and data-node (containing points). The following two algorithms, exemplifying the rotated range query, were designed to treat these two cases: Algorithm 1 for non-leaf nodes and Algorithm 2 for data-nodes.

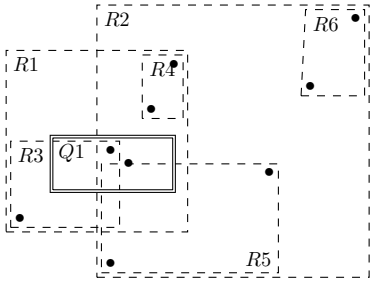

Figure 3.6: Query filtering - range query for minimum bounding regions

We added sized queries because we wanted to extract only the hull of an object. This operation is relatively simple to be implemented: we need to implement the Boolean function verifying if a point belongs to a given region, defined as a box with a hole in the middle. We validate if the point is contained in the outer boundary and not in the interior boundary. The rotated sized range query is similar to the rotated query, but we apply a change of the coordinates on the points to simplify the test (see Figure 3.7).

This is not an exhaustive set of possible queries. We could add more operation queries depending on the basic 3-dimensional shapes we are looking for.

Algorithm 1 X-Tree Node method for rotated range query

```
Xtree {the reference to the index (x-tree)}
currentNode ← node {root node to begin search and then child node}
mbr {set to minimum bounding region without rotation}
α, β {rotation angles in a spherical coordinate system}
rotatedmbr ← transform(mbr, alpha, beta) {set to minimum bounding region with rotation}
pointList ← ∅ {list of all points in the rotate range mbr}
for i = 0 → numberChildrenNodes(currentNode) do
  for j = 0 → numberOfPoints(mbr) do
    if insideDirBox(rotatedmbr[j], childNode[i]) then
      RotatedRangeQuery(childNode[i], pointList)
    end if
  end for
  plans ← rotatedFaces {get the list of all plans representing the range}
  for j = 0 → numberOfPlans(plans) do
    if isInsideOrTouching(currentNode, plans[i]) then
      RotatedRangeQuery(childNode[i], pointList)
    end if
  end for
end for
```

Algorithm 2 XTree Data Node method rotated range query

```
Xtree {the reference to the index (x-tree)}
currentNode ← dataNode {leaf node}
mbr {set to minimum bounding region without rotation}
α, β {rotation angles in a spherical coordinate system}
pointList {list of all points in the rotate range mbr}
for i = 0 → numberOfPointInNode(dataNode) do
  rotatedPoint ← invertRotation(α, β, point[i]) {we make a rotation of the point,
  which is simpler for testing}
  if inside_mbr(rotatedPoint) then
    pointList ← point[i]
  end if
end for
```

Nearest neighbour searching

The nearest neighbour search is a search that is based on point queries. The spatial database system should be able to retrieve the nearest neighbour point of the query point. The main problem of searching the nearest neighbour is its time consumption. This is the reason why in [5] an approximative nearest neighbour search is proposed instead of the exact nearest neighbour search. Consider the query point to be $q \in \mathbb{R}^d$ (in our case $d = 3$). If we set the maximum error to be $\epsilon > 0$, we consider p as the $(1+\epsilon)$-approximate-nearest-neighbour of q if $dist(p,q) \leq (1+\epsilon) \cdot dist(p^*,q)$ - in this formula p^* is the real nearest-neighbour. One way of computing this approximate nearest-neighbour is to use the *priority search* introduced in [6]. The idea of this method is:

- Given the query point q, we begin by locating the leaf cell containing the query point. According to tree definition, this is done in $O(log(n))$ time.
- Next, we enumerate the leaf cells by increasing order of distance from the query point. This method is called priority search.
- We keep the track of the closest point p. As soon as the distance of the current leaf to q exceeds $dist(q,p)/(1+\epsilon)$, we can stop the search and return the point p which is the approximate nearest point.

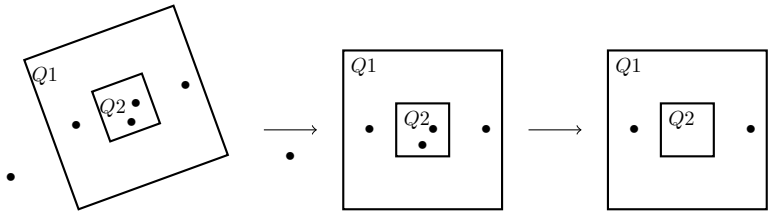

Figure 3.7: Query filtering - steps for rotated size query

The nearest neighbour search can be extended to the k-nearest neighbour search.

3.2.5 Query Estimation

We saw in section 3.2.4 that we can define different kinds of queries. These queries depend on the indexing structure and on the number of points in the selection. In the following subsection we will present methods to compute an estimation of the computational time of the queries. These estimation techniques can be categorized basically into three groups: *parametric*, *sampling* or *histogram* techniques.

Estimation for Range Query

The range query being a very common query, it would be interesting for a user or for any optimization process to know the estimated time of execution. An estimation technique using histograms has been done in [82]. By definition, a histogram is an array of intervals, denoted buckets. The set of buckets approximates the frequency distribution of the points. We keep the statistics for each subset, as values stored in each node of the tree.

Each bucket is defined by $[x, y]$, where x and y are respectively the lowest and the highest values of the interval. Consider a range query defined by two parameters (points) q_1 and q_2. We identify the bucket which overlap the set of points used by the query. The space region determined by the two points q_1, q_2 (in 3D it's the cube having as diagonal the segment $\overline{q_1 q_2}$) is called the range query predicate. A specific formula defines how to estimate the number of points in the bucket that satisfies the range query predicate.

Estimation for Spatial Joins

The spatial join query is a query that finds pairs of objects that satisfy a spatial predicate (for example intersection/overlap). The spatial joins is not useful for 3-dimensional point clouds, but only for objects. Usually, the extraction of the objects satisfying a spatial predicate is done in two steps: firstly, on retrieve the minimum bounding regions for the objects; secondly, on analyse the exact geometry of the pairs produced and on remove the false hits.

In order to perform an estimation of the execution time, we can use a sampling method, as described in [4]. One of the simplest method to generate the sample is the regular sampling (RS) method, which takes any k^{th} point in the set of all points N. It has been showed that, for this kind of estimation, it is better to create a new index based on the sample. The reason is that, for massive databases, even a small percentage of the data can result in a huge number of data items to be joined.

Moreover, given two data sets A and B and the samples α of A and β of B (α, β being percentages), an estimation of the selectivity is then given by $\frac{R}{\alpha \cdot \beta}$, where R is the selectivity of the join of the samples.

3.3 Conclusions

In this chapter, we saw several categories of indexing structures and query languages. No structure can claim to be the best indexing structure. Some indexes are better for storing and retrieving points whereas others are better for geometrical objects. In this chapter we saw the common queries and operation queries that are necessary to access the data in a spatial index. We saw some extension that can be done on the query system to improve the retrieval process for applications that need to extract 3-dimensional shapes.

Numquam ponenda est pluralitas sine necessitate.

William of Ockham

4

Specification of 3-dimensional objects

4.1 Introduction

As we explained in Chapter 1, one of the purpose of this work is to introduce a framework allowing a high-level description of a spatial scene containing objects. For such an object, the most important spatial aspects are *topology*, *orientation* and *distance* (the same concepts are acquired in this order by children). Other less important aspects can be considered, such as *size*, *morphology* and *spatial changes* (motion).

The description model containing these spatial information will be added in a semantical layer (also used in a further step for reasoning purposes). In this chapter, we will explain the necessary steps in order to create a semantical layer from a spatial 3D scene, where a scene is defined as a composition of basic or user-defined shapes. The steps of the layer creation process can be resumed as follows:

- Decomposing the scene into basic shapes.
- Adding topological relations between objects.

The two main objectives of this process are to represent the scene in a semantic meaningful way in order to use reasoning processes, and to acquire common sense knowledge about a scene. In respect to the acquisition of the knowledge about spatial relations, we can distinguish two kinds of predicates: quantitative and qualitative predicates. The quantitative predicates are describing distances and angles (e.g., "right of" or "parallel to"), whereas the qualitative ones are describing relative aspects (e.g., the size "is big" or the position "is near to"). We will see that these predicates depend on the context and pre-existing knowledge. The quantitative predicates over distances and angles are of primary importance.

With the number of objects increasing in a scene, the number of relations to verify is also increasing. If the number of objects defined in a scene is n and the number of possible relation is r, then the number of relations to verify is, in the worst case, $r\frac{n(n-1)}{2}$. Because the number of objects can increase quickly, it is important to focus only on the interesting relations. The complexity can also increase if some objects are moving, because the validity of some relations is changing over time. In [64] some predicates were defined to be true only over a certain time interval.

A semantic layer is decomposed into different parts, including

- the basic knowledge about spatial geometry;
- the user input;
- the high-level knowledge about the spatial scene.

These elements are the basic building blocs for the layer. After encoding the definition of all elements of a scene in the semantic layer, we get a high-level scene representation. On top of this semantic layer further knowledge can be inferred.

In the following, we will describe how we create a Geometrical Scene Description (GSD) based on an ontology. After the introduction (section 4.2) of the concept of GSD and the concept of ontology as the fundamental representation for the semantic layer, we will present (section 4.3) the development of a spatial ontology for representing a spatial scene. We will see in section 4.4 that we can use reasoning processes to add new relations and concepts to this ontology. The section 4.4.1 will present the RCC-8 calculus, which is the basic building bloc for reasoning on spatial concepts, whereas the section 4.4.2 will present the description logic viewpoint.

4.2 Geometrical Scene Description

A Geometrical Scene Description, conceived as a middle-level description of a spatial scene, it is a composition of objects and the spatial relations that exist between them. The available information after the extraction of GSD are the position and the rotation angle of spatial basic objects, and some relative location relations between objects. This information is used as basic input for inferring relations determined based on the context and pre-existing knowledge.

4.2.1 Ontology

An ontology represents a set of concepts and relationships between these concepts. To use an ontology we need an ontology language to formalise the representations. The OWL (Web Ontology Language) is an interesting language because it is endorsed by the World Wide Web Consortium (W3C) and is based on the meta-data model RDF (Resource Description Framework). For this reason the OWL is compatible with most software using semantic representations. The OWL is divided into three sub-languages, those characteristics depend on the balance between the expressiveness and decidability.

Even if a representation language is chosen, the knowledge is still depending on the kind of information we want to represent.

For our purpose we can define three categories of ontologies: *foundation ontologies*, *domain ontologies* and *user ontologies*. A Foundation ontology defines the basic concepts common to different domains. A Domain ontology is an ontology that can define a part of the world in a particular domain. For this research work we created a domain ontology defining the concepts related to 3 dimensional objects. The user ontology is the part of the ontology used to define instances created by the user. All the specifications were developed in OWL.

4.3 Spatial Ontology

In this section we will describe the spatial ontology used to represent a spatial scene. In [19] we defined an ontology for basic 3D shapes. Each ontology concept is linked with the spatial database system (SDS) through the set of standard queries defined in 3.2.4. It means that for any basic shape created in the ontology, the indexing system provides a way to extract the shape from the SDS. If the domain ontology is changed and new basic shapes are created, the indexing system has to be adapted to add the possibility to extract the new shapes. This can be done by adding a new query corresponding to the shape or by composing it using the ones already present in the system. The result of the queries consists of point sets representing the 3D objects searched for. The process leads to generation of groups of 3D objects which describe simple or complex ontology concepts.

The first implementation of the spatial ontology was sufficient to define pure mathematical relations between the shapes. The approach used to define a complete spatial scene is to decompose the ontology into two parts or group of concepts: the upper (reference) group and the lower (user) group. The advantage of such a distinction has been showed in [35]. More particularly, we were interested in the possibility to use different

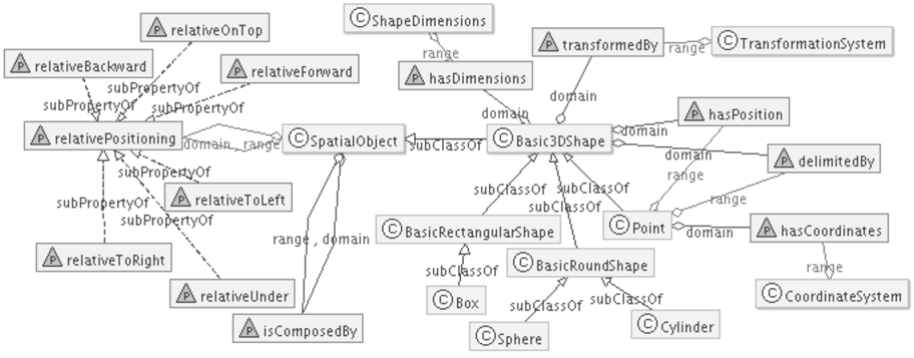

Figure 4.1: Ontology defining the basic shapes

composition of basic 3D shapes to generate more complex (composed) shapes. Therefore the lower ontology will provide the definitions used to create new concepts that are described by the user. In this way a user can develop his own objects and models.

The main part of the ontology, which is used to define the basic 3D shapes is showed in Figure 4.1. Because the ontology has to be adaptable to different environments, this ontology integrates different coordinate systems and rotation systems.

Furthermore, in the reference ontology presented in [19], we defined that an object can have a position relative to another. To implement this functionality we included the relations *relativeToLeft*, *relativeOnTop*, *relativeBackward* and their corresponding opposite relations. These relations have no specification about the distance between the objects. The relative positioning defines only the orientation of the relation. The ontology may by extended to add the notion of distance between two objects.

As we can see in Figure 4.1, we define all the pre-existing known objects starting from the basic concept *basic3DShape*. These objects are called primitives. These primitive objects can be categorized into two subsets: *BasicRectangularShape* and *BasicRoundShape*. All of these objects can be transformed by some operations (*transformedBy*) which are listed in the *Transformation System* ontology.

The reference ontology must allow to add elements representing compositions based either on primitive objects or on user-defined objects. The definition of a composition it based on the concept of spatial object. This is done using the relation *isComposedBy*. An instance of a spatial object which is not a subclass of *basic3DShape* is a new composition. An interesting characteristic of spatial object definitions is that an instance can use basic shapes, already defined compositions or a mix of basic shapes and compositions to make a new composition.

We also had to make sure that the ontological definitions for basic/composed shapes can use different coordinate systems for describing a position in space: actually, Cartesian, spherical and cylindrical systems are supported. Each coordinate system has properties that maps its specific characteristics (the coordinates and/or the angles necessary to

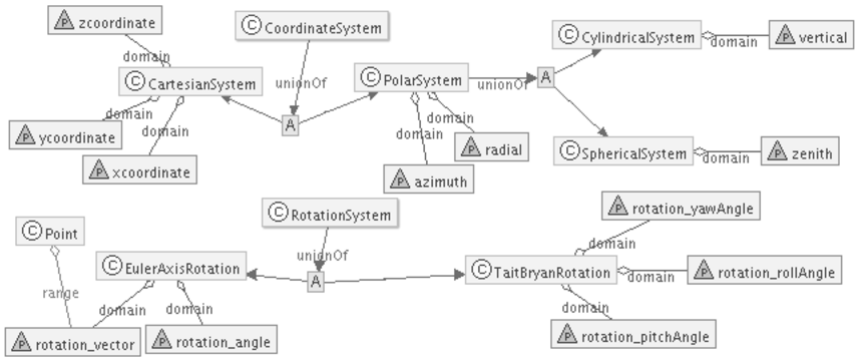

Figure 4.2: Spatial ontology: the coordinate systems

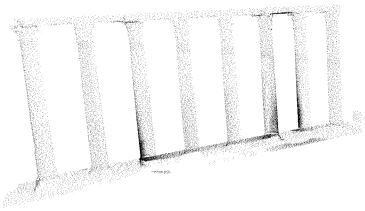

Figure 4.3: Entrance of the Pantheon represented by the point cloud

uniquely identify a point in the space) to one single reference system. Thus, in the example shown in Figure 4.2, the *CartesianSystem* has three metric-like properties (corresponding to the x-, y- and z- coordinates), while the *CylindricalSystem* has two metric-like properties and a degree-like property that correspond to the radial, vertical and azimuth values, respectively. The ontology can easily be extended with other systems.

