

SEMANTIC DESCRIPTION OF SCENES IN VIRTUAL REALITY ENVIRONMENTS

by

KONTAKIS KONSTANTINOS

Bachelor Honours Degree in Applied Information Technology & Multimedia, Technological  
Educational Institute of Crete, 2011

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF INFORMATICS ENGINEERING

SCHOOL OF APPLIED TECHNOLOGY

TECHNOLOGICAL EDUCATIONAL INSTITUTE OF CRETE

2015

Approved by:

Major Professor  
Dr. Athanasios G. Malamos

**Copyright © 2015**

KONTAKIS KONSTANTINOS

All rights reserved. No part of this material may be reproduced, displayed, modified or distributed without prior written permission from the author or author's institute – except in the cases of noncommercial research and nonprofit education which has to be accompanied by the appropriate citation.

## **Abstract**

Over the past few years, VR environments have been widely adopted in numerous domains as the de facto artificial reality solution for the immersive experience of simulated worlds. Moreover, their practicality was further enhanced with their integration in HTML5-capable browsers via X3DOM framework, while the introduction of Semantic Web gave birth to various technologies for their attribution with semantic metadata. However, all these additional layers of information focused into the sufficient representation of geometric, textural, or ontological concepts, dismissing an efficient spatial representation mechanism. This thesis aims to provide such a mechanism with the proposal of a 3D R-tree data structure for the spatial indexing of X3DOM scenes and the establishment of a computational model for the automated implication of 3D content's spatial relations. Finally, the indexed dataset can also be queried by a set of spatial predicates, laying in this way the foundations for a spatial query language on the Web based on X3D standard and X3DOM framework.

***Keywords:*** R-tree, Semantic Web, Spatial Relations, Spatial Searching, X3D, X3DOM

# Table of Contents

Copyright © 2015 .....	ii
Abstract .....	iii
Table of Contents .....	iv
List of Figures .....	vi
List of Tables .....	vii
Acknowledgements .....	viii
Dedication .....	ix
Preface .....	x
Chapter 1 - Introduction .....	1
Chapter 2 - Motivation .....	3
Chapter 3 - Related work .....	4
Semantic Web solutions .....	4
Leveraging MPEG-7 features .....	6
Designing semantically-rich VR environments .....	8
Dealing semantically with real-world problems .....	9
Contribution of semantics to spatial arrangement .....	10
Chapter 4 - Components .....	13
Extensible 3D (X3D) Graphics .....	13
X3DOM .....	16
JavaScript .....	18
Chapter 5 - Implementation .....	21
The R-tree spatial data structure .....	23
A brief state-of-art in R-trees .....	25
An R-tree for indexing X3DOM scenes .....	28
R-tree operations .....	31
Insertion .....	32
Splitting .....	34
Searching .....	38

Point Query .....	39
Region Query .....	40
k-NN Query .....	41
A computational model for spatial relations .....	45
Topological relations .....	46
Directional relations .....	50
Semantic annotation of spatial relations .....	53
Chapter 6 - Experimental evaluation .....	56
R-tree taxonomy & stretch tests.....	56
Performing spatial queries .....	61
Implication of spatial relations .....	66
Chapter 7 - Conclusion .....	69
Chapter 8 - Future work.....	70
References.....	71
Appendix A - Glossary of Terms.....	77

## List of Figures

Figure 3-1 Annotating an X3D object using MPEG-7 Descriptors .....	7
Figure 3-2 Spatial relations disclosure in DEC-O framework .....	11
Figure 4-1 The tiered architecture of X3D profiles .....	14
Figure 4-2 The current state of X3DOM's fallback model.....	17
Figure 5-1 A simple k-d tree bisection .....	21
Figure 5-2 Octree subdividing a hypothetical cuboid search space.....	22
Figure 5-3 Comparison of R-tree variants overlap ratio .....	26
Figure 5-4 An indicative R-tree taxonomy .....	29
Figure 5-5 Each MBR is an AABB container .....	30
Figure 5-6 Splitting & Adjustment operations on tree structure.....	37
Figure 5-7 The MBR face property .....	42
Figure 5-8 Semantic spatial properties of DEC-O .....	55
Figure 6-1 Various levels of an R-tree instance.....	57
Figure 6-2 Indexing a Shopping Mall for location identification purposes.....	61
Figure 6-3 Estimating location based on a chosen object.....	62
Figure 6-4 Execution steps for a region query .....	63
Figure 6-5 The result set of a region query.....	64
Figure 6-6 Searching for 2-NN candidates with BFS algorithm .....	65
Figure 6-7 Automatic implication of spatial relations between three objects .....	67

## List of Tables

Table 5-1 Traversing R-tree to find out the best leaf node for Insertion .....	33
Table 5-2 Inclusion condition in 3D space between point and rectangular parallelepiped .....	39
Table 5-3 Overlap condition in 3D space between two rectangular parallelepipeds.....	40
Table 5-4 k-NN BFS algorithm pseudocode .....	43
Table 5-5 DE-9IM topological relations.....	47
Table 5-6 Disjoint or Touch condition for rectangular parallelepipeds in 3D space.....	48
Table 5-7 Equal condition for rectangular parallelepipeds in 3D space.....	49
Table 5-8 Contains condition for rectangular parallelepipeds in 3D space.....	49
Table 5-9 Left & Right conditions for rectangular parallelepipeds in 3D space.....	51
Table 5-10 Above & Below conditions for rectangular parallelepipeds in 3D space .....	52
Table 5-11 Over condition for rectangular parallelepipeds in 3D space .....	52
Table 5-12 Front & Behind conditions for rectangular parallelepipeds in 3D space .....	52
Table 6-1 Time measurement pseudocode .....	58
Table 6-2 Time costs for each R-tree operation.....	59

## **Acknowledgements**

At this point, I would like to express my gratitude to Dr. Athanasios G. Malamos, Associate Professor in the Informatics Engineering Dept. of Technological Educational Institute of Crete and Head of Multimedia Content Laboratory, for providing me with the opportunity to pursue this thesis under his valuable supervision. The accomplishment of this work wouldn't be possible without his helpful guidance and contribution. Special thanks also deserves to my close friends and laboratory colleagues, who supported me all the way during this work, even in its hardest of times.



## **Dedication**

I would like to dedicate this thesis to my parents for supporting and encouraging me all these years, being the reason of what I have achieved and become today. Last but not least, to all friends and colleagues who had to pursue a thesis in order to further enhance their research and IT skills, like it happened to me.

## **Preface**

Having worked in the area of semantics before, I have become quite familiar with the current state of art in Semantic Web and virtual reality frameworks. The majority of these implementations focused into the efficient semantic representation of specific aspects from various domains. Amongst them, only a few immature works dealt with the 3D spatial arrangement between the objects of virtual environments. Such a lack is not only thoroughly elucidated in this thesis, but it is also resolved with the provision of a computational model for the implication of spatial relationships in X3DOM framework.

## Chapter 1 - Introduction

In our days, an immeasurable amount of visual information is enclosed in digital form, which is mainly broadcasted through World Wide Web. This kind of information is usually rendered with a virtual reality (VR) environment according to the underlying application's needs. Typical domains of use for such applications range from complex simulations and video games, to realistic representation scenes and interactive learning environments. However, one problem that quickly arises is the quality of the conveyed information, since in the majority of the situations it does not contain any semantic description. Moreover, the current 3D visualization formats come across various problems as concerns this semantic annotation, since their main purpose is the modeling of 3D content rather than the definition of a semantic description scheme. A characteristic example regards the adequate annotation of the objects' geometry, which however comes at the cost of the logical or functional semantics. In order not to suffer such losses, the definition and integration of the semantic information had to be decided before the implementation of the environment itself.

However, the authoring of such environments comes along with a set of restrictions in its interoperability and extensibility, making its future modification difficult if not impossible at all. This lies to the fact that each VR environment has particular requirements regarding its functionality and performance. To fulfill these requirements in the best possible manner, a variety of diverse data models and network protocols are utilized in each occasion. This heterogeneity makes difficult to develop or adapt an existing software system based on each type of environment, since such a thing would not only require the modification of the system's interface, but also radical changes in its backend services.

Previous attempts to resolve the aforementioned issues have already been performed using various technologies. In [1] the MPEG-7 standard has been deployed in order to enhance the querying and navigation processes, while in [2] and [3] the standard itself was extended with various descriptors, in order to efficiently annotate complex X3D scenes. On the other hand, there were other approaches, which proposed platforms either for the efficient indexing and retrieval of XML-based annotated 3D scenes [4], or for the annotation of 3D scene objects with semantic descriptions closely related to their conceptual meaning and functionality [5]. The

majority of these research works explored successfully the semantic conceptualization of the 3D objects in a virtual environment, along with their corresponding functions and relationships. Moreover, they emphasized the main idea behind any semantic representation scheme, which is none other than the efficient semantic annotation of a 3D scene with linguistic predicates. However, all of them failed -or didn't intend- to deliver a spatial representation of the underlying content. Thus, a standard generic tool has to be developed for the description of multimedia content, capable of filtering a VR environment through specific criteria and attribute its contained objects with the appropriate spatial semantics. This study encounters with the challenge of describing a virtual environment and its 3D contents, through the indexing of the latter in a tree data structure for the implication of their relative spatial relations and spatial reasoning purposes. In the upcoming sections is thoroughly discussed the algorithmic approach followed in this work, while experimental results from various developed use cases are explained and evaluated in the last section of this thesis.

## Chapter 2 - Motivation

Today, virtual reality is available to everyone thanks to the exponential advances in Web and its support from various devices and platforms. Such factors broadened the scope of 3D applications, making their publishing and sharing a practical requirement [6]. However, the latter two requirements are difficult to be met, since all these numerous environments are thriving from 3D content that lacks of a high-level description layer. Moreover, each one of them has been authored according to a specific 3D graphic format, which usually comes with an unfriendly web-based interface. So, it is crucial more than ever, the enrichment of these environments with semantic concepts relative to what kind of environment is being presented, what objects are contained, their spatial placement compared to others, etc. The most of research works have already provided a solid foundation for the semantic representation of geometric and textural aspects of these environments, but there is a significant gap in the literature regarding a sufficient spatial reasoning methodology between their 3D contents.

This thesis aims to deal with the above mentioned issues by providing a 3D R-tree data structure –implemented with JavaScript language- for the spatial reasoning and semantic representation of the objects contained into such environments. The spatially indexed VR environments comply with the X3D standard [7], which is a widely used XML-based presentation format for the creation and visualization of interactive 3D content. X3D alone is powerful enough to describe the majority of virtual environments, but it does not specifically define a way to semantically annotate or reuse the objects that compose such environments. Due to this inability, it was deemed necessary the creation of a semantic concepts layer which was able to extract spatial relations between numerous objects in a virtual environment. The spatial reasoning process is coupled with X3DOM framework for presentation and ease-of-use purposes. This spatially semantic description can find appliance to a wide range of domains ranging from the creation of more realistic 3D scenarios, or the execution of advanced queries with proximal and faster content searching. Lastly, the definition of such spatial relationships not only provides complex and richer capabilities to the content providers, but also boosts the virtual potentials of the end-users.

## Chapter 3 - Related work

A lot of research has been conducted since the introduction of Semantic Web [8] about the enhancement of the existing Web information in a machine-understandable way. The majority of these works took advantage of various Semantic Web technologies, like *RDF*, *OWL* and *SPARQL* to semantically describe and retrieve any kind of information. However, because the use of such semantic languages is clearly based on the selection of specific domain concepts and the correlation between these concepts and the underlying information, different routes of defining and exploiting the wide range of semantic knowledge had to be found. Most of them made use of the MPEG-7 standard thanks to its diverse content description and retrieval capabilities, while considerably less resorted to extending a 3D presentation standard or embedding metadata into the objects of the 3D world. Other approaches turned to more advanced -but specific purpose- solutions, like the implementation of platforms for the efficient semantic annotation of patrimony buildings and urban scenes, or the extraction of spatial semantics from real-world scanned environments. The latest and most noteworthy studies in these areas are described in the upcoming chapters, where each one has been classified by the underlying standard being used for the semantic representation of virtual environments. Last but not least, they not only record the current state of art among these technologies, but they also denote the practicality of the implemented frameworks in various domains.

### Semantic Web solutions

In the most cases, VR environments are designed in such a way that favors the human perception capabilities by laying emphasis on presentation and interaction features. However, such technical approaches along with the diversity of data models which are traversed through various network protocols, narrow down the usability of the underlying information from a machine scope. Over the past decade various Semantic Web technologies emerged in order to address this problem, with *RDF* (Resource Description Framework) proving to be the cornerstone of Semantic Web, capable of applying an abstraction layer to the underlying information. With the use of *RDF* terminology we are able not only to attribute semantic information to 3D content, but also guarantee the utilization of such information in a machine-

readable way. The whole procedure is based on the efficient mapping of the information contained into a VR environment, in the form of subject-predicate-object statements. These statements known as RDF triples are capable of describing conceptual and abstract information amongst the objects of this scene. Moreover, thanks to its high-level representation syntax and a variety of serialization formats, RDF also proved quite efficient in the development of application logic systems that deal with protocol messages, which were expressed as common events in order to conceal unnecessary low-level complexity [9]. This work pointed out the potential usability of separating the development of VR environments into high and low-level phases that both make use of an ontology language. Such an approach was followed in [10], where a *DAML+OIL* ontology -a language which was later replaced by OWL- was used as a modeling tool to represent various concepts of a specific domain along with a finite set of relationships between these concepts. This mapping of the domain knowledge to an ontology terminology, led to an explicit distinction of the presented objects and an improved comprehension of the properties that take place into a 3D scene from the scope of the end-user. Ultimately, this high-level ontological representation where concepts and terminology have been borrowed from a specific domain, gave birth to various semantically designed applications. Sometimes, such applications made use of additional standards for the exploitation of the underlying semantics. In [11], the X3D standard was used thanks to its presentation and interaction features as the intermediate interface between the system and the user, while at the same time, all the semantic information of this environment was subsumed in an MPEG-7 file allowing its future utilization.

From the above research works, it is quite evident that regardless of the application's goals the semantic representation of a virtual environment is vividly based on an ontology, which contains the very basics and most commonly used features that can be attributed in such environments such as name, color, size, etc. [12] [13]. These features along with a predefined set of conceptual relationships coming in the form of ontology properties, form a content-oriented semantic model that is capable of sufficiently annotating objects, but unable to describe the VR environments' interactions, communications and behaviors. To resolve such issues, various platforms have been implemented using a wide range of technologies, from hardware-based approaches to the following MPEG-7 standard.

## Leveraging MPEG-7 features

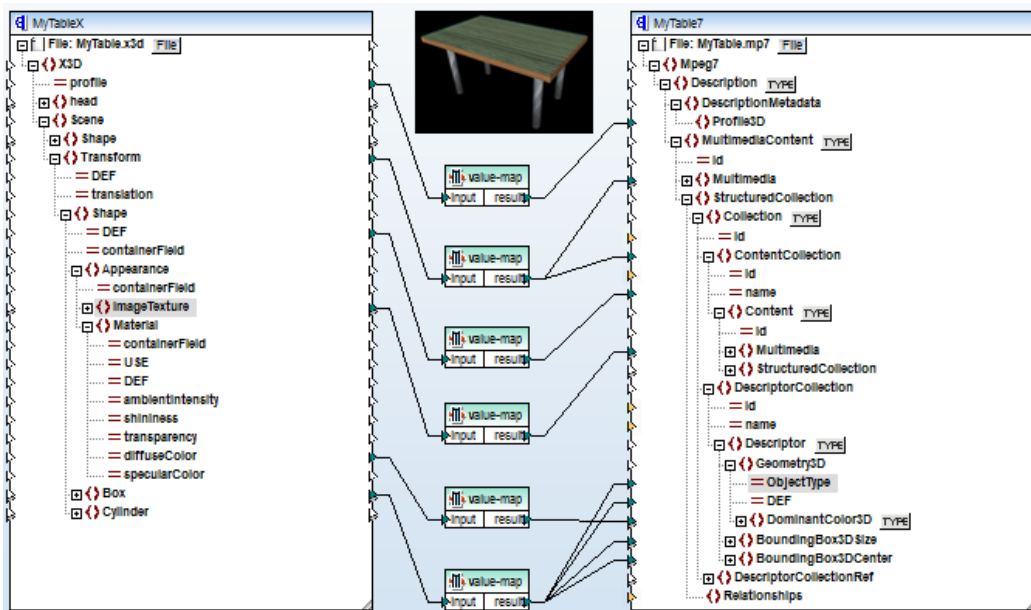
The MPEG-7 standard has been widely accepted as the best multimedia description tool, making it the perfect candidate to annotate audio, image and video elements. However, the first versions of the standard lacked the ability to sufficiently annotate the elements of a 3D scene from a pure semantic scope. In [14], the newly (at that time) presented MPEG-7 was chosen as the appropriate language to semantically annotate X3D, VRML or SVG scenes. Their semantic description was heavily based on the Semantic Entity, Semantic Attribute and Semantic Relation tools provided by MPEG-7 standard, where they were used for the creation and storing of metadata about the spatial relationships between the objects of the scene. Their implementation also involved the translation of the generated MPEG-7 graphs into an interactive visual application, as an alternative visualization solution to the textual nature of MPEG-7 which can become quite difficult to comprehend in complex 2D/3D scenes. Besides that notable attempt, throughout the last years various amendments have been added to the standard in order to guarantee a semantic conceptualization.

A different indexing procedure of a VR environment using MPEG-7 was proposed in [15], where MPEG-7 Descriptors and a Description Definition Language (DDL) were used. Their work adopted the notion of *segment* for the hierarchical indexing and attribution of semantic information. MPEG-7 *MediaLocatorType* was used to structurally localize 3D objects, while *RegionLocatorType* introduced geometric localizations. The link between those two Descriptors was made feasible through 3DSEAM (3D Semantics Annotation Model), a platform which made use of various concepts to define real-world objects, semantic profiles, properties and relations with the use of the before mentioned 3D region/object locators. However, the proposed framework [16] [17] had to surpass a number of obstacles, such as the implementation of an automatic localization and annotation system, and the definition of a query language capable of applying semantic-based queries on the generated instances of this model.