Example: Entrance of the Pantheon

When we look at the entrance of the Pantheon of Rome, we can recognize immediately eight columns. In Figure 4.3 we can see the point cloud defining the entrance and in Figure 4.4 the ontology representing the entrance is given. The two figures are two representations of the same information. Figure 4.4 shows how a user can define the composition object *entrance* which is an instance of *spatialObject*. This new instance is composed of different shapes (here, eight columns instances of the basic shape *Cylinder*). Each column has a relation *onLeft* with the column on the left of itself, except for the last column to the left.

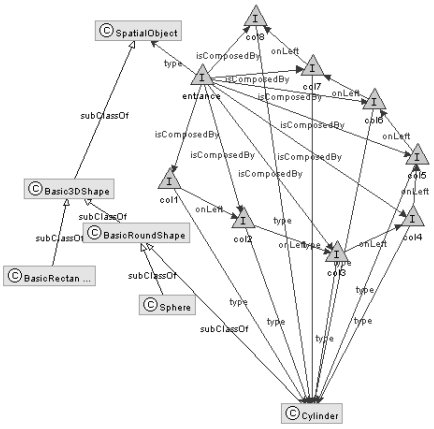

Figure 4.4: Ontology representing the entrance of the Pantheon

If there exists an interested on a more precise definition of an entrance column, we can modify and refine the instances *col1, col2, ..., col8.* E.g. a simple representation for a Corinthian column is a cylinder but we could define a Corinthian column as a composition (sockets plus cylinder).

4.3.1 Modification of the Spatial Ontology

When we look at a particular spatial scene, we note that the previously described reference ontology is not sufficient to describe all the visual information. To complete the ontology, we have to add some further conceptual elements. The upper-ontology we defined in [19] does not include size relations and the distance between objects. Thus, some constraints were added in order to define a scene in a more precise way. The relations related to size (*bigger* and *same size*) are sub-relations of the relation *property.* We defined also the concept *ConceptualSize* in order to express the size of a spatial object in respect to the whole scene. These modifications are showed in Figure 4.5. It has to be remarked that all these new relations have a *relative* signification (e.g., an object is *relatively small* regarding the whole scene, but to simplify the notation, the word "relative" was removed).

4.3.2 Contextual Relations

In a 3D scene we can compute the exact distance between two objects. Even if we could use this information and store it in an ontology, the exact information is sometimes less interesting than an ordered value. As it was shown in [20], qualitative information can be more meaningful than pure quantitative information. For example, if we say that Alaska has a surface of 1'518'800 km^2, it is less meaningful for some users than saying that Alaska alone is bigger than all the states of the East coast from Main to Florida. This is in particular true, if the objective is to capture the human point of view. Therefore, in order to capture this particular point of view, we defined *contextual relations.*

Consider two sets of points Ψ_i and Ψ_j and a number of relations between them. These relations can be split into two groups: the *non-contextual* and the *contextual* relations. The Table 4.1 presents an example of different kinds of this type relations.

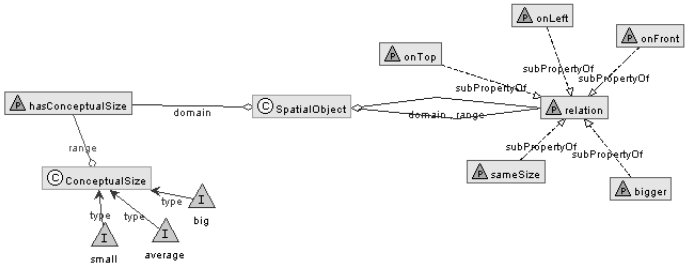

Figure 4.5: Ontology defining the relations between shapes

contextual relations	non-contextual relations
$big(\Psi_i)$	$isSphere(\Psi_i)$
$small(\Psi_i)$	$isCylinder(\Psi_i)$
$average(\Psi_i)$	$isBox(\Psi_i)$
$nearTo(\Psi_i, \Psi_j)$	$leftTo(\Psi_i, \Psi_j)$
$nearToFront(\Psi_i, \Psi_j)$	$frontTo(\Psi_i, \Psi_j)$
$nearToTop(\Psi_i, \Psi_j)$	$topTo(\Psi_i, \Psi_j)$

Table 4.1: Decomposition of the different kind of relations

The concepts of *big*, *average* and *small* are linguistic variables. Their meaning depends on the point of view of the observer. These are not precise definitions, they simply express the fact that, if an object is considered to be *small*, then an observer is not very much inclined to consider it *big*. It is the responsibility of the observer to fix the limits between the categories. A way to implement these relations is to consider them to be linked to some properties of the point cloud. We can consider the following non-exhaustive properties:

- the average distance between two nearest points τ;
- the size of the global scene: w (the width), l (the length), h (the height) of the smallest parallelepiped containing the whole set of points.

The set of linguistic variables could be incremented with other conceptual values. The notion of contextual size can be attached to any spatial object as we can see in Figure 4.5.

4.3.3 Completeness Property

The last improvement proposed here in regard to the basic ontology given in [19] is the addition of the completeness property. The importance of this notion derives from the fact that the purpose of recognition process is not only to find complete primitive forms but also composite forms and even parts of these forms. In Figure 4.6 we defined some mathematical properties of a *spatialObject*. If we want to describe the whole object, we express it by setting the property *completeness* to 1. If we are looking for an half-sphere, the *completeness* value would be set to 0.5, expressing that the surface we are describing is divided by two.

4.4 Reasoning

The idea behind creating an ontology layer is to make reasoning and deduction possible in order to infer new relations based on the current GSD. In the following sections we will see the basics of reasoning about spatial regions and some description logic used to implement rules inferring spatial knowledge. To create new rules about spatial data

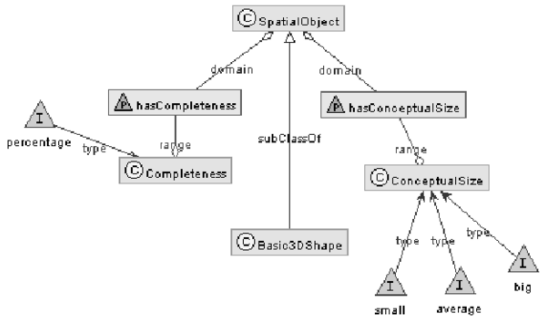

Figure 4.6: Adding the completeness property

we need to introduce the fundamental notions describing regions. These notions will be defined in the following section.

4.4.1 RCC-8 Calculus-based

The RCC-8 (for Region Connection Calculus 8) defines a language to express relations which may be defined between two regions. The specification for all spatial environments should be based on this basic set of relations. In RCC-8 we can distinguish eight possible relations:

- **DC** (disconnected) means that the two regions do not share any point at all.
- **EC** (externally connected) means that they only share borders.
- **EQ** (equal) means that the two regions are the same.
- **PO** (partially overlapping) means that the two regions share interior points.
- **TPP** (tangential proper part) means that one region is a subset of the other sharing some points on the borders.
- **TPPi** is the same as the TPP, but its true if we inverse the regions.
- **NTPP** (non-tangential proper part) is the same as TPP, but without sharing any bordering points.
- **NTPPi** is the same as the NTPP, but it's true if we inverse the regions.

This basic relations can be extended by basic constructs borrowed from description logic. For example, the logic described in [83] uses a dyadic relation $C(x, y)$. This relation has the meaning of *connected*, where x and y are two regions. To create this relation, we can use the RCC-8 as basis to describe it (e.g. the rule $\neg DC(x, y) \rightarrow C(x, y)$). Then some basic implications can easily be inferred, as "If two regions x and y are disconnected,

then the connected predicate for those two regions is false", or $DC(x, y) \rightarrow \neg C(x, y)$. And we can add e.g. two other axioms (i) $\forall x C(x, x)$ and (ii) $\forall xy[C(x, y) \rightarrow C(y, x)]$. Of course, there are other formalisms that could be used to described relations between sets. For example, RCC-5 is obtained by generalizing some relations (e.g. the TPP and NTPP are relations having each other as opposite relations, therefore one may consider them as one relation).

In our case, as the point cloud representation comes from a laser scanned building, the only information we have about the objects concerns their surfaces and not their interior. This means that the $EC(x, y)$ and $DC(x, y)$ predicates will be the only ones really useful for scanned 3D scene. Obviously, in a scanned point cloud, the elements can't overlap because they represent surface elements.

4.4.2 Description Logic

Description logic is a formalism for describing concepts, individuals and roles. Even if the same representation is used in OWL, the naming of the concepts is changing. Description logic (DL) is the general form of any language used to describe a graph of concepts. Because description logics is a wide set of languages, it defines a naming convention to describe the operators that are allowed for any language. A very common DL is ALC which is the minimal description logic including full negation and disjunction. An combination of RCC and ALC (ALC_{RCC}) only make sense if also the appropriate inverse relationships of the base relations of RCC are respected. For $ALCI_{RCC8}$, the set of role names is $N_R = (DC, EC, PO, EQ, TPP, TPPI, NTPP, NTPPI)$. More details about spatial description logic are given in [99].

4.5 Conclusions

As we could seen, the ontology representation provides a better mechanism for reasoning in comparison to a geometrical point cloud representation. We decided to use OWL-DL to describe the ontology layer because it is an XML based language accepted as a standard for semantics by the W3C, which have enough expressiveness and the capability of doing reasoning. To keep the ontology in OWL-DL, we have to respect the following limitations :

- A separation between classes, datatypes, datatype properties, object properties, annotation properties, ontology properties is strictly done.
- The four following property characteristics *inverse of*, *inverse functional*, *symmetric* and *transitive* are not used.
- Axioms are well-formed.
- Annotations are not allowed.

Once we have made the description of the spatial ontology, we have provided the basic concepts to create a spatial scene. Even thought, the domain ontology could be extended

by new basic 3D shapes (e.g., the torus or any iso-surface). These adding could be implemented if necessary, depending on the usage made of the ontology.

Do or do not. There is no try.
Master Yoda

5 Recognition of spatial objects

As described in chapter 2, there exist different representations of a spatial scene: unstructured points, meshes, mathematical objects or complete forms. A recognition process for these objects depends obviously on these representations. We will distinguish here two kinds of recognition approaches: the first one is using the basic mathematical object definitions and the second one uses a database of stored objects as input for the algorithm.

If the mathematical description of primitive objects is used, then a scene can be decomposed into logical parts to simplify the description of the scene. If a database is used to find registered objects, then the recognition algorithm is implemented as a search process. In order to implement this process, objects have to be described based on some specific features (e.g. their surface). In the following sections we will describe the mathematical tools and the specialized algorithms that can be used to recognize objects. A particular attention will be paid to the algorithms for the normal estimation. As the normals attached to points can be used as an estimation of the surface curvature, they may lead to a better description of the object's surface. Therefore, these basic algorithms using normals are often used when very sophisticated matching/recognition approaches are needed.

5.1 Mathematical foundation

In this section, we will present some basic concepts that are used for recognizing spatial objects. The concepts are: *local feature size*, *distance function* and *distance matrix*.

5.1.1 Local feature size

The local feature size is an important value in order to estimate the complexity of an object in the neighbourhood of a point of the cloud. For an object represented as a point cloud S, it has to be assured that the number of points (or the cardinality of S) is sufficiently large to represent all the features of the object. Therefore we need a measurement that tells us how complicated an object is in the neighbourhood of any point $x \in S$. In order to estimate this complexity we firstly compute the medial axis representing the middle skeleton of an object. The medial axis of a surface is the closure of the set of points in $\mathbb{R}^k$ that contains at least the two closest points on the surface. By definition, the local feature size at a point $x \in S$ is the distance from this point to the medial axis as defined above.

To reduce the complexity of the medial axis calculus, we may use a sample of the point cloud S instead of entire set S. The difference between the medial axis calculated using a sample of the object and the real medial axis based on the complete point cloud depends on the "quality" of the sample. If the difference between the two representations is very small (where the quantity "very small" should be defined), we consider that the sampling can be accepted.

5.1.2 Distance functions

The definition of *local feature size* given in the previous section is based on the concept of *distance*. Given a set X, a *distance function* (or *metric*) is a function d with non-negative real values defined on the Cartesian product $X \times X$ and satisfying the following four conditions:

1. $\forall x, y \in X, d(x, y) >= 0$ (any distance has to be positive or null)
2. $d(x, y) = 0$ if and only if $x = y$ (the distance between two objects is null only if the two objects are the same)
3. $\forall x, y \in X, d(x, y) = d(y, x)$ (the distance function has to be symmetric)
4. $\forall x, y, z \in X, d(x, y) + d(y, z) \geq d(x, z)$ (the triangular inequality has to be respected)

Once a distance between the elements of a set X is defined, we can use it to define a measurement of similarity between two sets X and Y. Suppose we are given a point cloud R of n points (the object representation), and a point cloud M of m points (the object model). The similarity measurement is a function of the smallest distance between the sets R and M, over all transformations from a transformation group G. If the object is static (see section 2.2.1) and the group G contains only rigid transformations (translations, rotations and reflection) then we get a rigid similarity measurement. If the object is dynamic (see section 2.2.2) and the group G contains also non-rigid transformations, (e.g. stretch, shrink, and twist transformations and the inverse of these operations), then we get a well-defined similarity measurement only if the distance function is invariant to deformations.

Euclidean distance

In a 3-dimensional space the Euclidean distance between two points p and q is defined as: $d(p,q) = \sqrt{|p_1 - q_1|^2 + |p_2 - q_2|^2 + |p_3 - q_3|^2}$. This distance can be generalized for N dimensions: $d(p,q) = \sqrt{\sum_{i=1}^{N} |p_i - q_i|^2}$. The Euclidean distance is a particular case ($k = 2$) of a Minkowsky distance: $d(p,q) = \left(\sum_{i=1}^{N} |p_i - q_i|^k\right)^{1/k}$. Another particular case, denoted Manhattan or city block distance, is obtained for $k = 1$.

The Euclidean distances is sufficient for rigid operations but if we want also to recognize objects that have been transformed in a non-rigid manner, we would have to adapt this distance formula. Another problem with the Euclidean distance is that it is very sensible to outliers.

Hausdorff distance

The Hausdorff distance is a dissimilarity measure between two point sets; it gives a measure of their difference. Given two sets, $A = \{a_1, ..., a_m\}$ and $B = \{b_1, ..., b_n\}$, the Hausdorff distance is defined as $H(A,B) = max(h(A,B), h(B,A))$ where $h(A,B) = \max_{a \in A} \min_{b \in B} d(a,b)$ with $d(a,b)$ the underlying distance (e.g., the Euclidian distance).