After some years, motivated by the necessity of improving the annotation effectiveness of 3D models and embracing once more MPEG-7 Descriptors, [2] implemented an efficient and complete description mechanism for the semantic annotation of X3D nodes. The proposed annotation scheme dealt exclusively with X3D scenes and it was capable of describing not only the geometrical and appearance characteristics of the 3D content, but also its animation and interactivity features. The description mechanism was based on the addition of several



Descriptors, extending the original MPEG-7 Visual and Metadata Descriptors of the standard and defining a new MPEG-7 Schema Definition for the validation of the description file. The latter file was generated through the employment of an *XSLT* algorithm, capable of automatically transforming any X3D object into its corresponding MPEG-7 description. Fig. 3-1 depicts a simplified version of the algorithmic process that takes place for the annotation of an X3D table.



**Figure 3-1 Annotating an X3D object using MPEG-7 Descriptors**

At first, a *Profile3D* datatype is used to identify the X3D profile being used. Afterwards, the scene is scanned for Transform nodes, where each one is represented by a *ContentCollection* Descriptor. The latter ones retrieve all the information that can be found inside these nodes, like geometric shape, color, material, etc. In the case that a Material node comes with a texture, its path is recorded with the assistance of the *MediaURI* element. In addition to the displayed example, plenty of other Descriptors and elements are provided to annotate complex scenes. Moreover, the extended MPEG-7 framework was used in conjunction with other MPEG standards (MPEG-21, MPEG-4) enhancing the indexing, retrieval and reusability capabilities of an online advertising platform [3]. Finally, this research work still remains in continuous progress aiming to annotate even more X3D nodes, such as *Sound* or *AudioClip* nodes.

## Designing semantically-rich VR environments

Besides the MPEG-7 area of solutions, there were also a few remarkable works with mixed results on the direct manipulation of the 3D content found in a VR environment. One of the first works in this area [18] proposed a new X3D profile dedicated to the description of interaction techniques and the design issues met in such environments. The proposed profile named *InTml* -which stands for Interaction Technique Markup Language- composed of various components which guaranteed the co-existence of developer and designer roles under the same application. However, the presented description language gave birth to serious defects, such as underperformance during the scene's navigation and the software's inability to detect non-pc devices, affairs that come in conflict with the expeditious and interoperability traits established from the X3D standard.

Staying in design territory but extending the applicability of semantics to various 3D presentation formats, [19] implemented an annotation model for the enhancement of communication and knowledge management between the developers and end-users. The presented model was composed from three separate but interconnected components which were respectively responsible for the presentation format of the annotation, the placement of the annotation in VR environment, and the storing of additional data (i.e. metadata) about the annotation. Such annotations were able to be placed directly in any object of the scene, but their indexing procedure was based on an ontology unable to cover the domain knowledge sufficiently, resulting to the continuously adjustment of the presented model each time a new visual concept had to be added. Taking into account this deficiency and using a novel UML modeling approach, MASCARET Framework [20] went one step further in the semantic representation of 3D content by covering system-oriented semantics. Unlike the content-oriented solutions provided by the Semantic Web stack of technologies, MASCARET focused into the creation of intelligent semantically environments comprising not only the classical domain knowledge, but also the available set of interactions along with the relative behavior of participant entities. The framework developed a number of scenarios, which emphasized the simulation of human activities and interaction tasks in a typical VR environment. The same scenarios were later used for validation purposes, providing quite satisfactory results in the proposed semantic modeling methodology.

## Dealing semantically with real-world problems

An alternative path to the above mentioned multimedia protocol-driven solutions lies to the use of advanced AI techniques for the semantic representation of specific 3D content. The semantic description and 3D representation of patrimony buildings [21] showed that it was deemed necessary the met of two conditions, in order to mine all the necessary information without losing any semantics inscribed on architectural shapes. The first condition relates to an immature but realistic 3D representation of the patrimony building based on the existing architectural patterns, while the second one concerns the deduction of the appropriate semantic information stemming from this 3D model. The combination of these two conditions was able to formalize the architectural knowledge within a finite number of architectural objects, where each one was qualified by various domain concepts. In the end, these objects composed a VR environment enriched with architectural semantic information, accessible via the GUI of a utilitarian platform. After a few years and changing the domain of interest from heritage buildings to urban scenes, it was presented a system [22] capable of adding semantics for the efficient skyline and windows detection. Their purpose was to imitate the human brain perspective capabilities, which lay into the fact that humans are able to immediately dissociate similar or intersecting objects even when their point of view suffers from distortion or obstacles. Their system was based on the combination of the *FIT3D* Matlab toolbox and a *skyline detection algorithm*, producing satisfactory results after being tested on various 3D urban scenes. Even though the derived semantics were quite limited compared to the human perception abilities, it was proved a valuable start towards the distinction of intersecting objects in real-world environments.

On the other hand, the rapid development of hardware over the last years led to more advanced and interactive approaches as concerned the mining of semantic information from 3D environments. In [23] was firstly presented the idea of a semantic net which contains and implements general knowledge of indoor environments. However, even though that their approach was able to represent the majority of indoor environments, they dealt with the 3D scanned representations of real world environments. The initial feature extraction was done with a 3D laser scanner using a combination of *ICP* algorithm and *RANSAC* approach -and then- nodes of the semantic net represent the entities of these environment, which are accompanied with relationships and constraints between them. Despite the fact that their semantic concepts

were well-defined and well-structured, this semantic interpretation was purely boosted through the 3D analysis of the extracted features. Moreover, ICP implementations tend to find appliance in shape registration problems [24], where the former is combined with various types of data structures for the registration of nearest neighbor queries.

After almost a decade and following the same pattern, it was developed a system capable of modeling real world indoor environments with the assistance of an RGBD camera [25]. At first, it was taking place the segmentation and labeling of the captured images, which differentiated the models that this indoor environment was composed of. Afterwards, a 3D shape matching algorithm replaced each segmented region with the most identical 3D model found in a database, ultimately leading to a 3D representation of the captured real world environment. However, the recognition accuracy of the system was clearly depending on the quality of the captured depth data, and the final reconstructed environment lacked of some significant semantic features (such as the geometrical information of the objects).

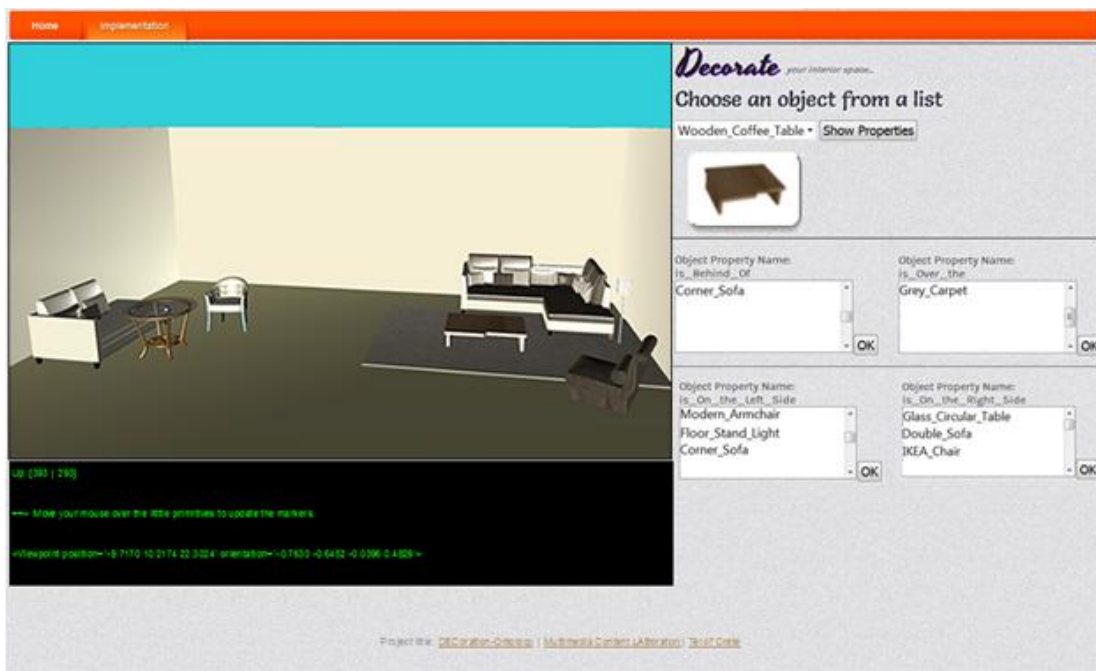
### **Contribution of semantics to spatial arrangement**

As we can see, the majority of research works focused into the rendition of a VR environment and its objects with semantic terminology, paying little attention to an efficient space interpretation scheme. A recent work [26] in this area adopted various spatial relations in order to qualitatively represent a 3D space. A spatial relation is any relation that specifies how an object is located into the space in relation to another object according to a topological, directional, or distance relation. A finite set of spatial relations drawn from these three types of relations defined two distinct modes of semantic spatial knowledge:

- ❖ The topological spatial knowledge composed of *on*, *in*, *at*, *near* and *surrounded* spatial relations
- ❖ The view-point dependent spatial knowledge composed of *right*, *left*, *between*, *in front of*, *behind*, *above* and *below* spatial relations

The selection and demarcation of these spatial relations were based on the human reasoning capabilities, aiming at a faster and more accurate settlement of reasoning procedures (such as guiding a visual object search). However, some of the chosen topological relations come in contrast to the space relation theory, which states that any topological relation should be invariant to rotation, translation and scaling transformation [27]. Keeping in mind the latter one

and taking into consideration that not all types of semantic frameworks have to categorize their corresponding spatial relations, *DEC-O* [28] [29], an ontological framework for interior decoration applied a more generic spatial arrangement. In this work, the semantic spatial knowledge of the implemented OWL-DL ontology was based on the definition of various object properties reflecting a wide range of spatial relations, without being necessary their grouping to one of the three most commonly used spatial relations types. A small subset of these spatial relations is presented in the following Fig 3-2, where each one corresponds to a different OWL object property of the instantiated interior room-space. The room-space is cooperating with X3DOM framework to enhance the presentation and interactivity capabilities of the application, while Apache Jena framework is used to traverse the underlying OWL ontology. In this example, the end-user of the application has selected the desired X3D object, which in turn returns its spatial relationships with the rest of the objects. Doing so, it was made feasible a fast and reliable mapping of the location of objects that coexist in the 3D scene. However, this approach lacked the ability to automatically attribute space annotations to the existing objects or to efficiently represent the correlation of these objects compared to the 3D space itself. Its primary target was the implementation of an ontological framework for the annotation of interior room-spaces, rather than providing an automated spatial reasoning mechanism.