It is easy to see that this function is very sensitive to outliers. However this metric can be modified to be less sensitive to outliers or noise. As shown in [78], this can be achieved by dividing the set A into two new sets: A_k (the points close to B) and $A - A_k$ (the outliers, or noise). Even with this modification the Hausdorff has still the drawback to be sensitive to deformations. An alternative is the Gromov-Hausdorff metric.

Gromov-Hausdorff distance

The main advantage of using the Gromov-Hausdorff (GH) distance in shape matching, as shown in [63], is that it is a well suited metric for the recognition of isometric shapes. The distance is defined as follows : $GH(A,B) = inf_{f,g} H(f(A), g(B))$ where f and g are isometric (distance preserving) transformations. In other words, we use the functions f and g to apply transformations to the two sets A and B. If the distance computed is very small, then the two sets are representing the same object.

In [61] it has been shown that the implementation of such a distance has to be done using a heuristic. The Gromov-Hausdorff distance can only be approximated, the reason being that the operations defined by f and g can represent any isometric transformation.

5.1.3 Euclidean distance matrix

If we have a collection of N objects and we define a distance ρ_{ij} between the i^{th} and the j^{th} object then a distance matrix is given by:

$$\Delta = \begin{pmatrix} \rho_{11} & \rho_{21} & \cdot & \rho_{N1} \\ \rho_{12} & \rho_{22} & \cdot & \rho_{N2} \\ \cdot & \cdot & & \cdot \\ \rho_{1N} & \rho_{2N} & \cdot & \rho_{NN} \end{pmatrix}$$

This matrix can be used to implement comparisons in some of the algorithms described in this chapter. For a collection of points in R^k, we will get a special type of distance matrix, denoted as *interpoint distance matrix*. Although the distance used here is the Euclidean distance, any other metric could be used.

The interpoint distance matrix is used if we want to make comparisons between two sets of points (point clouds) having the same cardinality. We can conclude that there exists a rigid isometry (rotation, reflection, translation) from one point cloud to the other if the two matrixes are the same modulo some permutations.

5.2 Normal-Vector estimation

In this section we will describe the different techniques used to find the normal-vectors (normals) to a surface represented by a set of points. The construction of estimated normals is important because it gives the curvature of a cloud of points. A large number of algorithms that are directly using the point cloud need to know for each point its corresponding normal (rendering algorithms, shape reconstruction algorithms, etc.). Therefore we need some methods to compute the normal for each point before being able to recognize objects in the spatial scene. Five properties of the point cloud have to be considered in order to select the right approach:

1. *the noise* in the point cloud,
2. *the curvature* of the manifold represented by the point cloud,
3. *the density* of the points,
4. *the distribution* of the sample, and
5. *the neighbourhood size* used for the estimation.

We can see in Figure 5.1 some errors that can occur when estimating the normals.

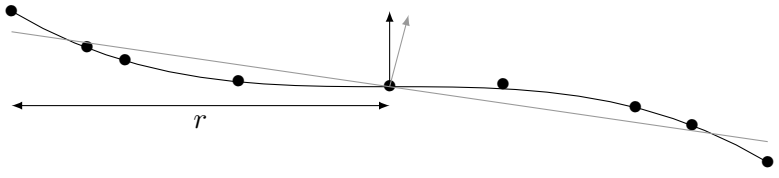

Figure 5.1: Curvature causes error in the estimated normal.

A particular problem occurs for surfaces that are very close to a plane (i.e., with a curvature close to zero). In this situation it can happen that the estimated orientation of the normal vector changes from one side of the plane to the opposite side. There exist several approaches to avoid his problem. The most simple solution is to take one of the normals ($\overrightarrow{v}$) as reference and then to check the neighbourhood: if the angle between the normal $\overrightarrow{v}$ and the next neighbour $\overrightarrow{v_i}$ (angle defined as $arccos(\overrightarrow{v} \cdot \overrightarrow{v_i})$) is greater than $\pi/2$ then we take the opposite orientation for the neighbour normal.

Estimation using least squares methods

In ideal conditions, if the cloud point has a high density and a uniform sampling was applied, then the estimation errors of a normal-vector could be considered to be very small. In this particular case we could use the k-least square method, where the parameter k (neighbourhood size) must be fixed by a supervised or unsupervised procedure. It is crucial that k is fixed in an appropriate way: too small k values could produce large errors for noisy data (which is usually the case for data coming from 3D scanners).

Briefly, the k-least square method for estimating the normal in a point p consists to find firstly the k nearest neighbors of p and, secondly, to compute the total least square plane fitting those points. The normal vector to the fitting plane is the estimate of the (undirected) normal at p.

In our spatial database system, the least square method is implemented using the k-neighbours query on each point in the space. In [79] it has been shown how to estimate local surface properties from the underlying point cloud using the covariance matrix. For a point p with k nearest neighbours $\{p_i\}, i = 1..k$, the 3×3 covariance matrix is defined as follows:

$$C = \begin{bmatrix} p_1 - \bar{p} \\ \vdots \\ p_k - \bar{p} \end{bmatrix}^T \cdot \begin{bmatrix} p_1 - \bar{p} \\ \vdots \\ p_k - \bar{p} \end{bmatrix}$$

In the expression of C, $\bar{p}$ is the centroid of the neighbours p_i of p. The covariance matrix is symmetric and positive semi-definite, which implies that it has three eigenvalues λ_1, λ_2 and λ_3, all of them being real-values. The minimum eigenvalue is associated with an eigenvector $\overrightarrow{v}$ which is the normal of the local surface. With this approach the problem that the eigenvector $\overrightarrow{v}$ could have a wrong orientation is still present.

Moving least squared (MLS) surfaces

The MLS surface approach is today one of the most widely used approaches for calculating normals. It smooths the surface and is therefore a good approach for noisy spatial scenes. At the basis, it is a point-based surface representation but it can also be used to compute the normals. Given a point set P, the MLS surface $S_{MLS}(P)$ is defined implicitly by a projection operator or a scalar field.

There exist two categories of MLS, *projection MLS surfaces* and *implicit MLS surfaces*. The method of MLS is based on the WLS (weighted least square), which implies

the minimization of an expression depending on a defined weighting function (assumed to be smooth, positive, and monotonically decreasing). A common function implementation for the WLS is the Gaussian function $\theta(d) = e^{-\frac{d^2}{h^2}}$, where h is a parameter that can be used to smooth out small deviations in the data. Another common weighted function used for WLS is the Wendland function $\theta(d) = (1 - d/h)^4(4d/h + 1)$. In the literature, a lot of different weighted functions have been proposed (see [18]).

The projection MLS surface (PMLS) is defined in [3] as a set of stationary points of a projection operator or as the local minimum of an energy function along the directions given by a vector field. The implicit MLS surfaces (IMLS) is specified by the moving least-squares solution to a set of constraints that force the implicit function to assume given values at the sample points and also force its upward gradient to match the assigned normals at the sample points.

Adaptive MLS surfaces

In order to remove the noise from a point cloud, a well known approach is to smooth the surface and to make it similar to the original surface. One of the methods implementing this approach is the Adaptive MLS, which is based on the implicit MLS surfaces. The implicit function around a point considers a global scale factor h for all the neighbour points [26], which may induce instabilities in the optimization process for non-uniformly sampled point clouds. An idea to overcome this problem is to use a weighting function giving more importance to the points near the original point. More generally, the surface should adapt to the feature size and the sampling of the object. For example, for a point p with the neighbourhood $\{p_i\}$, $0 \leq i \leq N$, the weights may be expressed as Gaussian functions parametrized by the distance between the point p and p_i.

There are also other solutions to compute the surface, derived from the MLS surface approach. An example is the VMLS (Variation of the PMLS), which uses an energy function $\varepsilon : R^3 \times R^3$, $\varepsilon(y, n(x)) = \frac{1}{2}\sum_{p \in P}[(y - p)^T n(x)^2]\theta_p(x)$, where $n : R^3 \times R^3$ is a given vector field and θ is a weighting function. The energy function is simpler to implement and more stable then the approach used in PMLS.

For all variants of weighted least squares methods (MLS, AMLS, VMLS), the normal could be estimated as the direction of smallest weighted covariance for the covariance matrix $W(p)$ with

$$w_{ij} = \sum_{k=1}^{N} \left(e_i^T(p - p_k)\right)\left(e_j^T(p - p_k)\right)\theta(||p - p_k||),$$

where $e_i, i = 1..3$ is a basis of R^3.

Estimation Using Voronoi Based Methods

The Voronoi diagram represents the set of all cells V_p (a Voronoi cell associated to the point $p \in P$), where any cell V_p is the set of all points for which the distance to any arbitrary point $q \neq p$ is greater than the distance to p. For two points p and q, the

common boundary of the Voronoi cells V_p and V_q belongs to the bisector of the segment $\overline{pq}$. This implies that for 3-dimensional data, the Voronoi cells are 3-dimensional convex polytopes.

A method showing how the Voronoi diagram can be used to estimate normals is presented in [2]. The *power crust algorithm* uses the Voronoi poles (two poles by cell V_p, a positive pole p^+ and a negative pole p^-) which are the farthest vertices from the point p. The balls that touch the point p and contain the poles are called *polar balls*. The union of those polar balls approximates the surface. This algorithm is not able to produced good estimation in the presence of noise. However, in [25] a modified version of the *power crust algorithm* is described which can overcome this problem. The idea is to consider either p^- or p^+ and the sample point p. The vectors $\overrightarrow{pp^-}$ or $\overrightarrow{pp^+}$ are good estimators of the normal of the surface. Some of the normal estimators might have a wrong orientation, so we have to apply a method to compute a consistent global orientation of the surface. The method described at the beginning of this section is appropriate, but it is also possible to apply more sophisticated approaches.

5.3 Matching algorithms

In the following sections we will present different recognition algorithms. We distinguish between those using primitive objects and those using stored models. The matching algorithms use different representations of the objects and different ways to measure the similarity between the objects. For example, one possibility consists of measuring the amount of modifications needed to fit a model to the raw data (particularly useful in the case of curve fitting). Another possibility is to measure the modifications that have to be applied to a graph in order to match the real data (in the case where the point cloud is represented as a graph).

5.3.1 Curve extraction

Let's consider two curves c_1 and c_2. In order to calculate the similarity between the two representations, the matching algorithm is based on finding the minimum-cost deformation applied to curve c_1 to obtain the curve c_2. A curve can be stretched or banded, therefore the deformation operator is defined by a sequence of these operations. The efficiency of this method has been proved in hand-written character recognition. Even if the curve based model is a natural choice for recognition, it does not describe the interior of a 3D form and can therefore miss some important features of an object.

5.3.2 Iterative closest point methods

The iterative closest point method (ICP) is a heuristics developed to match two point clouds [102]. This algorithm can be used for different inputs: *point sets*, *line sets*, *implicit curves/surfaces*, *parametric curves/surfaces*, *triangle sets*. Essentially, the algorithm has to select some set of points in one or both point clouds, to associate these points to

samples in the other point cloud, to weight the corresponding pair and to minimize a given definition of error by iteration.

The association step uses the nearest neighbour criteria. If we consider the point cloud $C = c_1, \ldots, c_n$, the distance between a point p and C is defined as: $d(p, C) = \min_{i=1..n} d(p, c_i)$. Here the basic idea consists of matching iteratively one point to the closest point in the set of points. In general the point cloud is compared to either a parametric entity or an implicit entity (see 2.2.1). In the case of parametric entities $\vec{r}(u, v)$, the distance from a point to the parametric entity E is given by: $d(p, E) = \min_{\vec{r}(u,v) \in E} d(p, \vec{r}(u, v))$. Once we have described a parametric entity, we can define the distance from a point to a set of parametric entities $F = \{E_1, E_2, \ldots, E_m\}$ as $d(p, F) = \min_{i=1..m} d(p, E_i)$. For an implicit entity defined as the set of zeros for a function $\vec{g}(r)$, the distance from a given point p to an implicit entity I is given by $d(p, I) = \min_{\vec{g}(r)=0} d(p, r) = min_{\vec{g}(r)=0} ||r - p||$.

The classical error metric used by ICP is the sum of squared distances between corresponding points. For such a metric there exists closed-form solutions for determining the rigid-body transformation that minimizes the error [87]. Another used metric is the sum of squared distances from each source point to the plane containing the destination point and oriented perpendicular to the destination normal [17].

5.3.3 Graph matching methods

The graph matching methods are using a simplified representation (a graph) of the point cloud. The computed graphs are compared with graphs stored in a database. The main issue consists of extracting a meaningful graph from a sample S.

Shock graphs extraction

In order to apply graph matching, a refined version of the skeleton graph (denoted as *shock graph*, see section 2.3.2) can be used. The changes between two shock graphs occur where the topology is changing. It has been shown that the shock graphs are well working for articulated objects, deformable objects and parts of objects, in presence of shadow and highlights. Shock graphs are defining objects, and therefore, from some point of view, such a graph is similar to an ontology. However, this approach only gives the topological relations between parts of objects, whereas the ontological approach typically adds richer relations.

Graph matching algorithms

Consider we have two shock graphs that we want to match: one is extracted from the scene (G_1) and the other from a database of objects (G_2). We define the distance between G_1 and G_2 to be the minimum deformation path (sequence of transitions) needed to go from one graph to the other. The shock transitions are composed of the following four types of edit operations: *slice operation*, *contract operation*, *merge operation* and *deform*

operation. We assign a cost to each of these transformations. More details are given in [96].

Reeb-graphs approach

Another approach is to use Reeb-graphs. To create the Reeb-graph of a cloud point, we use a function $\mu : M \to \mathbb{R}$ that returns the classes of the points. The function $\mu(\cdot)$ is computed for each point $p \in S$, which allows the creation of subsets of points (more precisely, partitions of the cloud point). All these partitions are represented as nodes; the connections between them are created by computing the adjacency of these partitions. Because an object can be detailed in a very large graph, we may create a multi-resolution Reeb-graph having different level of details.

5.4 Detection

After having introduced the different representations of spatial objects, we can now used these ideas in the process of scene analysis. In this section we will show how the detection of objects may be realized. We are focusing on two kinds of detections. The first one is based on registered objects in database. In this case we need to use a simplified representation of the objects, e.g. we are seeking for a signature in order to have a fast matching algorithm (an example of such signature is the skeleton graph). The second case is based on geometrical primitives. The detection algorithm uses primitive basic shapes, such as spheres, boxes, cylinders, planes, torus etc. in order to detect the objects. For this approach there exists two kinds of algorithms that can be used to detect these primitive shapes: the *Hough transform* algorithm and the *Ransac* algorithm. These algorithms were firstly used for 2D images, where the concept of connectivity is well defined as the way in which the pixels relate to their neighbors (4 or 6 or 8 neighbours of a pixel). Therefore there is a fundamental difference in the applications of these algorithms for point clouds, due to the lack of explicit connectivity information.

5.4.1 Hough transform algorithms

The Hough transform algorithm was originally used for detecting lines in a 2-dimensional pictures. To apply the Hough transformation, we use the fact that a line in the image (or natural) space, defined by the equation $Y = a \cdot X + b$ with the parameters a and b, can be represented by a point (a, b) in a parameter space. To avoid the problem of vertical lines in 2D, it is better to use another pair of parameters, ρ and θ (the polar coordinates) which gives as consequence a line equation of the form $cos(\theta) \cdot X + sin(\theta) \cdot Y = \rho$.

Based on this idea, two processes can be defined to detect lines in a picture formed by pixels, namely *exhaustive enumeration* and *diverging map*. While the exhaustive enumeration will have to pass trough all the pixels of an image, the diverging map has to do a mapping of the points into the parameter space.

The randomized Hough transform (RHT) is an extension of this method and has been developed to speed up the computation [100]. It uses a combinations of randomized

sampling and converging mapping to improve the speed of search. Because of the randomized sampling, RHT is considered as a heuristic. The randomized sampling can be done in several ways, the simplest being to taking randomly n pixels (or points in 3D).

The Hough transform principle has been extended for 3-dimensional scenes. A point in the 3D space (x, y, z) can be considered as the parameters of a plan (a, b, c). The corresponding normal form is given by: $cos(\theta) \cdot cos(\gamma) \cdot X + sin(\theta) \cdot cos(\gamma) \cdot Y + sin(\gamma) \cdot Z = \rho$. This approach is particularly well suited for detecting surfaces defined by planes such as roofs or walls (see [77]).

5.4.2 RANSAC algorithms

The basic principle of the RANSAC algorithm, introduced by Fischler and Bolles [34], is also applicable to 3-dimensional shape detection, as it was developed in [90]. In this article, the authors defined an algorithm based on random sampling. The main motivation for using random sampling comes from the fact that the database containing the digitized objects may be very large. The approach is essentially the same as in the case of the RHT algorithm.

The algorithm decomposes the spatial scene into basic 3-dimensional shapes. The main principle is to extract basic shapes by randomly drawing minimal sets from the point cloud. This kind of algorithm is efficient for some primitive shapes such as planes, spheres, cylinders, cones and toruses. The number of points and normals needed for creating a candidate objects in minimal sets depends on the complexity of the shape. This algorithm is particularly efficient for primitive shapes, which is sufficient for our purposes. Furthermore the algorithm is simple to implement and can easily by adapted to different types of problems. The RANSAC approach has also the advantage to be robust against noise. It has been shown efficient even in the case of point clouds up to 50% of outliers.

The basic idea of the algorithm can be described as follows. Consider as input a point cloud $P = (p_1, ..., p_m)$ with the associated normals $N = (n_1, ..., n_m)$. In a first phase, minimal sets are randomly drawn. A minimal set is defined as the minimal number of points required to uniquely define a given type of geometric primitive. After a fixed number of trials, the candidate shape which approximates the biggest number of points in P (called the best candidate) is extracted and the algorithm continues on the remaining data. The best candidate is only accepted if, given the size $|m|$ (the number of points) of the candidate and the number of drawn candidates $|C|$, the probability $P(|m|, |C|)$ that no better candidate was over-looked during sampling is high enough. If a candidate is accepted, the corresponding points P_m are removed from P and the candidates C_m generated with points in P_m are deleted from C. The algorithm terminates as soon as $P(\tau, |C|)$ has reached a user defined minimal shape size τ. Therefore, RANSAC is a non-deterministic algorithm producing reasonable results only with a certain probability (depending on the number of iterations).

This algorithm exhibits some interesting properties, as its *simplicity* (which makes it easily extendible), its *generality* (which makes it adaptable to specific purposes), and its particular *resistance against outliers*. For these reasons we decided to use the basic

idea of the RANSAC algorithm to develop two new algorithms using spatial ontologies. These algorithms produce as output a set of primitive shapes $\Psi = \{\psi_1, .., \psi_k\}$ with corresponding disjoint sets of points $P_{\psi_1}, ..P_{\psi_k}$. These k objects represent a smooth approximation of the scene.

Sampling strategy

In order to improve the performance of the algorithm, the sampling strategy must be adapted and defined such to increase the probability of drawing minimal sets belonging to the same shape. The main idea is based on the assumption that two points that are close together have a higher probability to belong to the same shape than two points that are far apart. The spatial proximity between points may be efficiently encoded in an octree [1]. The sampling strategy starts by selecting the first point p_1 of the minimal set uniformly among all points of P. A cell C containing p_1 is randomly selected from any level of the octree. The following $k-1$ points of the minimal set are randomly drawing from C. If the cell was well-chosen, the points contained in it will mostly belong to a common shape ψ. All the details about how to estimate the probability to find a shape ψ of size n ($P_{local}(n) = P(p_1 \in \psi) \cdot P(p_1, .., p_k \in \psi | p_2, .., p_k \in C) \approx \frac{n}{md2^{k-1}}$, with d the octree level of C) or to adjust the probability of selecting a good cell are given in the article [90].

Score evaluation

One of the key components of the RANSAC algorithm is an evaluation function σ_P that measures the quality of a founded shape ψ. To compute this evaluation, the following elements are used:

- the number of points that fall within the ϵ-band around the shape.
- the number of points inside the band for which the normals are not deviating more than a given angle α from the expected normal.
- the points constituting the largest connected component of the shape.

These three aspects act like a refinement processes for the set of points P_ψ. Formally, $\sigma_P(\psi) = |P_\psi|$ (where $|\cdot|$ states for cardinality). The set P_ψ is defined by a three-steps procedure.

1. $\overline{P}_\psi = \{p | p \in P \wedge d(\psi, p) < \epsilon\}$, where $d(\psi, p)$ is the Euclidean distance between the point p and the shape ψ.
2. $\hat{P}_\psi = \{p | p \in \overline{P}_\psi \wedge \arccos(|n(p) \cdot n(\psi, p)|) < \alpha\}$, where $n(p)$ is the normal of the point p and $n(\psi, p)$ is the normal of ψ in the projection of p on ψ.

[1] An octree is a tree data structure in which each internal node has exactly eight children.

3. $P_\psi = maxcomponent(\psi, \hat{P}_\psi)$, where $maxcomponent(\cdot, \cdot)$ extracts the group of points in $\hat{P}_\psi$ whose projections onto ψ belong to the largest connected component on ψ.

Extract connected components

Once an object is detected, all the points corresponding to the definition of this object are identified. In the 2-dimensional case it is straightforward to identify the connected points: it is sufficient to incrementally connect parts in the 4- or 8-neighbourhood of a pixel.

To reduce the 3-dimensional case to a 2-dimensional case it is sufficient to project 3D points on a bitmap located in the parameter space of the shape (for primitive shapes, the parameter space is bi-dimensional). The points which are projected are those falling in the ϵ-band of the shape and for which the normals do not deviate by more than an angle α from the expected normal.

5.4.3 Global scene extraction

In the previous sections, we described different algorithms for finding basic shapes in a spatial scene. In chapter 4, we have defined how to represent the knowledge that can be extracted from a spatial scene. The purpose of this section is to describe an algorithm we developed for global scene extraction. It is designed to provide a concise description of a spatial scene, using ontologies as representation tool. The presented algorithm (see pseudo-code in Algorithm 3) is an extension of the basic RANSAC algorithm. Even if the input is similar to the original RANSAC algorithm (a point cloud), the output is different. It is a complete description of the scene as basic 3D shapes with their relations (relative distance and size) in the space.

The first part of the global scene extraction ontology-based algorithm proceeds in the same way as the basic RANSAC algorithms. As for the RANSAC algorithm, this part terminates as soon as $P(\tau, |C|)$ is large enough, for a user defined minimal shape size τ. Then the relations between the objects are established using the brute force search. The complexity of this approach is $O(k \cdot n^2)$, where k is the number of relations tested and n the number of objects. As the number of objects in most of the cases we considered is relatively small compared to to number of points in the cloud, we did not investigate further heuristics in order to reduce the complexity.

If we want to extract only the interesting relations, then the scene can be filtered using the Euclidean distance matrix as defined in 5.1.3. This implies that the extraction of the position/size relations could be done only after having produced the description of the scene as basic shapes. From a practical point of view this means that the construction of the whole scene is done in two steps:

1. recognition of the shapes and their absolute positions;

2. parsing the ontology to discover size and position relations, directly depending on the attributes found for the shapes.

Algorithm 3 Ontology extended RANSAC Algorithm

$BPos \leftarrow \emptyset$ {the sets of all binary relations of type "position" discovered}
$BSize \leftarrow \emptyset$ {the sets of all binary relations of type "size" discovered}
$Unary \leftarrow \emptyset$ {the sets of all unary relations discovered}
$\Gamma \leftarrow \emptyset$ {extracted shapes}
$C \leftarrow \emptyset$ {shape candidates}
repeat
  $C \leftarrow C \cup newCandidate()$ {using minimal set}
  $m \leftarrow bestCandidate(C)$ {compute the score}
  if $p(|m|, |C|) > p_t$ **then**
    $P \leftarrow P \backslash P_m$ {remove points}
    $\Gamma \leftarrow \Gamma \cup m$
    $C \leftarrow C \backslash C_m$ {remove invalid candidates}
  end if
until $p(\tau, |C|) > p_t$
for $i = 1 \rightarrow number(\Gamma)$ **do**
  if $testUnaries(\Gamma_i)$ **then**
    $Unary \leftarrow Unary \cup unary(\Gamma_i)$
  end if
  for $j = i \rightarrow number(\Gamma)$ **do**
    if $testPosition(\Gamma_i, \Gamma_j)$ **then**
      $BPos \leftarrow BPos \cup rel(\Gamma_i, \Gamma_j)$
    end if
    if $testSize(\Gamma_i, \Gamma_j)$ **then**
      $BSize \leftarrow BSize \cup rel(\Gamma_i, \Gamma_j)$
    end if
  end for{We test what binary relations are satisfied}
end for

The list Γ of discovered shapes is first parsed to "encode" each object into an instance of the corresponding concept from the spatial ontology. The Boolean functions $testUnaries, testPosition$ and $testSize$ receives as input a list of spatial relations of a specific type defined in the ontology (e.g. $isBig$, $isSmall$ for unary relations, or $isOnLeft$, $isOnTop$, etc.. for position relations) and one/two instances of shapes, and return $True$ for each particular relation satisfied by these instances.

Complexity analysis for global scene extraction algorithm. As the Algorithm 3 is structured in two sequential main blocks - the first one (MB_1) constructing the set Γ of extracted shapes, the second one (MB_2) determining the relations between extracted shapes - the complexity of the full global scene extraction algorithm is the sum of the complexities for the two blocks, i.e. $O(\text{Alg. } 3) = O(MB_1) + O(MB_2)$.

We already noted that the first block uses the principle of the RANSAC algorithm.

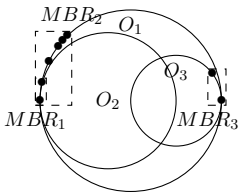

Figure 5.2: MBR principle for overlapping shapes

Therefore, according to the complexity analysis developed in [90], the first term of the sum O(Alg. 3) is $O(|\Gamma|\frac{C}{P_n})$, where $|\Gamma|$ is the number of extracted shapes, C is the cost to evaluate the score of a candidate and P_n is the probability to detect (in a single pass) inside a point cloud of size N a shape γ consisting of n points.

The complexity of the second main block is dominated by the cost of the tests checking if a given object satisfies possible unary relations and of the tests checking, for a given pair of object, the satisfiability of binary relations of type *size* and *position*. These costs may be considered as bounded by a constant C_r, because the tests are using the mathematical representation of an object, and not its cloud representation. Therefore, $O(MB_2) = O(C_r|\Gamma|^2)$, as the number of possible pairs is $|\Gamma| * (|\Gamma| - 1)/2$.

The complexity of the global scene extraction algorithm is thereby $O(|\Gamma|\frac{C}{P_n})+O(C_r|\Gamma|^2)$. Depending of the number of objects contained in the scene and of the probability P_n, the complexity of the algorithm is either dominated by the first term $O(\frac{|\Gamma|}{P_n})$ or by the second term $O(|\Gamma|^2)$.

Overlap filtering

In the original RANSAC algorithm, after an object is found, the points corresponding to this object are removed from the index. Another way to extract all the best objects for an ontological description of a scene would be to use the principle of minimum bounding regions. This principle is often used for indexing purpose. The approach is based on the following idea: instead of removing the points from the scene, get all the objects matching in the scene and sort them by their score. An object is added to the final list Γ if and only if the MBR (the minimum bounding region) of the corresponding points does not overlap any region of an object added earlier in the process.

Figure 5.2 illustrates this ideas using three circles with corresponding minimum bounding regions (MBRs). The object with MBR_1 has a higher score than the one with the MBR_2, but the second object can not be kept because the two MBRs are overlapping. As the MBR_3 is not overlapping any object with a higher score, the circles O_1 and O_3 are considered to be the best representation for this scene.

Of course this process will not cover all the points with a basic shape. This fact is not a big issue, because the idea of the algorithm is not to have a better rendering, but to simplify the process and to give a concise description of what we can see in a scene.

5.4.4 Ontology-driven object detection

In order to improve the detection of objects we are now considering an approach combining a known recognition system with spatial ontology descriptions. In this section we will define a filtering algorithm based on this approach, and illustrate how it can be used to find a pre-defined complex spatial object.

There are some interesting aspects of a spatial ontology that can be used in a filtering algorithm. It allows a user to define complex objects and to name these concepts. These objects can themselves again be used to define more complex objects. In general this helps to describe a scene in a more human-centric view and to integrate this view into the different machine driven approaches.

From now on we are considering three kinds of representations of a spatial scene: the mathematical form, the cloud point form and the semantic form. Figure 5.3 presents the three different points of view of a spatial scene.

The ontology-driven object detection algorithm, described in Algorithm 4, uses an ontology as input for the search process in the spatial space. This modified RANSAC algorithm is starting from a point cloud representation to a semantic form by passing through a mathematical form. For the purpose of filtering objects, we can use a description of a scene, seen as a semantic representation of the scene. This description will be used to guide the search in the scene under its point cloud representation. To filter the scene and to extract the composition defined in the semantic specification, we need to identify the three aspects of the composition we are looking at: the shapes, their unary relations, and their binary relations.