**Figure 3-2 Spatial relations disclosure in DEC-O framework**

Among others, this study takes into consideration the lack of the latter feature in DEC-O's annotation mechanism, implementing and providing an efficiently methodology for the deduction of such spatial relationships, which is thoroughly described in the following chapters of this thesis. Last but not least, the presented approach is not only independent of the underlying platform being used, but it can be also easily integrated in various systems to enhance their corresponding automated capabilities, like supplementing the spatial relations of DEC-O without any input or further action from the end-user of the application.

## **Chapter 4 - Components**

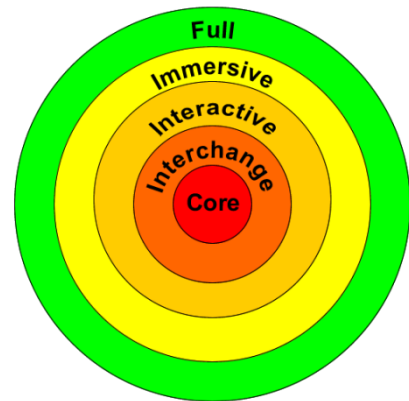
It is common knowledge that the VR representations of real world environments are far more attractive and realistic than their corresponding 2D. This notion has been further acknowledged over the last years through the elevation of various multimedia software products and the increased rendering of 3D content, thanks to the graphics hardware acceleration. Today, X3D standard provides advanced virtual and augmented reality capabilities that cover a wide range of domains, leaving far behind other 3D visualization technologies. It has been publicly acclaimed as the mainstream visualization format on the Web, while the last years its usability was further boosted due to its adoption from the X3DOM framework. X3DOM makes possible the publishing and manipulation of X3D scenes as DOM elements in any HTML5-capable browser, turning X3D standard also into an interchange format for the declaration of 3D interactive content on the same medium. The rendering process is supported by various back ends, including WebGL, which is the latest tech trend for the plugin-less rendering of 3D content assisted only by the graphics processing unit of the system. The smooth cooperation between these standards and their interoperability amongst various operating systems and devices (desktop computers, smartphones, etc.) are guaranteed through the usage of JavaScript as the main programming language. Its platform independence, ease of use and scalability features do not only meet the requirements set by this study, but they also enhance the implemented algorithm's future potentialities. A short introduction to the capabilities of the aforementioned technologies is described in the following subchapters.

### **Extensible 3D (X3D) Graphics**

Today, X3D is the most widely used standard for the presentation of 3D content on the web, defining a runtime environment and a delivery mechanism encoded usually in XML format and represented as an n-ary tree. The architecture of the standard complies with various ISO standards, providing three different encoding options, ensuring its applicability to a wide range of areas and supporting every browser on the Web. One of the most remarkable characteristics of its architecture lies to the existence of variform profiles, where each one defines explicit functionalities for closely related target groups. This architectural layout enabled not only the

rapid expansion of X3D to mobile devices such as smartphones and tablets, but also gave the opportunity to software developers to choose amongst a subset of the implemented functionalities, absolving them from the necessity of conforming with the entire specification sheet of the standard. The majority of these profiles [7] along with a brief description of their functionality and possible areas of use are briefly described below, accompanied by an illustration of the tiered architecture in Fig. 4-1:

- ❖ Core is the profile comprised of the absolutely minimum required components that compose any X3D scene. However, because of its extremely minimal nature it is rarely met in applications.
- ❖ Interchange is one of the most widely used profiles of the X3D standard. It supports a variety of features -such as geometry, textures, lighting and animation- for the rendition of geometric models, while at the same time its applicability is a trivial procedure since it does not define any runtime rendering model.
- ❖ CAD Interchange contains the majority of Interchange profile, plus a few additions targeting at the efficient compilation and integration of CAD application's data as an interactive X3D application.
- ❖ Interactive is a slightly component-richer X3D profile compared to the Interchange one, specializing at the interaction of the end-user with the 3D environment via the provision of advanced lighting, motion detection and navigation nodes.
- ❖ MPEG4-Interactive combines the capabilities of Interactive profile with the standards set by MPEG-4 for the efficient usage of X3D environments in broadcast and mobile applications.
- ❖ Immersive profile not only implements the same components as the Interactive profile, but also offers several features, such as the audio support, weather effects nodes and script functionality (X3D-EcmaScript). These features maximize the total immersion and effectiveness of simulation and gaming applications, making this type of profile to continuously gaining ground amongst the others.
- ❖ Full profile contains the entire set of components determined by the X3D specification. This profile extends the Immersive profile with four different components, the Distributed



**Figure 4-1 The tiered architecture of X3D profiles**



Interactive Simulation (DIS), Humanoid Animation (H-Anim), Non-Uniform Rational B-spline Surfaces (NURBS) and GeoSpatial. These components find appliance to very specific and complex domains, such as the synchronized 3D simulations, character animations, medical implementations and GIS applications, respectively.

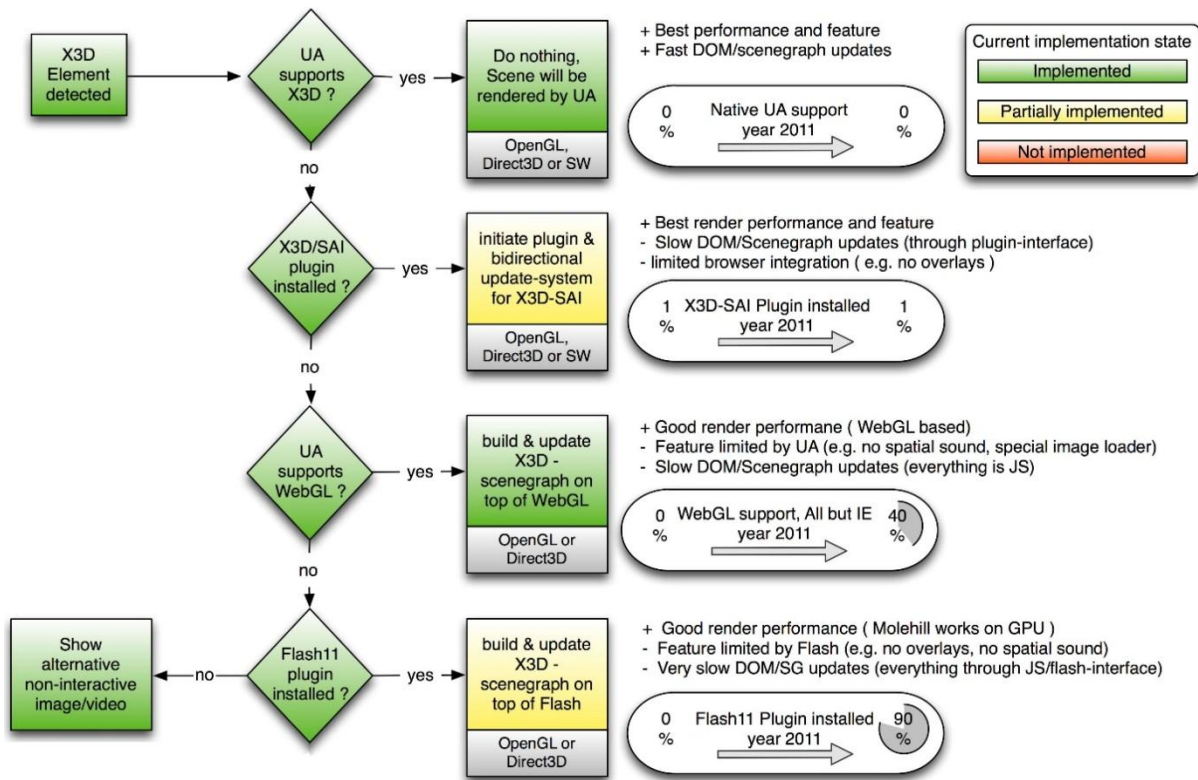
The latest stable release of the X3D standard is the version 3.3 enumerating 41 components that constitute the aforementioned profiles. Each component provides a set of nodes with similar functionalities, designating an articulation of different levels based on the characteristics of these nodes that range from the definition of geometric primitives and their corresponding transformations, up to the settlement of alternate content and multi-level representation. Thereby, plain applications are free to use a low-level profile without serious trade-off on the performance of the underlying 3D content, while demanding applications can chose according to their presentation and interaction needs, amongst the available high-level profiles. On the other hand, regardless of the chosen profile, X3D offers a flexible mechanism for the inclusion of additional data about the relative X3D scene. These data are enclosed in metatags, which are in turn encapsulated into the unique head tag of the XML serialization. The X3D specification provides a finite number of metatags that can sufficiently annotate any scene with the most commonly used information, such as the name of the creator, date created, title of the environment, license file, etc. Even when these tags are deemed inadequate, the standard always allows to the user to define his own tags, guaranteeing in that way a decent metadata description scheme of the X3D scene. However, its capabilities do not stop here since the standard also defines SAI (Scene Access Interface), an abstract API responsible for the cooperation of X3D with different technologies. *SAI* is a programming interface used for the establishment of connection between the X3D language and an external programming language, like Java or JavaScript. The utilization of such languages provides powerful interaction features and improved behavior on the scene's elements, while any communication that takes place is achieved through the exchange of specific events among the participating languages.