The core of the Algorithm 4 is represented by the recursive function *search*. The list of input parameters includes:

1. the list A of $basic3Dshapes$ α_i, i.e. the models of the shapes searched in the scene;

2. the set $BRel$ of binary relations satisfied by pairs of models from A; these binary relations are of type "position" ($isOnLeft$, $isOnFront$, $isOnTop$,..);

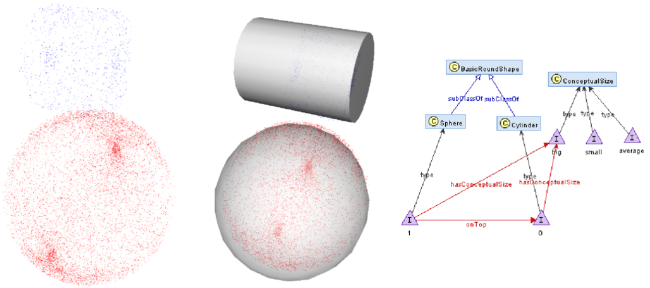

Figure 5.3: Point cloud, Shape and Ontology view point of the same composition

Algorithm 4 Ontology-driven Object Detection Algorithm

$A \leftarrow \{\alpha_1, \alpha_2, ..., \alpha_n\}$; {basic3Dshapes (ontological concept)}
$BRel \leftarrow \{\beta_1(\alpha_i, \alpha_j), \beta_2(\alpha_k, \alpha_l), ..., \beta_m(...)\}$ {binary relations (ontological concept)}
$URel \leftarrow \{\mu_1(\alpha_i), \mu_2(\alpha_j), ..., \mu_s(...)\}$ {unary relations (ontological concept)}
$\Gamma \leftarrow \{\}$ {the list of the current shapes found}
$space \leftarrow x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}$
FUNCTION $search(A, BRel, URel, \Gamma, space)$
if $A = \emptyset$ **then**
  return *true*
else
  $c \leftarrow Head(A)$; $A = A \setminus \{c\}$;
  $\Gamma_c \leftarrow searchObjectIn(c, space)$ {Γ_c - the list of potential objects of type c}
  $found = false$
  while $\Gamma_c \neq \emptyset$ **do**
    $sceneObj \leftarrow next(\Gamma_c)$
    if $isValid(sceneObj, c)$ **then**
      if $ValidateUnaries(sceneObj, URel)$ **then**
        $Neighbors \leftarrow getNeighbors(sceneObj, BRel)$
        if $Neighbors = \emptyset$ **then**
          $found = search(A, BRel, URel, space)$
        end if
        for each $neighbor \in Neighbors$ **do**
          $A = PutHead(A, neighbor)$;
          $spaceN \leftarrow getNSpace(sceneObj, neighbor, BRel)$;
          $found = search(A, BRel \setminus \{\beta_i(c, neighbor)\}, URel \setminus \{\mu_i(c)\}, spaceN)$
        end for
        $\Gamma \leftarrow \Gamma \cup sceneObj$
      end if
    end if
  end while
  return *found*
end if

3. the set $URel$ of unary relations satisfied by the models from A; generally, these are relations of type "size", as *isBig, isSmall, isAverage,*.etc..;

4. the searched space - for the first call of the *search* function, *space* is represented by the entire space scene.

For each concept c extracted from A we create a list of potential candidates (object in the scene) by calling the function $searchObjectIn()$ (a particular form of RANSAC algorithm). The neighborhood structure of the concept c (defined by all concepts α_i for which a binary relation $\beta_j(c, \alpha_i)$ is included in $BRel$) is determined only for the candidates satisfying the corresponding unary relations. The *search* function is then

called recursively for each element *neighbor* from the neighborhood, after adapting the input parameters (the concept *neighbor* is push in front of list A, and the binary/unary relations implying c and *neighbor* are deleted from the sets $BRel$ and $Urel$).

The probability value $P(\tau, |C|)$ used as the stopping criteria of RANSAC algorithm in its basic version is removed in the present algorithm, because this time the ending condition is different ($A = \varnothing$). Moreover, the minimal sets used to search the composition in the space are modified, and the search process itself is driven by the position relations (function $getNSpace(..)$). *Remark:* if the bounding region is defined by a cube, there are two approaches for defining what is a neighbor: the first one, denoted V_6, consider two cubes as neighbors if they share a common face; the second one, denoted V_{26}, include also the cubes sharing a ridge or a peak - like a Rubick cube. In order to avoid cycles of binary relations (e.g. $\beta(\alpha_i, \alpha_j), \beta(\alpha_j, \alpha_k), \beta(\alpha_k, \alpha_i)$) which "explode" the complexity of the $search()$ function, the right approach to be considered is V_6.

Modification of the score evaluation

The evaluation of the score has been modified in order to fit better with the new purpose of the algorithm. In the classical RANSAC algorithm we do not need to evaluate a precise score, but only to validate a candidate. Because the purpose of the modified algorithm is to create an ontology, we need to evaluate each of retrieved objects immediately after it is found.

This functionality is realized in the function $isValid(obj, \alpha_i)$. This function has to compare the object in its the mathematical representation with the set of points returned by the extraction process in order to measure the correspondence with the formal definition. Therefore the average distance between two points in the set τ has to be computed. The number of points that should be contained in the set is estimated by the value $\frac{area}{\tau^2}$, where the *area* depends obviously on the shape found.

Complexity for ontology-driven object detection algorithm The algorithm stops when all ontological concepts from the set $|A|$ are parsed. For each concept α_i a variable number of candidates $|\Gamma_{\alpha_i}|$ are identified. The iterative form of the algorithm is equivalent with a depth-first search algorithm on a tree T_Γ defined by:

- the root α_1;
- for all nodes on level $i, i = 1...n-1$, the number of successors is $|\Gamma_{\alpha_i}|$.

Therefore, the complexity of ontology-driven object detection algorithm is linear in the size of the graph, expressed as $\prod_{i=1}^{n} |\Gamma_{\alpha_i}|$. For each concept, the candidates are identified by a modified form of RANSAC algorithm, which implies a complexity of $O(|\Gamma_{\alpha_i}| \frac{C}{P_n})$ related to each node of the tree T_Γ. On the other hand, the identification of candidates is applied only for minimal sets (with a size $\ll N$), which allows us to approximate the parameters C and P_n with constants. Moreover, the same argument justifies the inequality $|\Gamma_{\alpha_i}| \leq M$, M constant, which implies a complexity of $O(M)$ in each node,

and thus a worst-case complexity of $O(M^n)$ for the ontology-driven object detection algorithm.

5.5 Conclusion

In this chapter, we presented two different groups of recognition systems: one using basic shapes or surface to fit a point cloud and another using a database of models.

For any model it is necessary to find an equilibrium between complexity and expressiveness. For example, a simple model for a shape is a shock graph, allowing to represent the skeleton of the shape. However a shock graph provides no information about the nodes and the relations between those nodes.

Furthermore we presented a new recognition system based on a semantic layer using ontologies and a common recognition systems. We presented several algorithms. A first algorithm was defined as a global scene extraction algorithm which can simplify the description of a scene and extract all the conceptual components present in the scene.

A second algorithm, called ontology-driven object detection, uses an ontology description as input to filter a point set in order to find an approximation of the objects defined by the ontology. This algorithm can deal with complex definitions of objects. The ontology approach lets the user define his own concepts and reuse them in another even more complex model.

Truth is stranger than fiction, but it is because Fiction is obliged to stick to possibilities; Truth isn't.

Mark Twain

6 RRR system

The idea behind the RRR system is to decompose the spatial interpretation process into three steps: Representation, Recognition and Retrieval. The three parts of the RRR System can be briefly summarized as follows:

- The Representation is done by adding a compact and meaningful description of a spatial concept into an ontology.

- During the Recognition step, new algorithms using ontologies, developed in Chapter 5, are applied. The framework is able to recognize shapes in a scene and export them into an ontology or make recognition using the elements represented in the ontology database.

- The Retrieval step uses the knowledge about spatial data or some information added by the user (rules) to reason about the spatial scene and to discover new knowledge that can be added in the semantic layer.

In this chapter we will have a look on the complete framework including the details of some implementations. Some scenarios will be developed to show how a user can interact with the ontology and the spatial scene. The first section 6.1 is a description of the global framework with the corresponding schemas. The next section 6.2 will describe the pre-processing steps to create groups of interesting points in the point cloud. In the section 6.3 we will see how the user is able to interact with the different components of the framework. In particulary we will describe the format file *ontology graph file* (ONG), which is a summary of the semantic composition of an object. This format file will be used in the implementation of the algorithm. Finally, in the section 6.4.1, we will explain the representation of a scene using the framework.

6.1 Global description of the framework

The RRR system framework receives as input a description of a model that has to be located in a 3-dimensional scene. The framework uses instances defined by basic shapes compositions, their properties and by some relations between the basic shapes. These compositions (representing models of complex objects) are expressed as geometrical scene descriptions (see section 4.2). The semantic layer used by the RRR system is the spatial ontology defined in Chapter 4. This ontology contains information about the shapes, the relations between objects, their position and relative size. Basic information defined in form of ontologies can be used as reference regardless if they are provided in the base system or if they are produced by users.

In the global framework, the semantic description is the most important concept. It creates a bridge between an abstract representation and a point cloud. We can distinguish at least three main tasks that can be achieved with our framework (see Figure 6.1):

(i) *ontology filtering*

(ii) *global scene extraction*

(iii) *ontology-based point cloud generation*

Ontology filtering is the capacity to use models from a database of basic 3D shapes to search the corresponding object in a spatial scene. An algorithm defined for this purpose was described in section 5.4.4. The global scene extraction process uses the point cloud as input and extract all the basic 3D shapes in the scene, encoded in a semantic layer. We described the algorithms for this purpose in section 5.4.3.

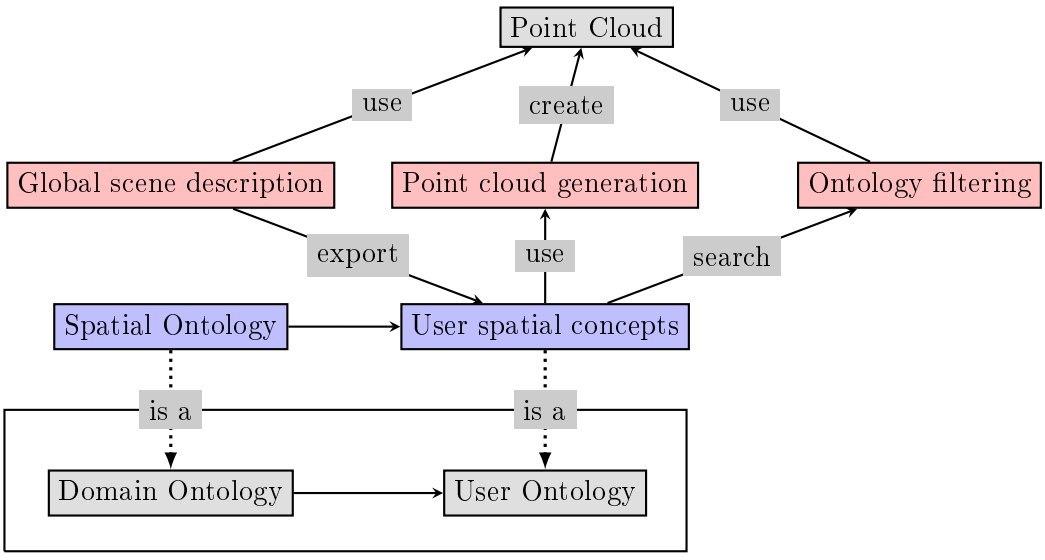

Figure 6.1: RRR framework: architecture of the spatial semantic approach.

6.2 Segmentation

The segmentation is a pre-processing step that will create labeled subsets of the whole scene. This step allows to create a distinction between some elements of the scene. We are calling these groups, which should represent meaningful sets, the *elements of interest* (EOI). Briefly, the segmentation consists of partitioning a set of measurements from a 3D object space (point cloud) into smaller, coherent and connected subsets [27].

A more formal way to define the segmentation for a point cloud R is the partitioning of R into a subset of regions $(R_1, R_2, ..., R_n)$, where each region is a connected component and, for some coherence predicate P, we have $P(R_i) = true, \forall (i \leq n \text{ and } i \neq j)$ and $P(R_i \cup R_j) = false$ for any adjacent regions. There are different approaches for segmentation, the main difference being the criterion used to measure the similarity for a set of points. The main categories are:

- *Edge-based segmentation* - this two-step process try firstly to detect the edges used to define the boundaries of a group. Once the limits are obtained, the interior points are added to get the final group. This approach can also be applied to detect planes from a point cloud.

- *Surface-based segmentation* - this approach was applied on point clouds in [45]. To make a segmentation based on the surface, an estimate of the curvature, based on the point cloud, must be calculated. The idea is to group the neighboring points whose normal vector differences are less than a given threshold α_{max}.

- *Scanline-based segmentation* - in a first step, an independent segmentation of each scan line is realized based on proximity, curve fit/height continuity and normal vector direction. During the second step, the scan line segments across the neighbouring scan lines are merged.

Once the segmentation is done, the different generated groups can be used as distinct point clouds for the RRR system. Therefore, this pre-processing step can be executed before any of the different processes described in Figure 6.1 are started.

6.3 User interaction

From a user point of view, we can define an interface displaying the basic shapes which may be used to compose more complex objects. On top of the set of basic shapes, we may add relations that bind the basic objects into complex objects denoted *compositions*. After having created visually the composition, we can call a script to export the composition either as an OWL file or as a corresponding ONG file. The OWL file is the standard XML format which is used to define the ontologies included by the system. The purpose of this format is to allow the use of reasoning processes. The ONG file is a simpler format file for defining a composition. The specifications of the ONG file is developed in 6.3.1.

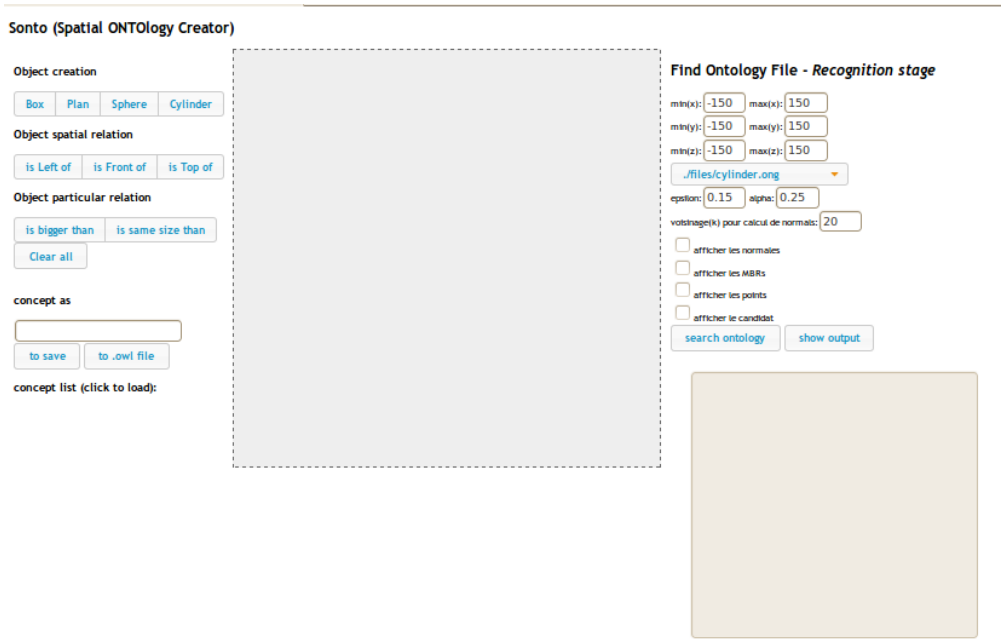

Figure 6.2: Web interface for the creation of a user ontology

We have implemented a web interface allowing the user to create his own composition in a graphical way. A screen-shot of the graphical interface can be seen in Figure 6.2. The interface permits the user to create instances of *basic3Dshapes*. These instances are linked by position relations or size relations. We allow the user to define size relations between two objects only if they are already linked together.

On the left side of the interface we have the basic primitives. Once we have chosen one by clicking on it, we can generate an instance of this object. The user can afterwards add new links by choosing a relation and clicking on two basic objects in the gray(working) space.

The files are created in a database of ontology as we can see in Figure 6.3. The user can simply add new concepts and visualize them either in a specialized ontology viewer, or using our implemented web interface. Any compositions done by the user can be stored directly in the ontology format (.owl) and in a (.ong) file. Each new created composition is considered to be a new concept and is added in the concept list situated on lower left window of the interface.

6.3.1 Ontology graph file (ONG)

We have different possibilities to describe the ontology that represents an object composed of primitive basic shapes. A first possibility is to use the OWL format. The OWL description of the spatial ontology contains the hierarchical knowledge about spatial objects. But this format is a language that has a lot of constraints. Instead we decided to

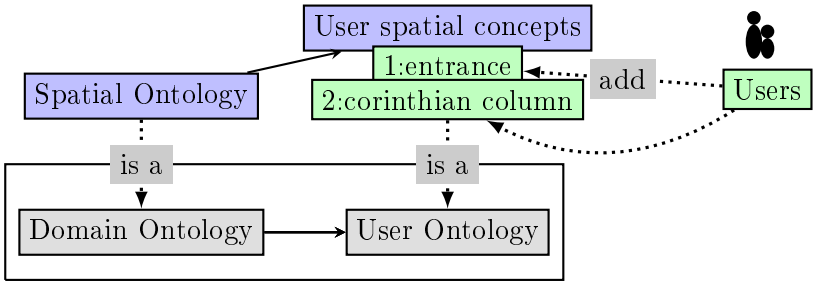

Figure 6.3: RRR framework: insertion of user concepts

```
0:sphere
1:cylinder
0:relativeOnTop:1
0:isBig
1:isSmall
```

```
<Sphere rdf:about="#0" >
   <hasConceptualSize rdf:resource="#big" />
   <onTop rdf:resource="#0" />
</Sphere>

<Cylinder rdf:about="#1">
   <hasConceptualSize rdf:resource="#small" />
</Cylinder>
```

Figure 6.4: Left side: ONG file example; Right side: the corresponding OWL file

use a simpler format to express the relations between the basic objects. We call this file format the ontology graph file.

In the ontology graph file we can define only four objects: *sphere*, *cylinder*, *plane* and *box*. Each of these four objects have a unique number (id) (e.g., 0 for sphere). Any object can be extended by some relations and properties. The basic position/size relations are the same as those defined in the spatial ontology: *relativeOnTop*, *relativeOnLeft* and *relativeOnFront*, respectively *isBig*, *isAverage* and *isSmall*.

This format is much simpler to parse for any system, because a lot of spatial information is implicitly known or not necessary for filtering process. But of course, if we are looking to infer knowledge, we need a more expressive language. In the Figure 6.4 we can see an example of an ONG file and of the corresponding OWL file. The same information can be expressed in both format, so we can pass from the (ONG) to the (OWL) with a simple re-writing process.

6.4 RRR stages

In the following subsections we will describe the specific challenges related to each of the three processing stage of the RRR system and our approach for solving these problems. This description follows the ideas and concepts related to the topic of semantic interpretation of 3D point clouds and to the design of a scene interpretation system, as we already published in [22].

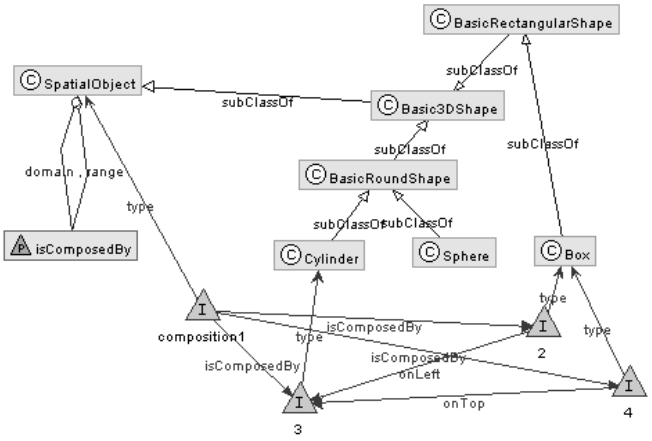

Figure 6.5: A composition of a complex object with the ontology

6.4.1 Representation stage

The objective of this step is to derive from the point cloud a rich, compact yet meaningful description of the objects for efficient storage and for fast and accurate retrieval during recognition. This can be done with the ontology defined in [19]. The ontology keeps only the conceptual view of the spatial scene. The complete definition of the semantic layer is given in section 4.2. In Figure 6.5 we can see a visual representation of an OWL file. The complex object, denoted *composition1*, is composed by the basic shapes named 2, 3 and 4, bound by some spatial relations between them. Visually, this composition represents a box on a cylinder which is on another box.

6.4.2 Recognition stage

The recognition step has two branches, the recognition through a database model or the decomposition of a spatial scene into primitives. The decomposition into spatial primitives in the semantic layer has been explained in the previous chapter in sections 5.4.3 and 5.4.4. Both are based on the RANSAC algorithm.

More generally, for the RRR framework, we can use different recognition systems to extract the ontology. We could use the Hough transform for the detection of planes in the ontology and a modified RANSAC algorithm for other shapes. Both tasks need to derive a spatial and geometric descriptions of the partial point cloud from the scene. These descriptions are compared with objects in order to identify which of those objects are present in the scene. This process involves the tasks of classification (of instances), determination of alignment parameters (rotation, translation) and localization. For the RRR framework we implemented different RANSAC-based but modified algorithms:

- Global scene extraction: algorithms for recognizing basic 3D shapes in the scene.

- Ontology driven detection: algorithms using defined compositions in an ontological form as input for the search in a point cloud.

The output of these algorithms is a semantic representation. Based on this information the retrieval stage can be implemented.

6.4.3 Retrieval stage

A real useful and valuable functionality of an intelligent computer vision system would be its capacity to describe an unknown scene as concisely as possible in terms of known objects, transformations of them, and of their mutual spatial relationships. Therefore, as we proposed in [23], we extend the meaning of the scene interpretation process by considering that an optimal interpretation of a scene is a symbolic description (based on a specific spatial language) which explains the scene in terms of the smallest number of known objects (i.e. known models) and simplest neighborhood relations between them, according to the Minimum Description Length Principle.

The spatial relationships existing between the objects in the scene are discovered by the *global scene extraction* process, defined before. In a first phase, these relations are analyzed by a pattern finding algorithm to extract possible regular patterns implying the models. In a second phase, a specific ontology describing the scene (which includes as instances the previously discovered relations), is processed to extract an optimal scene description.

In the retrieval stage, we want to use the specific ontology as input for a reasoning engine. This will add high-level knowledge and, in a further step, find inconsistencies in the conceptualization. Therefore the reasoning engine depends on the kind of reasoning used and on how the concepts are defined in the framework. The type of reasoning we want to apply to the semantic layer is based on first order logic. The reason of this choice is that, as underlined in [62], the more expressive logics are more difficult to reason with. Moreover, in the worst case scenario there exist no strategies that could ensure the termination of the reasoning process.

According to the MDL Principle, the optimal description minimizes the length of the set {*theory*, *data encoded using the theory*}. In our opinion, an appropriate theory for RRR system is represented by rules. Two types of rules of interest are:

- Spatial Association Rules : these rules describes "*the implication of one or a set of features by another set of features in spatial databases*" ([49]). An example of such a rule may be: $is\_a(X, column) \wedge close\_to(X, entrance) \rightarrow is\_big(X)$, where the spatial predicates are $close\_to()$ and $is\_big()$.

- Grammar Rules: the idea is to apply a semi-supervised grammar learning algorithm able to infer context-free grammars from a set of positive examples in an annotated corpus [81]. In our case the words are replaced by basic shapes, the binary relations between tokens are replaced by binary spatial relations and the positive examples are represented by a set of true sentences describing objects in the scene linked by spatial relations.

The rules found with these methods can be added to the common geometry rules, completing a database of rules. Some simple inference rules for the spatial ontology, which may be added without calling any mining process, are:

(i) $is\_left(x, y) \wedge is\_left(y, z) \rightarrow is\_left(x, z)$,

(ii) $is\_left(x, y) \rightarrow is\_right(y, x)$,

(iii) $is\_bigger(x, y) \wedge is\_bigger(y, z) \rightarrow is\_bigger(x, z)$.

6.5 Experiments

In order to validate our approach we conducted a series of experiments. In a simulated experiment we firstly chose some basic compositions of two objects, e.g. a cylinder on top of a sphere (these two objects have only one relation of the type *relative position* and some size properties). In a second phase we generated the point cloud that follows this model. Finally we checked how well our algorithm performed for reconstructing the original model in form of an ontology.

We are able to show in this experiments that, contrary to the classical RANSAC algorithm, only a very limited part of the scene has to be searched (implying a drastically increases of the efficiency of the algorithm). The ontology description is also very similar to the description given by a human being while looking at the scene.

A second series of experiments were conducted on real data, representing a 3D scan of the Pantheon in Rome. In the following section we will describe how these point clouds where used and the detailed results obtained for the two algorithms.

6.5.1 Ontology filtering experiments

For our experiments we created a basic ontology representing a composition of two basic shapes, a cylinder on the top of a sphere. These two objects are characterized by only one *position* relation and one *size* relation. The center of the sphere, with a radius of 20, is at $(0, 0, -20)$, whereas the center of the cylinder, with a radius of 10 and a length of 15, is at $(0, 0, 15)$. In Figure 6.6 we see the ontological point of view.

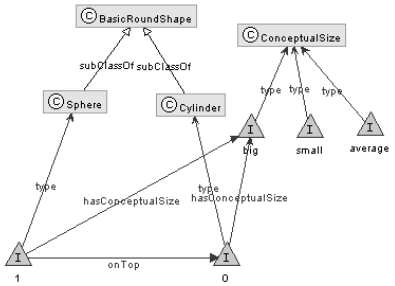

Figure 6.6: A cylinder on the top of a sphere: the ontological representation

We generated a point cloud represented by a random sample of points corresponding to these two basic shapes. This point cloud corresponds to the concrete representation of the ontology. The algorithm is searching the objects by extracting appropriate blocs, in which the target shapes are identified using a list of models. In Figure 6.7 we may see the boxes identified as containing the composition, together with the objects found by the algorithm (in grey). Contrary to RANSAC algorithm, our algorithm is able to use the conceptual information given by the user, by creating a bridge between the ontological representation and the point cloud.

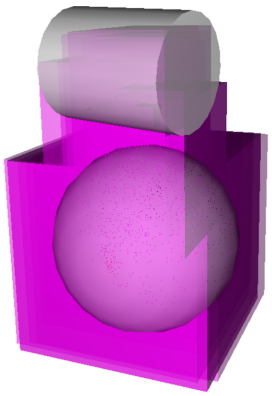

Figure 6.7: Blocs where the objects have been searched

To check the robustness of the algorithm against noise, we created different scenes with shapes (sphere and cylinder) having some errors on the points (errors relative to the geometrical model of the shape). Three different point clouds were generated, containing a percent of noise points of 0%, 5% and 10%. For each point cloud, eight trials were made to find the composition.

The presence of the noise affects the quality of the approximation of the normals, which finally conducts to wrong estimations of the center's shapes. However, the normals are not always correctly calculated even for a noise free point cloud, because the estimation is made based on the k-neighbourhood of the points for which we want the normals. If the parameter k is too small, the estimation can be wrong because the lack of

Noise	**Comp.**	**Param.**	**Param.**	**Sphere**	**Cylinder**	
in (%)	**found**	α	ϵ	*center error*	*radius error*	*center error*
0	6	0.25	0.2	2.885	0.659	0.619
5	5	0.25	0.2	4.888	0.589	0.328
10	1	0.25	0.2	23.797	1.585	2.068

Table 6.1: Summary of eight trials for noisy point cloud

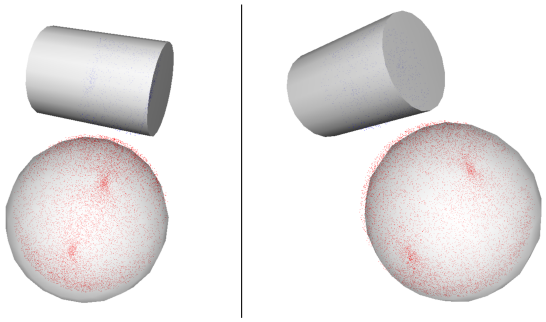

Table 6.2: The identified composition (cylinder on sphere) in a point cloud with 0% noise (on left) and with 5% noise (on right)

information (on the point vicinity) is too high; and if k is too large, the estimation of the normal is not a good local estimation. These errors can lead to a wrong center for shapes (in our case, sphere or cylinder) and so the search process is sometimes ending without having found a candidate (and, consequently, a composition). Table 6.1 summarizes the results of the eight trials applied on three point clouds with different percentages of noise. The column "Comp. found" shows the number of times the algorithm found the composition during the trials. The parameter α is the maximum angle error tolerance, whereas the parameter ϵ is the maximum distance of the point accepted. This two parameters are the same as those defined by the RANSAC algorithm, the angle between the normal of the candidate shape and the normal of the point should be lower than α. And the ϵ is for the ϵ-band where the points are considered for the score computation. The columns "center/radius error" were calculated as an average of the absolute difference between the true center/radius of the objects and the estimated center/radius of candidates found in the point clouds.

We must notice that our algorithm stops only if the entire composition (here, the two objects) is found. This implies that if the complexity of the composed object increases, the probability to find the composition gets smaller. In order to compensate this degradation of the performance, we can decrease the angle error tolerance (α) and the number of points that is needed to have a shape candidate for highly complex objects.

As we can see in Table 6.1, when we increase the noise up to 10%, the matching algorithm is no more able to find the composition. By increasing the tolerance angle error or the tolerance distance error, the capacity of the algorithm to find the objects is also increased. In Table 6.2 we can see the fitted objects (in grey) against the basic point cloud (in blue and red). The approximations obtained in these trails are very close to the real generated objects, which implies that the search is efficient even with 5% of error for the cloud points.

6.5.2 Global scene extraction experiments

For the evaluation of the global scene extraction, in a first experiment we started again with a point cloud representing a sample of the same complex object, a cylinder on top of a sphere. After the extraction, we obtained the representation of the ontology in an OWL file, including the approximations for the properties *size* and *position* for the objects.

The second experiment was conducted on the point cloud representing the Pantheon of Rome. In this scene, the shape including the largest number of points is the "floor" of the Pantheon. The algorithm found that it could be approximated by a sphere of radius 1252.726685. This might be surprising at the first glance but this result showed that the difference in altitude between the point situated at the center of the Pantheon's floor and a point situated at the border of Pantheon's floor is approximately of 33 centimeters, which corresponds to the expected information. The floor of the Pantheon is curved because the rain which floods in the building through the hole of the roof must be evacuated to the borders of the building. We can see in the Figure 6.8 the ontology that represents the Pantheon floor. This ontology has been extracted by the global scene algorithm.