All of the above point out the X3D standard's rich and polymorphic nature, making it the ideal representation format for a wide range of domains from engineering and scientific to architecture, multimedia and entertainment. Moreover, the X3D standard has already made great strides compared to its predecessor VRML, being adopted by various XML-based languages (HTML5, XHTML, SVG, etc.) and the last years strives to become the 3D standard for the World

Wide Web. However, its 3D scenes tend to provide sufficient information only on the geometric features of its contents, since the standard itself does not provide any medium to incorporate any semantic information. With the passage of time this inability became quite troublesome in the majority of the X3D scenes, since not only primary goals of the standard (like the precise and rich presentation) were left behind, but also the idea of a Semantic Web made inevitable the use of such information. The attribution of semantics in X3D scenes enhances the overall representation capabilities of the any application, provides advanced identification techniques amongst its objects and guarantees its reuse in a more sophisticated way (fidelity applications that need improved degree of accuracy and reduced search time).

## **X3DOM**

X3DOM is an HTML5/X3D integration model that makes feasible the publishing and updating of declarative X3D content into any HTML DOM tree [30]. This model has been implemented with an open-sourced architecture which is available to the public as a JavaScript framework. The most distinctive feature of its architecture lies to the definition of a modular backup approach for the rendering of the 3D content, which is ultimately supported by a variety of back ends, like native, X3D plugin, WebGL and Flash. Amongst them, WebGL stands out since it allows the rendering of interactive 3D content without the need of installing any plugin for the majority of the latest desktop and mobile browsers. Moreover, its rendering capabilities include physics and shading support, while the overall procedure is accelerated with the Graphical Processing Unit. WebGL and the rest of back ends may vary in functionality and performance terms, but each one of them is available during X3DOM's runtime. By doing so, the 3D content being presented in the specific browser is the one that sets the necessary requirements, which in turn lead to the selection of the appropriate backend for the rendering of this content. On top of that, the subjected architecture is able to undergo the integration of additional back ends, filling in possible needs that may arise in the near future. In the following Fig. 4-2 is displayed this intermediate fallback model provided by X3DOM framework.



**Figure 4-2 The current state of X3DOM's fallback model**

As concerns the 3D declarative content, X3D language was chosen as the appropriate one, because it is a mature ISO standard coming with an XML encoding similar to that of XHTML's. Based on this feature, X3DOM defined an integration methodology for the declaration of these X3D scenes in any XHTML document and a mechanism for the direct live manipulation and updating of the underlying DOM tree. In other words, X3DOM serves as a gap cover between the X3D language and the participating web specifications. However, in order to sufficiently integrate an X3D scene in the DOM tree of a XHTML document, X3DOM had to modify the Interchange profile of the X3D standard. The available nodes were increased by the addition of various higher-profiled nodes, like the *Inline*, *Switch* and *LOD* nodes, while any scripting capabilities are clearly left to the DOM/HTML side by eliminating the *Script* and the declaration of prototyped node types. Although this profile fulfills the requirements met in a wide range of applications, X3DOM provided the means for a richer and more realistic

<sup>1</sup> source: <http://www.x3dom.org/wp-content/uploads/2009/10/x3dom-fallback-Release-1.2.png>

presentation with the introduction of the Texture and Mesh nodes. The first one makes use of specific HTML tags (img, video and canvas) for the integration of their content into specific X3D nodes, while the second one can take advantage of the latest shaders found today. Besides these characteristics, software developers are free to use the native JavaScript methods or any JavaScript library they may like for the addition, removal or update of the X3D nodes and their relative properties. Even though that X3DOM supports all the methods defined in the HTML DOM specification, there are a number of drawbacks concerning its integration model. Amongst them, a minor drawback is the appliance of CSS language on the HTML canvas element alone, since CSS modules are hardly usable in the elements of an X3D scene. On the other hand, a more notable drawback resides in the utilization of the X3D format as its unique 3D presentation format, bringing forth the need of converting every 3D content into this type of file format. Finally, the most worth mentioning drawback lies to the lack of an efficient progressive transmission mechanism [31], since the only one implemented is capable of receiving batches of geometry data over multiple HTTP requests which can lead to network congestion and low performance issues in case of immense 3D content.

Summarizing, X3DOM not only incorporates 3D content on the Web without the use of plugins but this integration also takes advantage of existing Web standards instead of defining new ones. Its innovation lies to the concatenation of the HTML5 and X3D standards through the provision of a robust programming interface and a flexible fallback model. At the same time, its versatile architecture allows not only the adoption of the X3D standard from the majority of browsers, but also guarantees the integration of future amendments to this model. With features like these, X3DOM is definitely one of a kind 3D visualization technology which can be applied to a variety of areas desiring an open source plugin-less 3D content presentation solution.

## **JavaScript**

JavaScript is a structured object-oriented programming language that conforms to the ECMAScript Language Specification, inheriting from it powerful scripting capabilities and making JavaScript a constantly evolving ISO standard. The interpretation and execution of the language scripts is done with the assistance of an independent JavaScript interpreter or engine, which is usually contained into the relative web application, browser or plugin. The latest version of the language (1.8.5) introduced new functions, a new object (Proxy), strict mode support and

it was aligned fully with the 5.1 edition of the before mentioned specification. This rapid and widespread adoption of JavaScript forced it to extend the core objects and elements vocabulary [32], in order to sufficiently provide support for client and server side scripting on the web:

- ❖ Client-side scripting provides advanced capabilities for the manipulation of web pages, enabling in that way a dynamically changing content depending on a set of environment variables, like the user's interactions, system conditions, etc. The authoring of client-side JavaScript (CSJS) is assisted by the definition of additional objects and event handlers compared to the core language specification. Typical examples are the Window object which represents the browser's window and various mouse events that indicate special user actions (i.e. *MouseUp* event indicates that a mouse button has been released). This type of scripting occupies the largest portion of use and is usually met on browser implementations where the asynchronous communication is a main asset.
- ❖ Server-side scripting involves operations that are performed by the server in order to lighten the workload of client and/or shelter sensitive information. The employment of server-side JavaScript (SSJS) is achieved through the extension of the core language with various functions, classes and objects, like the write, Connection and database elements respectively. One of the most commonly used scripts deals with the communication attainment of an application with a database for storage and retrieval purposes, while more complex implementations provide runtime environments for the development of games or applications.

In addition to client and server side scripting, there are numerous implementations that define their own JavaScript engines which play the role of an embedding scripting language or a distinct application platform. Even though such implementations make use of an exclusive object-oriented interface, their basic set of objects and elements is borrowed from the JavaScript's core. Typical applications of this kind are met into Adobe Systems products, where in Adobe CS scripting is available with the use of JavaScript language and Adobe Flash works with a dialect of ECMAScript as its main programming language, known as ActionScript. However, the last years JavaScript surpassed the software barrier and geared with microcontrollers, serving as an alternative solution to the reliable and power-efficient control of hardware in embedded devices.

However, JavaScript's nature comes with a number of serious security vulnerabilities that have to be taken into account by any web application or browser that deals with the authoring and execution of JavaScript code. Amongst them, the two most commonly used exploits are the cross-site scripting which involves the injection of malicious script on end-user's system to steal his personal data, and the cross-site request forgery, which corresponds to the execution of unauthorized commands from a trusted web page or application. JavaScript addresses such exposures incorporating a couple of security mechanisms, like the same-origin policy and virtual sandbox environment. The first one prevents the execution of scripts that provide access to data between pages that do not reside under the same protocol, port and host combination, while the second one sets up a virtual environment for the execution of scripts having limited access to hardware and network resources. Besides the security issues, there is also quite limited support from the majority of the existing JavaScript engines as concerns their compliance with the latest JavaScript version. This state of affairs compels programmers into taking special precautions during the software development process, by testing and validating the underlying JavaScript code on multiple environments (i.e. amongst the varied versions of browsers) through the utilization of the relative script debugger.

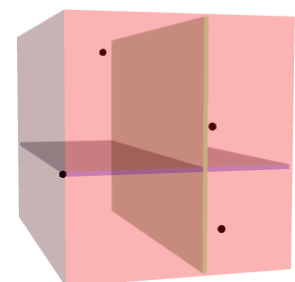
Today, JavaScript is being used as a general purpose programming language to a variety of domains, from web-based implementations and web browsers to electronic documents and standalone applications. It is considered to be the scripting language of World Wide Web thanks to its dynamic cross-platform capabilities, while at the same time, its unperceived presence in almost every computer transfuses a lightweight and reliable character into the language itself.