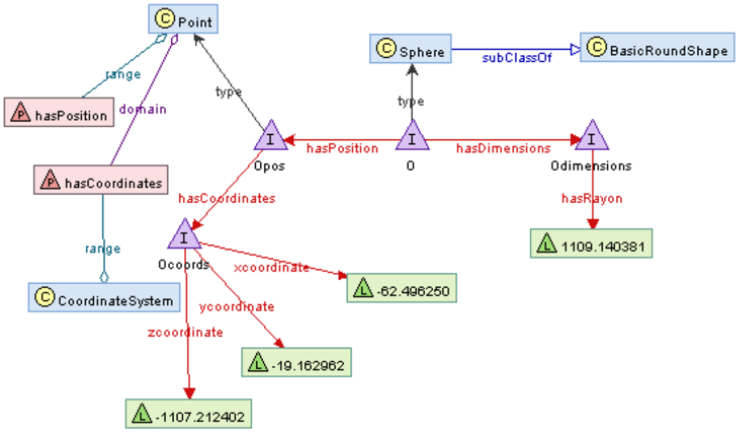

Figure 6.8: Ontology representation of the Pantheon floor

The composition obtained by the global scene extraction process can be named by the user and saved into the database. As explained in 5.4.3, the links between the objects may be created after the objects have been extracted from the scene. The orientation, size and distance relations can be added after going through the ontology using mathematical comparisons.

6.6 Conclusion

The RRR framework is a flexible and powerful tool, allowing the mapping between the point clouds and a mathematical and/or a semantic description. The user may define new compositions, name them and search for them in a virtual scene. All these procedures are implemented in different modules, building the three principal components of the proposed framework. We also showed in this chapter that the framework can improve the way in which we describe objects. This description can be used either to find objects, or as a global representation of a scene. We have also explained the approach of learning rules about user defined concepts from a spatial scene.

Finally, we conducted a number of experiments, on both simulated and real data (3D scan of the Pantheon in Rome), in order to validate the processes that can be realized with our framework. A particular attention was given to evaluate the performance of the algorithm in the presence of noisy points and we were able to conclude that the search procedure is still efficient even with 5% of error in the point cloud.

7 Conclusion

During the last couple of years, point clouds have emerged as a new standard for the representation of largely detailed models. This is partly due to the fact that range scanning devices are becoming a fast and economical way to capture dense point clouds. However, the huge amount of data captured during the acquisition phase may limit the applicability of the algorithms and methodologies currently developed for 3D computer vision. At the same time, it is obvious that the process of extracting useful knowledge or models from unstructured information spaces (and a point cloud is such a space) is a topic situated at the junction of several research fields, as spatial data mining, computer graphics, pattern recognition, machine learning, spatial reasoning, data bases/data warehousing, etc.. Our vision about the treatment of this topic is linked to a new approach implying the use of techniques and methodologies from Artificial Intelligence and Knowledge Management in Computer Vision.

The main goal of this thesis was the development of a flexible approach (including framework, methodology, processing methods and finally a working system) for the analysis of a scene containing a large number of objects, represented as unstructured point clouds. The analysis process (also denoted as scene interpretation) is not limited - in our opinion - only to the task of identifying *which* object is located *where* in the scene, but also of *how* the objects are related, which implies the discovery of the spatial relationships between these objects. The information about the objects found in the scene and the relationships connecting them allows us the extraction of the "optimal scene interpretation", a scene description based on a specific scene ontology and satisfying the MDL principle. To reduce the complexity of the scene interpretation process in the perspective of the large diversity of real-world situations, the framework assumes that the objects of interest are rigid, free-form shapes (so no statistical defined shapes) and a database of models for each object already exists.

A simple description language allowing the scene interpretation must include at least rigid, opaque 3D objects, and a set of spatial relationships. From a technical viewpoint, our approach was to "encode" such description language inside a dynamically created semantic layer, added to our 3D point cloud, and expressed as a set of ontologies (denoted as the reference ontology), for adding/sharing conceptual knowledge about spatial objects and comprising the description of different systems and representation models that might be used.

The semantic layer (described in Chapter 4) is mainly composed of two groups of ontologies: the upper (basic) group and the lower (user) group. Without losing in generally, the user can describe a spatial object in a specific environment by actually constructing the 3D object from elementary shapes. Each elementary shape is described mainly by a transformation (scaling, translation, rotation), one or more positions and one or more dimensions. Since each transformation can be expressed in different ways or is shape-dependent, the upper ontologies comprise the description of different systems and mathematical models that might be used:

1. The *Coordinate Systems Ontology*. This ontology defines a few systems for describing a position in space: cartesian, spherical and cylindrical - the ontology being easily extensible with other systems.

2. The *Systems Ontology*. The same approach as above has been used for the transformations ontology, i.e. each instantiable rotation system has predefined attributes (e.g. roll angle, vector, etc.) that match their corresponding mathematical elements.

3. The *Geometrical Shapes Ontology*. This ontology is the most complex one and it formalizes the fundamental geometrical shapes such as cubes, cylinders, spheres, torus, etc. The central concept of this ontology is the SpatialObjet, all basic shapes as well as any user-defined spatial object being subclasses or instances of the SpatialObjet concept.

4. The *Contextual Relations Ontology*. This ontology defines spatial relation (related to the size -e.g. *isBig* - or the position of an object - e.g. *isNearTo*) for which the meaning depends on the point of view of the observer.

The complete framework supporting the interaction of the user with the spatial scene and the semantic layer (denoted RRR system) was designed as a three stage process. The system receives as input a description of a model that has to be located in a 3-dimensional scene and uses instances defined by basic shapes compositions, their properties and by some relations between the basic shapes. This semantic description creates a bridge between an abstract representation and the spatial scene defined by the point cloud. Stated succinctly, the three stages of the RRR System can be summarized as follows:

1. *Representation*: For each basic object type, a compact and meaningful model, based on point cloud representation, is proposed. The model must allow efficient storage and a fast retrieval process.

2. *Recognition*: Various characteristics (spatial, topological) extracted from partial point cloud are compared with models defined in the Representation stage to identify the objects present in the scene.

3. *Retrieval*: A complete scene description (the set of object instances and the set of spatial relationships between these instances, inferred by a reasoning engine) is expressed using a spatial description language.

The choice of the object representation (which is largely detailed in Chapter 2) is one of the most important decision for the performance of our RRR system, and must be accompanied by robust techniques for extracting compatible features from both the object model and the input point cloud. Between the two fundamental categories of representation, *object-centered* and *view-centered*, the nature of input data and the objective of the system clearly impose techniques from the first category, which attempt to describe the entire 3D volume occupied by the object. Moreover, a necessary condition for the object representation is the existence of an enough simple ontology representation, which favorises especially the geometric shapes model and excludes (for the moment) the mesh representation.

Recognition is performed by matching features derived from the scene with those stored in the model database. The matching strategies (graph matching, information-theoretic matching, iterative model fitting) are either spatial or structural. As example, the graph-based structural approaches is highly sensitive to noise and perform well as long as the scene and the model graphs have the similar number of points (see [43]). On the other hand, in information theoretic approaches the matching is performed purely in the feature/spatial domain.

Between the two possible approaches for object detection - one based on registered objects (signatures) in database, the other based on geometrical primitives - we were focused on the second one. In this case the detection algorithm uses primitive basic shapes, such as spheres, boxes, cylinders, planes, torus etc. in order to detect the objects. We implemented two algorithms (see details in Chapter 5), both extensions of the classical RANSAC heuristic, which take as input the point cloud and provide a concise description of a spatial scene, using ontologies as representation tool. The difference between the two algorithms consists in their output: whereas the first algorithm generate a complete description of the scene as basic 3D shapes with their relations (relative distance and size) in the space, the second algorithm identifies (or not) in the scene a specific description, using a semantic representation in the input to filter the objects.

The process of identifying multiple objects in a spatial scene demands also careful analysis of the underlying database technology. With increasing size of the database, the importance of a system's method for quickly and accurately indexing the selected model becomes more important. For a large set of input data represented by unstructured point clouds, even the task of simply identifying the set of points closest to a particular point in the cloud could be computationally expensive. The speed of the search process in spatial databases is strictly correlated with the choice of the indexing technique. From the multitude of possible techniques described in Chapter 3 (kd-trees, grid files, buddy-trees,

BD-trees, etc.), we opted for an adapted version of the X-tree approach, as it satisfies best the needs of the semantic part of our system. In the X-tree approach, the super-nodes are created during insertion only if there is no other possibility to avoid overlap, which will increase the speed of the queries made in the spatial database system. Moreover, to optimize the query engine, we implemented a set of sophisticated range queries allowing to extract points corresponding to some regions defined by basic shapes, i.e. queries extracting the interior of the shape and queries extracting the ϵ-band around the surface of the shape.

The experiments conducted both on simulated and real data (a 3D digital model with more than 620,000,000 points of the Pantheon in Rome) validated the processes that can be realized with our framework. A very convincing proof was given by the confirmation, by the global scene extraction algorithm, of the spherical shape of the Pantheon floor (curvature which was already bring out by the specialists).

7.1 Future work

As we underlined, the three stages of the proposed framework (representation, recognition and retrieval) are implying various research domains in computer science. The approaches made in this thesis are not covering all the possibilities of the RRR system. Each module of the framework can be implemented in several ways, or some new modules can be added. For example, even if it was formally described in the Section 6.4.3, the module representing the reasoning engine for processing the low-level knowledge structures captured in the reference ontology was not implemented in the working system. This module, designed to deduce new, high-level knowledge and to signal inconsistencies in the conceptualizations, will be implemented in a future version of the RRR system. And due to the huge dimension of input data and of its nature (point cloud stored in a spatial database system), this engine may use one or several data mining tasks, briefly described in the following.

Spatial clustering. The clustering is the task of grouping set of objects into subclasses (clusters), based on the principle that the members of a cluster should be as similar as possible. When this task is applied on spatial data, the clusters obtained from the initial point cloud can be considered as parts of the spatial scene. In the literature, different types of algorithms have been proposed, as CLARANS (k-medoid clustering) [66], DBCLASD (which assumes that the items within a cluster are uniformly distributed) [101], BANG (which uses a tree) and WaveCluster [95] approach (the n-dimensional space is seen as a signal).

In the recognition process, the spatial clustering can be used to make a segmentation of the point cloud. A fast and reliable segmentation process is able to simplify the geometric reconstruction process. For example, if we have a point cloud that describes planes, we may apply a method developed in [60] consisting of three steps: *mesh construction*, *extract parameters of triangle mesh* and *apply clustering method*. After these steps, the meshes with similar parameters are considered to be in the same plane.

Spatial classification. The task of classification is to assign an object to a class from a given set of classes based on the attributes of this object. For spatial objects, two kind of attributes are considered: non-spatial and spatial attributs. In the neighborhood graph introduced in [31], the edges represent spatial relations whereas the nodes represent the objects in the database. Two nodes (corresponding to the objects n_1 and n_2) are connected by an edge if and only if the predicate $neighbor(object(n_1), object(n_2))$ holds. A *neighbor* relation may be expressed either by a *topological relation* (similar to the $ALCI_{RCC8}$ described in Section 4.4.2), or by a *metric relation*, or a *direction relation*. For the RRR system, the process of classification could be performed after the recognition process has decomposed the scene into basic objects.

Spatial rules. Spatial rules describe associations between objects based on spatial neighbourhood relations. There are three types of rules that can be found by spatial data mining techniques: *spatial characteristic rules*, which are used to define a general description of a set of spatial-related data; *spatial discriminant rules* which describe the difference between different classes of the data; and *spatial association rules* which are association rules about spatial data objects, where either the antecedent or the consequent of the rule must contain some spatial predicates. These rules could be adapted to the representation of a spatial scene [50]. As example, an illustration of what could be obtained as an optimal interpretation of the scene representing the pantheon entrance (Figure 4.3), using spatial rules as encoding theory, is the following set:

```
{ RULES:
    R1: IF instance1 of CorinthianColumn THEN instance2 of CorinthianColumn
    relativeToLeft atDistance d
 DATA:
    instanceA # instance of the last column at right, satisfying the rule R1
    instanceB # instance of the first column at left, exception for the rule R1
}
```

7.1.1 Reference ontology extension

The reference ontology, as we introduced in Chapter 4, can describe the common basic objects and the relations between these objects. Moreover, the ontology allows to choose the coordinate system or the transformation system. We also added an ontology for contextual relations related to objects size and position. The capacity of the ontology model to be extended by including new concepts is one of the main arguments for the integration of the semantic layer in our framework.

Shape description. A possible extension is the inclusion of non-static objects, which allows to represent the deformable objects described in the section 2.2.2. If this extension is used to represent the variation over time of the dynamic shapes, then the reference ontology may be considered as a spatio-temporal ontology. Another interesting extension is the one allowing to specify objects features, i.e. particular aspects of the objects (not

only geometrical, but also physical, like the color). The class of shape representation may be also extended, allowing to have, for the same object, different models (parametric forms, superquadratics, implicit surfaces, mesh representation).

To conclude this section, we can assert that in all the components of the framework we can make some improvement. Theses improvements should be thought to respond to some specification given by a user or to new objectives imposed to the system. The design of the framework is enough flexible to accept all these changes.

'When we open our eyes each morning, it is upon a world we have spent lifetime learning to see. *Oliver Sacks*'

Bibliography

[1] Agathos, A., Pratikakis, I., Papadakis, P., Perantonis, S., Azariadis, P., and Sapidis, N. Retrieval of 3d articulated objects using a graph-based representation. In *Proceeding in Eurographics Workshop on 3D Object Retrieval* (2009).

[2] Amenta, N., and Bern, M. Surface reconstruction by voronoi filtering. *Discrete and Computational Geometry 22* (1998), 481–504.

[3] Amenta, N., and Ki, J. Defining point-set surfaces. In *Proceedings of ACM SIGGRAPH* (2004), A. Press, Ed., pp. 264–270.

[4] An, N., yu Yang, Z., and Sivasubramaniam, A. Selectivity estimation for spatial joins. In *IEEE ICDE* (2001), pp. 368–375.

[5] Arya, S., and Fu, H.-Y. A. Expected-case complexity of approximate nearest neighbor searching. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2000), SODA '00, Society for Industrial and Applied Mathematics, pp. 379–388.

[6] Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A. Y. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM 45* (November 1998), 891–923.

[7] Banerjee, J., and Kim, W. Supporting vlsi geometry operations in a database system. *IEEE Proc. Data Engineering Conf.* (1986), 409–415.

[8] Bayer, R. The universal b-tree for multidimensional indexing: general concepts. In *Proceedings of the International Conference on Worldwide Computing and Its Applications* (London, UK, 1997), WWCA '97, Springer-Verlag, pp. 198–209.

[9] Bayer, R., and McCreight, E. Organization and maintenance of large ordered indices. *Acta Informatica 1 3* (1972), 173–189.

[10] Bentley, J. Multidimensional binary search trees in database applications. *IEEE Transactions on Software Engineering 4* (1979), 333–340.

[11] Bentley, J. L. Multidimensional binary search trees used for associative searching. *Commun. ACM 18* (September 1975), 509–517.

[12] Berman, R., and Stonebraker, M. Geo-quel, a system for the manipulation and display of geographic data. *ACM Comp. Graphics 11* (1977), 186–191.

[13] Bern, M., and Eppstein, D. Mesh generation and optimal triangulation. *Lecture Notes Series on Computing 1* (1992), 23–90.

[14] Campbell, R. J., and Flynn, P. J. A survey of free-form object representation and recognition techniques. *Computer Vision and Image Understanding 81*, 2 (2001), 166–210.

[15] Chamberlin, D., Astrahan, M., and Eswaran, K. Sequel 2: a unified approach to data definition, manipulation and control. *IBM J. Res. and Develop. 20* (1976), 560–575.

[16] Chang, N., and Fu, K. Query-by-pictorial-example. *IEEE Trans. Software Eng. 6* (1980), 519–524.

[17] Chen, Y., and Medioni, G. Object modeling by registration of multiple range images. In *Proc. IEEE Conf. on Robotics and Automation* (1991).

[18] Cheng, Z.-Q., Wang, Y.-Z., Li, B., Xu, K., Dang, G., and Jin, S.-Y. A survey of methods for moving least squares surfaces. In *Volume Graphics* (2008), pp. 9–23.

[19] Ciorascu, C., Künzi, C., and Stoffel, K. Overview: Semantic management of 3d objects. *Proceedings of the SAMT Workshop on Semantic 3D Media* (Dec. 2008), 39–44.

[20] Clementini, E., Di Felice, P., and Hernández, D. Qualitative representation of positional information. *Artif. Intell. 95* (September 1997), 317–356.

[21] Comer, D. The ubiquitous b-tree. *ACM Comp. Surveys 11* (1979), 121–137.

[22] Cotofrei, P., Künzi, C., and Stoffel, K. Semantic interpretation of 3d point clouds of historical objects. In *Proceedings of DiPP 2011* (2011).

[23] Cotofrei, P., Kunzi, C., and Stoffel, K. Optimal scene interpretation: Semantic management of 3-d object from a point cloud scene. *Proceedings of International Conference of Computer Graphics and Artificial Intelligence* (2009), 189–196.

[24] Dey, T. *Curve and Surface Reconstruction: Algorithms with Mathematical Analysis*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2011.

[25] Dey, T. K., and Goswami, S. Provable surface reconstruction from noisy samples. *Comput. Geom. Theory Appl. 35* (August 2006), 124–141.

[26] Dey, T. K., and Sun, J. An adaptive mls surface for reconstruction with guarantees. In *Proceedings of the third Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2005), Eurographics Association.

[27] Dorninger, P., and Nothegger, C. 3d segmentation of unstructured point clouds for building modelling. In *Photogrammetric Image Analysis (PIA07)* (2007), U. Stilla, H. Mayer, F. Rottensteiner, C. Heipke, and S. Hinz, Eds., vol. 35, ISPRS, pp. 191–196.

[28] Eastman, C. M., and Zemankova, M. Partially specified nearest neighbour using kd trees. *Information Processing Letters 15 2* (1982), 53–56.

[29] Egenhofer, M. Spatial SQL: A query and presentation language. *IEEE Transactions on Knowledge and Data Engineering 6(1)* (1994), 86–95.

[30] Embley, D., and Nagy, G. Toward a high-level integrated image database system. *Technical Report, Image Processing Laboratory* (1986).

[31] Ester, M., Krieger, H., and Sander, J. Spatial data mining: a database approach. *Proc. of 5th Int. Symp. on Large Spatial databases.* (1997), 47–66.

[32] Fagin, R., Nievergelt, J., Pippenger, N., and Strong, H. R. Extendible hashing : A fast access method for dynamic files. *ACM Trans. on database sys. 4 3* (1979), 315–344.

[33] Finkel, R. A., and Bentley, J. L. Quad trees: A data structure for retrieval on composite keys. *Acta informatica 4* (1974), 1–9.

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

[35] Grenon, P., and Smith, B. SNAP and SPAN: Towards Dynamic Spatial Ontology. *Spatial Cognition and Computation 4*, 1 (2004), 69–104.

[36] Gunther, O. Efficient structures for geometric data management. *Lecture Notes in Computer Science 337* (1988).

[37] Guttman, A. R-trees: A dynamic index structure for spatial searching. *Proc. ACM SIGMOD Int. Conf. on Management of Data* (1984), 47–57.

[38] Henrich, A., werner Six, H., Hagen, F., and Hagen, F. The lsd tree: spatial access to multidimensional point and non-point objects. In *Proc. Int. Conf. on Very Large Data Bases* (1989), pp. 45–53.

[39] Herring, J. Tigris: topologically integrated geographic information systems. *Proc. AUTO-CARTO 8* (1987), 282–291.

[40] Hilbert, D. Uber die steitige abbildung einer linie auf ein flachenstuck. *Math. Ann. 38* (1891).

[41] Hinrichs, K., and Nievergelt, J. The grid file: A data structure designed to support proximity queries on spatial objects. *Proc. Int. Workshop on Graphtheoritic Concepts in Comp* (1983), 100–113.

[42] Ingram, K., and Phillips, W. Geographical information processing using a sql-based query language. *Proceedings of 8th Int. Symp. on Computer-Assisted Cartography* (1987), 326–335.

[43] Jagannathan, A. *Segmentation and Recognition of 3D Point Clouds within Graph-theoretic and Thermodynamic Frameworks*. PhD thesis, Department of Electrical and Computer Engineering, Boston Northeastern University, 2005.

[44] Jaklič, A., Leonardis, A., and Solina, F. *Segmentation and Recovery of Superquadrics*. Kluwer Academic Publishers, 2000.

[45] Klasing, K. Surface-based segmentation of 3d range data, 2009.

[46] Klein, R., Liebich, G., and Strasser, W. Mesh reduction with error control. In *Visualization 96. ACM* (1996), pp. 311–318.

[47] Knuth, D. E. *Fundamental Algorithms, 2nd Edition. The art of computer programming, vol. 1.* Addison-Wesley, Englewood Cliffs, N.J., 1973.

[48] Kobbelt, L., Campagna, S., and peter Seidel, H. A general framework for mesh decimation. In *in Proceedings of Graphics Interface* (1998), pp. 43–50.

[49] Koperski, K., and Han, J. Discovery of spatial association rules in geographic information databases. *Proc. of 4th Int. Symp. on Large Spatial Databases (SSD 95)* (1995), 47–66.

[50] Koperski, K., and Han, J. Discovery of spatial association rules in geographic information databases. In *SSD* (1995), M. J. Egenhofer and J. R. Herring, Eds., vol. 951 of *Lecture Notes in Computer Science*, Springer, pp. 47–66.

[51] Kriegel, H., and Seeger, B. Multidimensional order preserving linear hashing with partial expansion. *Proc. Int. Conf. on Database Theory* (1986), 203–220.

[52] Kriegel, H. P. Performance comparison of index structures for multi-key retrieval. *Proc. ACM SIGMOD Int. Conf. on Management of Data* (1984), 186–196.

[53] Lennes, N. J. Theorems on the simple finite polygon and polyhedron. *Am. J. Math. 33* (1911), 37–62.

[54] Levoy, M., Pulli, K., Curless, B., and all. The digital Michelangelo project: 3D scanning of large statues. In *SIGGRAPH* (2000), pp. 131–144.

[55] Lin, K. The TV-tree: An index structure for high-dimensional data. *VLDB Journal vol. 3 4* (1994), 517–542.

[56] Litwin, W. Linear hashing: A new tool for file and table adressing. In *vldb80* (october 1980).

[57] Lomet, D., and Salzberg, B. The hb-tree: a robust multi-attribute search indexing method. *Proc. IEEE 5th Int. Conf. on Data Engineering* (1989).

[58] Lomet, D., and Salzberg, B. The hb-tree: a multiattribute indexing method with good guaranteed performance. *ACM Trans. on Database Sys. 15 4* (1990).

[59] Lu, W., and Han, J. Distance-associated join indices for spatial range search. *Proc. 8th IEEE. Int. Conf. on Data Engineering* (1992), 284–292.

[60] Maas, H., and Vosselman, G. Two algorithms for extracting building models from raw laser altimetry data. *ISPRS Journal of Photogrammetry and Remote Sensing 54*, 2-3 (1999), 153–163.

[61] Mémoli, F., and Sapiro, G. Gromov-hausdorff. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (New York, NY, USA, 2004), ACM, pp. 32–40.

[62] Mika, P., and Akkermans, H. Analysis of the state-of-the-art in ontology-based knowledge management. Tech. rep., Vrije Universiteit, Amsterdam, 2003.

[63] Mémoli, F., and Sapiro, G. A theoretical and computational framework for isometry invariant recognition of point cloud data. *Found. Comput. Math. 5* (July 2005), 313–347.

[64] Neumann, B., and Möller, R. On scene interpretation with description logics. *Image Vision Comput. 26* (January 2008), 82–101.

[65] Neumann, B., and Weiss, T. Navigating through logic-based scene models for high-level scene interpretations. *Proceedings of 3rd Int. Conf. on Computer Vision Systems (ICVS-2003)* (2003), 212–222.

[66] Ng, R. T., and Han, J. Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering 14*, 5 (2002), 1003–1016.

[67] Nievergelt, J., Hinterberger, H., and Sevcik, K. The grid file: An adaptable, symmetric multikey file structure. *In Trends in Information Processing Systems, Proc. 3rd ECI Conf.* (1981), 236–251.

[68] Nievergelt, J., Hinterberger, H., and Sevcik, K. C. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. on Database Sys. 9 1* (1984), 38–71.

[69] Nievergelt, J., Hinterberger, H., and Sevcik, K. C. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst. 9* (March 1984), 38–71.

[70] Ohsawa, Y., and Sakauchi, M. The bd-tree : a new n-dimensional data structure with highly efficient dynamic characteristics. *Proc. IFIP Congress* (1983), 539–544.

[71] Ooi, B., Mcdonell, K. J., and Davis, S. R. Spatial kd-tree: An Indexing Mechanism for Spatial Database. *COMPSAC conf.* (1987), 433–438.

[72] Ooi, B., Sacks-Davis, R., and Han, J. Indexing in spatial databases. *Unpublished/Technical Papers* (1993).

[73] Ooi, B. C., Sacks-Davis, R., and McDonell, K. J. An integrated topologic database design for geographic information systems. *Photogrammetric Engineering and Remote Sensing 53(10)* (1987), 1399–1402.

[74] Orenstein, J. A. Spatial query processing in an object-oriented database system. *Proc. ACM SIGMOD Int. Conf. on Management of Data* (1986), 326–336.

[75] Orenstein, J. A., and Merrett, T. H. A class of data structures for associative searching. *Proc. ACM PODS Conf.* (1984), 181–190.

[76] Ouksel, M., and Scheuermann, P. Multidimensional b-trees: Analysis of dynamic behaviour. *BIT 21* (1981), 401–418.

[77] Overby, J., Bodum, L., Kjems, E., and Ilsøe, P. M. Automatic 3d building reconstruction from airbornelaser scanning and cadastral data using hough transform. ASPRS, 2004.

[78] Pan, G., and Wu, Z. 3d face recognition from range data, 2005.

[79] Pauly, M., Gross, M., and Kobbelt, L. P. Efficient simplification of point-sampled surfaces. In *Proceedings of the conference on Visualization '02* (Washington, DC, USA, 2002), VIS '02, IEEE Computer Society, pp. 163–170.

[80] Peano, G. Sur une courbe qui remplit toute une aire plaine. *Annales Mathematiques 36* (1890), 157–160.

[81] Petasis, G., Karkaletsis, V., Paliouras, G., and Spyropoulos, C. D. Learning context-free grammars to extract relations from text. In *Proceedings of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence* (Amsterdam, The Netherlands, The Netherlands, 2008), IOS Press, pp. 303–307.

[82] Poosala, V., Haas, P. J., Ioannidis, Y. E., and Shekita, E. J. Improved histograms for selectivity estimation of range predicates. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 1996), ACM, pp. 294–305.

[83] Randell, D. A., Cui, Z., and Cohn, A. G. A spatial logic based on regions and connection. In *PROCEEDINGS 3RD INTERNATIONAL CONFERENCE ON KNOWLEDGE REPRESENTATION AND REASONING* (1992).

[84] Reis, R. M. P., Egenhofer, M. J., and Matos, J. L. G. Conceptual neighborhoods of topological relations between lines. In *SDH* (2008), pp. 557–574.

[85] Robinson, J. T. The k-d-b-tree: A search structure for large multidimensio-nal dynamic indexe. *Proc. ACM SIGMOD Int. Conf. on Management of Data* (1981), 10–18.

[86] Roussopoulos, N., and Leifker, D. Direct spatial search on pictorial databases using packed r-trees. *Proc. ACM SIGMOD Int. Conf. on Management of Data* (1985), 17–31.

[87] Rusinkiewicz, S., and Levoy, M. Efficient variants of the icp algorithm. In *Third International Conference on 3D Digital Imaging and Modeling* (2001).

[88] Samet, H. The quadtree and related hierarchical data structures. *ACM Comp. Surveys 16* (1984), 187–260.

[89] Scheuermann, P., and Ouksel, M. Multidimensional b-trees for associative searching in database systems. *Information Systems 7 2* (1982), 123–137.

[90] Schnabel, R., Wahl, R., and Klein, R. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum 26*, 2 (June 2007), 214–226.

[91] Sebastian, T. B., and Kimia, B. B. Curves vs skeletons in object recognition. In *In IEEE International Conference of Image Processing* (2001), pp. 247–263.

[92] Seeger, B., and Kriegel, H.-P. Techniques for design and implementation of efficient spatial access methods. In *Proceedings of the 14th International Conference on Very Large Data Bases* (San Francisco, CA, USA, 1988), VLDB '88, Morgan Kaufmann Publishers Inc., pp. 360–371.

[93] Seeger, B., and Kriegel, H.-P. The buddy-tree: An efficient and robust access method for spatial data base systems. In *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings* (1990), D. McLeod, R. Sacks-Davis, and H.-J. Schek, Eds., Morgan Kaufmann, pp. 590–601.

[94] Sharma, K. D., and Rani, R. Choosing optimal branching factors for k-d-b trees. *Information Systems 10 1* (1985), 127–134.

[95] Sheikholeslami, G., Chatterjee, S., and Zhang, A. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *Proceedings of the 24rd International Conference on Very Large Data Bases* (San Francisco, CA, USA, 1998), VLDB '98, Morgan Kaufmann Publishers Inc., pp. 428–439.

[96] Siddiqi, K., Shokoufandeh, A., Dickinson, S. J., and Zucker, S. W. Shock graphs and shape matching. In *Proceedings of the Sixth International Conference on Computer Vision* (Washington, DC, USA, 1998), ICCV '98, IEEE Computer Society, pp. 222–.

[97] Stonebraker, M., Wong, E., and Kreps, P. the design and implementation of ingres. *ACM Trans. Database Sys. 1* (1976), 189–222.

[98] Tamminen, M. Efficient spatial access to a data base. *Proc. SIGMOD Int. Conf. on the Management for Data* (1982), 200–206.

[99] Wessel, M. On spatial reasoning with description logics-position paper. In *Proceedings of the International Workshop in Description Logics* (Touluse, France, April 2002), I. Horrocks and S. Tessaris, Eds., CEUR Workshop Proceedings, pp. 156–163.

[100] Xu, L., and Oja, E. Randomized Hough transform (RHT): basic mechanisms, algorithms, and computational complexities. *CVGIP: Image Understanding 57*, 2 (Mar. 1993), 131–154.

[101] Xu, X., Ester, M., Kriegel, H.-P., and S, J. A distribution-based clustering algorithm for mining in large spatial databases. pp. 324–331.

[102] Zhang, Z. Iterative point matching for registration of free-form curves and surfaces, 1994.

[103] Zloof, M. Query-by-example: a database language. *IBM Syst. J. 16* (1977), 324–343.