## Chapter 5 - Implementation

In this chapter it will be thoroughly described the implementation process of each individual part, which has been assigned a particular task for the semantic representation of VR environments. Even though that the implementation approach followed is independent of the underlying platform, any 3D scene that desires to be indexed must comply with the X3D standard and X3DOM framework. In this way, a common frame of reference is used to not only spatially query the indexed objects, but to also record any spatial relationships that take place between these objects. These parts come with a set of key objectives that can be divided into the implementation of a spatial indexing data structure and a sample OWL ontology for the translation of these relationships into semantic concepts. At first takes place an introduction to the most commonly used hierarchical representation types for the spatial annotation of a 3D virtual environment, which are none other than the Spatial Partitioning and the Bounding Volume Hierarchy.

A *Spatial Partitioning* data structure continuously sections a 3D space into distinct regions which are used to convey one or more objects. The splitting direction and the number of produced regions is depending on the segmentation methodology being used, while the recursive subdivisions terminate when certain criteria are satisfied. Even though that both of these conditions differ from one data structure to another, the generic pattern of the hierarchical representation remains similar to either. Today, plenty of data structures are based on spatial decomposition solutions, but k-d trees and Octrees are the two most widely known and used out there:

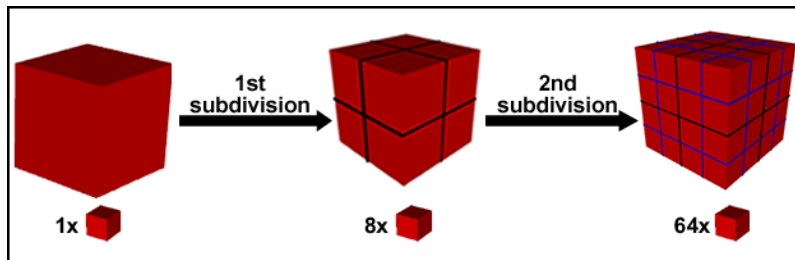
- ❖ *k-d tree* is a widely used BST space partitioning algorithm capable of bisecting a space into two separate parts, where each one contains half of the dimensional points existing into the original space. The search space is split along a specific axis each time and this procedure keeps repeating until every leaf node of the tree contains only one point of the primary space. The dimensional points can be queried with the assistance of pre-order or level-order search algorithms, while their query time is heavily depending on the distribution of the points in the



**Figure 5-1 A simple k-d tree bisection**

search space. In Fig. 5-1 to the right, it is displayed a 3D k-d tree which bisects an area of four data points. At first, it takes place a split on axis X, creating in that way two distinct areas with two data points each one. Afterwards, a second -and final- split along axis Y creates four leaf nodes containing a single point each one.

- ❖ *Octree* is a tree data structure which consecutively partitions the search space into octants in order to enhance common tree operations. Octrees make use of point or matrix region techniques to subdivide the space and allocate the corresponding 3D points inside the octants. Their ease of construction and update made them widely accepted in dynamic 3D space problems like collision detection and range search problems. Some of the most advanced octree implementations can be seen in 3D game industry, enhancing collision detection accuracy between objects thanks to the multiple-boxes approach provided by this data structure. However, octrees lack of an efficient mechanism for the manipulation of static search spaces, while at the same time, they tend to subdivide the given space based on a single point each time.



**Figure 5-2 Octree subdividing a hypothetical cuboid search space**

The above mentioned data structures primarily aim at the efficient clustering of 3D space, leaving in second place a satisfactory indexing mechanism for virtual environment's objects. Despite the fact that this kind of clustering guarantees a sufficient query performance for static datasets, interactive environments with deformable objects seriously suffer [33] from the lack of such a mechanism. So, applications that wish to resolve these deficiencies make use of the following hierarchical representation type instead.

A *Bounding Volume Hierarchy*, is a collection of nodes where each node is a data structure composed of a bounding volume and a list of node pointers. Any leaf node of the tree makes use of its bounding volume to enclose a different geometric object, while at the same time, it points to its parent node and keeps track of the recorded object's location (which is



usually stored in a database system). On the contrary, any internal node points to its parent node and a set of children nodes, while its bounding volume perfectly envelops the total area occupied by the bounding volumes of these children. This hierarchical clustering of volumes is carried on until an orphaned node is reached, which indicates that it is the root of the tree structure and its bounding volume has to enclose the entire set of geometric volumes under a recursively mode. This kind of organization is usually met in physics and graphics domains, while the majority of its implementations make use of axis-aligned bounding boxes (AABB) and spheres, or oriented bounding boxes (OBB).

For the purposes of this work, an AABB tree data structure has been compounded to efficiently annotate the spatial characteristics of a virtual environment's objects. Its design was entirely based on the *R-tree* spatial structure, which is a hybrid space partitioning solution that borrows concepts from both of the afore-mentioned hierarchical representation types. The implemented R-tree algorithm defines a fixed number of node entries and pointers, which are used to subdivide the 3D space into hierarchically nested set of nodes. Each node is represented by a bounding box crafted in such a way as to reduce its corresponding spatial redundancy. The leaf nodes designate a cluster of objects, while internal nodes tend to cluster particular parts of the search space. This hierarchical clustering of nodes aids to the efficient management of the object-subdivided search space, since a few only simple mathematical calculations are enough to decide for the usefulness of an entire cluster of 3D space.

### **The R-tree spatial data structure**

R-tree's roots are found back in '70s due to their origination from the B-tree data structure [34], where many concepts of the latter were left intact and adopted in the former. At that time, many variants of B-trees were brought to life in order to sufficiently deal with the increased need for storing in, or retrieving from RDBMS large datasets. However, this data structure was unable to provide an efficient mechanism for the indexing of multidimensional datasets. A few years later, a counterpart solution for two-dimensional applications was proposed by Guttman in [35], a spatial index structure capable of indexing, removing and retrieving thousands of spatial data. The relative algorithms for these operations partition each time a specific only subset of the primary space, forming in that way many rectangular regions which

are ultimately represented as tree nodes. This data structure is characterized as R-tree due to the definition of such regions of rectangles.

R-tree can be seen as a height balanced tree data structure that consists of a set of connected but acyclic nodes. These nodes contain a predefined range of entries, which can be either more nodes or indexed records. In the first occasion, these nodes are known as internal nodes, while in the second one are known as leaf nodes. Based on this basic tree terminology, any data structure that wishes to be considered as a valid R-tree implementation has to at least comply with the following set of properties:

- ❖  $M$  is the maximum number of entries that any node can contain
- ❖  $m$  is the lowest number of entries that any node can contain
- ❖ Any node contains between  $m$  and  $M$  entries, unless it is R-tree's root
- ❖ If the root node is an internal node, then it must point to at least two other nodes
- ❖ All leaves of the R-tree are piled all together at its lowest possible level

Each node of the tree is represented by a rectangular area, known as minimum bounding rectangle or minimum bounding region (MBR). The dimensions of this area are depending on two factors, the size and the placement of its entries in the search space. Node's entries are accessed one by one to retrieve their corresponding size, which is used to calculate the size of the parent area by aggregating these individual sizes. At the same moment, entries' coordinates are also retrieved to record their lowest and highest values, which are in turn used to properly collocate this area. Thereby, any node's MBR totally encloses its children, while each child's MBR totally encloses node's grandchildren. This nesting procedure keeps going on until a leaf node is reached, where its children contain the actual spatial data and do not point to more nodes. Such spatial data are consecutively indexed to the smallest sized leaf node and tend to represent an explicit point or various geometric shapes. This organized structure is independent from the distribution of the spatial data in the correlated application, even in the case of a sporadically distributed dataset, where changing the maximum and lowest number of allowed entries can reduce the size of MBRs and the overlaps between them. As an additional consequence of such changes, the execution time of spatial queries can improve drastically, since their functionality is closely related to the number of overlaps.

Most of the time, an R-algorithm will create many overlaps between the entries which have been indexed during the Insertion and Splitting operations on the dataset. Such overlaps are

represented in a hierarchically structured tree, where the MBR of a child node is partially or totally covered by the MBR of more than one parent. However, these duplicate entries are eliminated by storing the child record to the least enlarged parent. In this way, not only the space utilization remains in high levels -it has been estimated to be at least 50%- but the Insertion and Splitting algorithms are also easier to authored and maintained. Moreover, R-tree's carefully design and open architecture allows the acceleration of spatial queries by skipping nonessential subtrees of the search space during the Searching operation. At the same time, they are capable of improving queries accuracy by supporting several distance metrics according to the setup of the tested application. So, the appropriate pathfinding algorithm for multi-angle indexed objects is definitely the *Euclidean* distance, while the *Manhattan* distance would be preferred by quadrangular records. Finally, even a *Chebyshev* distance metric could be implemented in order to be applied on Moore neighboring datasets.

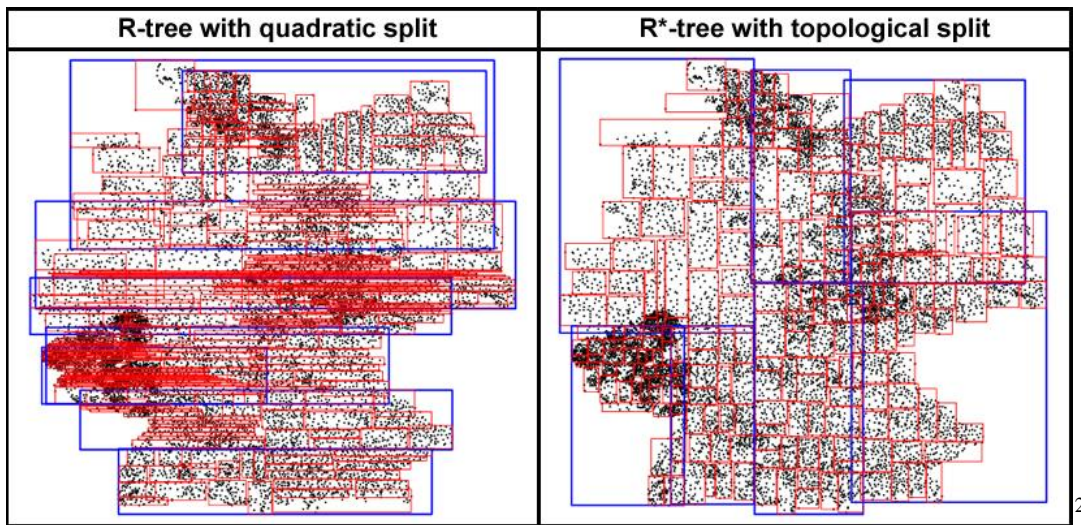
The advantages derived from such traits gave birth to numerous R-tree variants, where each one comes with special characteristics in order to satisfy the requirements set in various applications [36]. Even up to these days, R-tree implementations are being silently used in the background of both theoretical and technical domains, as their main data structure for the indexing of multi-dimensional datasets. In this work, a slightly modified version of the original R-tree data structure has been developed to deal with the indexing of X3D objects through the X3DOM framework and proceed to their spatial retrieval for future use.

### ***A brief state-of-art in R-trees***

R-trees have evolved through the passage of time, bringing forth many variants of the original proposal. All of them aimed to achieve optimality on various aspects of the data structure, like lower insertion cost and better query performance, or guarantee its applicability on specific areas of interest. A detailed report on the latest and most widely used R-tree variants is described in [37], where it has been also defined a classification system that takes place according to each variant's implementation and scope. Following the heels of this fission, there are variants that consist of slight modifications of the R-tree's construction methodology, hybrid variants that take advantage of other index structures and partially apply them into R-tree, and extended variants of R-trees which are used in specific domains by incorporating extra information and richer features. In the upcoming paragraph, the most notable R-tree variant from

each category is shortly described, in order to highlight the supple structure of R-trees and their usability in various areas.

The most widely known R-tree variant is none other than the *R\*-tree*, which makes use of advanced heuristic strategies for the insertion and splitting of spatial information [38]. Its novelty lies to the minimization of area coverage and overlapping MBRs, composing in that way a more rectangular R-tree structure. This is feasible thanks to its reinsertion algorithm, which first tries to find out the fittest node to place a new entry, instead of immediately splitting a leaf node and reassign its overflowing entries. Even in cases where a Splitting operation is deemed necessary, *R\*-tree* will perform this split with various topological variables, like the node's axis and perimeter values. In Fig. 5-3 has been deployed an R-tree instance to index a large dataset of differential points, which can be possibly spatially accessed for future use. In the image on the left side is displayed the generated R-tree structure, which was relied on the Quadratic algorithm [35] for the splitting of its nodes. Conversely, in the image on the right side is shown an *R\*-tree* structure which has been created with the assistance of a topological split algorithm. It can be easily perceived that the overlapping MBRs in the second image are much less compared to the first.



**Figure 5-3 Comparison of R-tree variants overlap ratio**

<sup>2</sup> source: <https://upload.wikimedia.org/wikipedia/commons/0/0e/Zipcodes-Germany-GuttmanRTTree.svg> & <https://upload.wikimedia.org/wikipedia/commons/c/c7/Zipcodes-Germany-RStarTree.svg>

The most obvious and valuable aftereffect of the aforementioned optimizations lies to the improvement of the query performance, despite the fact that its final structure resembles a typical height-balanced R-tree. However, all these techniques not only increase the insertion complexity, but they also introduce a negative impact on the total complexity and maintenance of the underlying algorithms. Because of these traits, it must be carefully investigated the possibility of adopting (or developing) a lighter R-tree variant, for applications that do not have to make use of such special characteristics.

Regarding hybrid variants, *R k-d tree* [39] is particularly interesting since it applies methodologies that take into account both the spatial data and the space partitioning. At first, the search space is partitioned using a slightly modified k-d tree algorithm capable of supporting overlaps between distinct partitions. Every time an intersecting partition is detected, a finite set of bounding rectangles is utilized to represent any overlaps. In this way, the initial BST structure is further enhanced with R-tree's insertion and deletion techniques, acting as a middle ground for a wide range of applications. On the other hand, *DR-tree* [40] is an extension to the modal R-tree algorithm, which comes with the particularity of storing application specific information to hasten queries performance. Such information is attributed with one of the four cardinal directions, which ultimately form additional entries to each internal node of the tree. The regions represented by these child nodes are used during k-NN distance calculations in order to eschew needless computational burden. The only drawback of this approach lies to the low space utilization, which results from the addition of these four cardinal pointing nodes.

Taken into consideration the research outcomes of the above mentioned variants, the R-tree which has been implemented for the purposes of this work comes with a special set of features. The Splitting operation is based on the Quadratic algorithm, which remains a fast and reliable solution to cope with the needs of VR environments on the Web. Even though that R\*-tree has very attractive properties, it would be an overkill to set up and run it under these circumstances. Moreover, it was not deemed necessary to massively change the data structure presented in the original version of R-tree. Only slight modifications to support X3D content and to secure cooperation with X3DOM framework were carried out. Lastly, a number of spatial properties has been authored to enable an efficient semantic representation of the search space, which has been integrated in such a way that any node and its entries are attributed with these semantic concepts.

### *An R-tree for indexing X3DOM scenes*

The original R-tree data structure and the majority of its variants were developed to deal with various 2D applications of specific domains. In the following subchapters however, it will be presented a novel R-tree structure which has been designed to efficiently index 3D virtual environments. The indexed records of these environments can be spatially queried at a later time, while the entire procedure is independent of the environment's origination domain. The only constraint that is imposed on the inputted dataset, lies to the inability of the underlying algorithms to quickly output moving objects, when the latter ones exceed a few hundreds.

The generic data layout of the implemented R-tree is based on the same key features that comprise any R-tree variant. Such features include the set of properties which was reported in the earlier subchapter "*The R-tree spatial data structure*" and a hierarchically organized structure of *log<sub>n</sub>* height. This structure dissociates the usage of its entries depending on which type of node they are located. So, the entries of a leaf node are represented by an array of spatial objects, where each one is attributed with an *id* value and a bounding container for the stigmatization of its boundaries into the 3D space. These id values point to the actual X3D objects of a virtual environment, which have been chosen beforehand by the user as the desired dataset to be spatially indexed and processed. Even though that the user is free to define its own identification mechanism according to application's needs, a couple of fast and reliable approaches are already provided during the Insertion operation. The first one affiliates each inputted object with its corresponding *DEF* value -a uniquely referencable attribute used by the X3D standard- while the second one employs the order of insertion, which is used to ascribe the current increment value to the inputted object. On the contrary, the entries of an internal node are represented by an array of other nodes, where each one is attributed with an identifier pointing to a rectangular parallelepiped and a bounding container which encloses this node's children. These identifiers are automatically produced during the construction of the R-tree with the assistance of a counter. The counter starts from *R1* which points to the rectangular area representing the root node and its value is increased each time a new node is created. This numbering methodology is quite useful for presentation and debugging purposes, since it monitors and reveals the subjacent updates that take place after the execution of any operation on an R-tree instance. In the following Fig 5-4 is depicted a simple X3DOM scene composed of five unrelated objects. These objects are afterwards indexed to an R-tree instance, creating the set of MBRs which is displayed in the

image underneath. The black-colored MBR denotes the root node of the R-tree, while the red-colored ones denote internal nodes of the data structure. On the other hand, a green-colored MBR denotes a leaf node which points to an indexed object. In the same figure is also presented the R-tree's taxonomy, which contains various information about its underlying data. The most important of them are the registration of each MBR's coordinates into the 3D space and the attribution of singular identifiers to each node and spatial object of the tree structure.

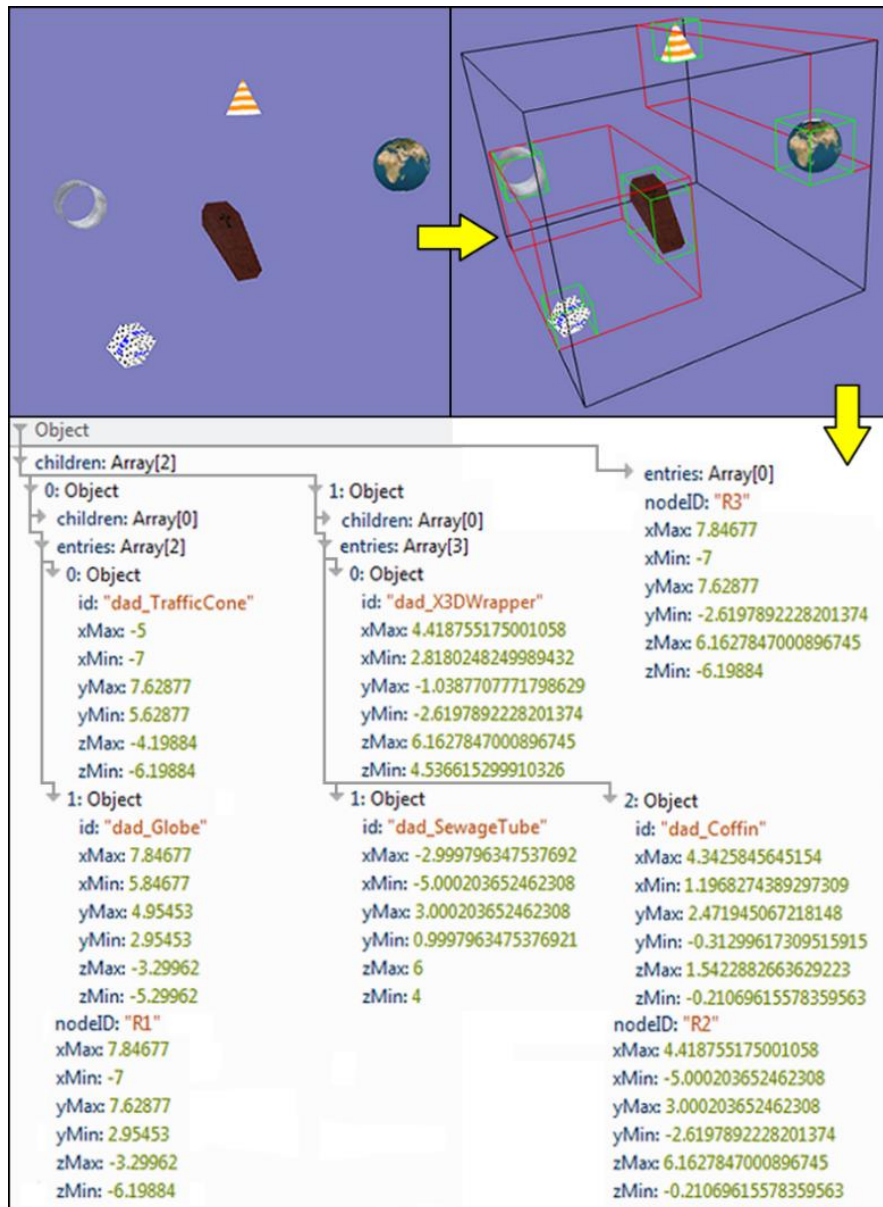
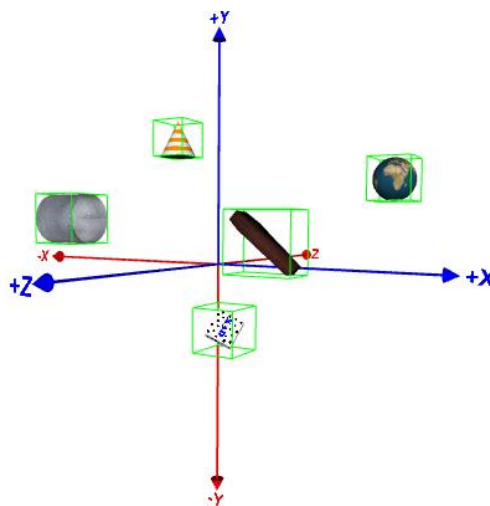


Figure 5-4 An indicative R-tree taxonomy

Leaving aside the identification variable, both types of nodes abide by the predefined range of minimum and maximum entries, denoted by the variables  $m$  and  $M$ , respectively. Another common point of reference between internal and leaf nodes lies to the projection of their location into the 3D space adopting a specific bounding container. The majority of R-tree variants make use of bounding boxes or bounding spheres [41] thanks to their simple and lightweight arithmetic computations compared to other bounding containers, like the bounding diamond, octagon and convex hull. Despite the fact that these containers can be also applied to any X3D shape, they are shipped with increased algorithmic complexity and gravely higher cost of computation power for web-based applications. Moreover, X3DOM's runtime environment comes with a concise API capable of inferring the raw coordinates of diaphanous bounding boxes, where the latter ones have been strictly implemented according to the X3D specification. The specification sheet states that all these boxes are oriented in the same direction with the axis, making inequality comparisons between these precomputed coordinates an easier procedure, compared to OBBs or the rest of bounding containers. Fig. 5-5 demonstrates the use of such bounding boxes for various geometric shapes in a 3D virtual environment, emphasizing at the unchanged orientation of the displayed MBRs, regardless of their relative enclosed object's plane angle. This figure is also accompanied by a schematic representation of the right-handed Cartesian coordinate system used by X3D standard, where  $+X$  points to the right,  $+Y$  points straight up and  $+Z$  towards the viewer.



**Figure 5-5 Each MBR is an AABB container**



The reasons described in the previous paragraph are more than enough to choose bounding boxes as the appropriate medium for indexing external information. However, the implemented R-tree takes advantage of bounding boxes not only for its structural purposes, but to also perform spatial queries on a given instance. The Searching operation –which is described in a later subchapter- provides three types of queries that involve space availability checking on a generated R-tree data structure. Intersection and overlap conditions constitute typical use cases of such tests and they are used to ascertain the sole existence of a bounding box in a finite search space. The only parameter that has to be checked is the integrity of its boundaries, where a non-trespassed perimeter validates successfully these two conditions. In this way, computationally heavy collision tests between the geometric figurines of X3D objects are avoided and they are instead reduced to simple inequalities relations between their corresponding MBRs (otherwise the number of possible collisions is factorial to the number of these X3D objects).

Summarizing, an R-tree data structure has been implemented for the efficient spatial indexing of 3D virtual environments. These environments have been fully integrated into the content of any modern Web browser thanks to X3DOM framework, which in this work is employed for information retrieval and presentation purposes. Any object of the 3D space that can be indexed is represented by one of the geometric shapes that are defined by the X3D standard, the 3D visualization technology used by X3DOM. The selected X3D objects' boundaries are approximated by a rectangular parallelepiped area which totally encloses this object. The indexing of objects takes place in the leaf nodes of R-tree, while internal nodes tend to reference their underlying set of nodes. After the insertion of the desired objects is finished, the compiled R-tree instance can be spatially queried and translate the result set to the appropriate X3D identifiers. In the following subchapters is extensively described the functionality of each operation used by the implemented R-tree data structure.

### ***R-tree operations***

The original R-tree data structure demonstrated satisfactory indexing and retrieval capabilities thanks to the utilization of a spatially modulated operation kit. That kit was composed of a set of cornerstone operations -like the *Insertion*, *Deletion*, *Update*, *Splitting*, and *Searching*- which were backed by more procedural routines. The usefulness of each operation was weighted according to the purposes of this study, in order to eliminate those routines that

could possibly be of no match for the tested 3D content. The Insertion and Splitting operations are inextricably linked to each other and their functionality is a must for any R-tree implementation. At first, it takes place the Insertion of the appropriate X3D objects, which have been chosen beforehand according to application's needs. Such objects are always indexed under a single leaf node, favoring the one which has to conduct the least enlargement of its area. In case this node has run out of records, then the Splitting operation is commenced to create the necessary space in this leaf and propagate the required changes upward. In this way, the primary space is constantly partitioned in several MBRs after an Insertion or Splitting algorithm fulfills its tasks, improving the space utilization factor and the execution time of spatial queries. On the other hand, a Deletion operation along with its Update routine were not deemed necessary to be implemented, since this work exclusively deals with virtual environments that contain static 3D content. Even though that the dynamic insertion of objects may violate the height-balanced leaf nodes, the underlying Splitting algorithm makes use of heuristic techniques to not only reduce the overlapping MBRs and their corresponding size, but to also provide a self-balancing feature to R-tree structure.

All of the above mentioned operations have as ultimate objective the spatial retrieval of the indexed objects at a later time. These objects' retrieval is accomplished with the help of the Searching operation. The implemented R-tree structure supports three of the most commonly used queries on spatial datasets, which are none others than the *Point*, *Region* and *k-NN* queries. Each one of them comes with a carefully designed algorithm for the swift deduction of accurate results under various scenarios, e.g. a location-based search. Such a scenario could involve a search on a finite collection of X3D objects, which have been attributed with a unique identifier and are spatially represented by an MBR and a set of Cartesian coordinates. The interrelated processes of this area and the rest of the algorithmic procedures have been classified in the following three subchapters, where each one describes in detail a major operation of the implemented R-tree data structure.

### ***Insertion***

Insertion can be defined as the operation of indexing a new entry to the appropriate leaf node of an R-tree instance. Such entries represent the objects that can be found in a virtual environment and they have marked for spatial registration. Each time that an entry insertion is requested, the corresponding algorithm has to traverse the tree in a recursively manner starting

from the root node. At this point, a well-founded and agile tree traversal methodology had to be implemented. However, even though that there is a variety of tree traversal options, R-tree can work flawlessly with only a few of them. So, an in-order traversal is rendered useless in front of an R-tree instance, since the data structure of the latter is not necessarily a binary tree. On the other hand, a level-order traversal would definitely spend much time on visiting inappropriate nodes, despite the fact that it could be applied to an R-tree structure. Such potentially inefficiencies led to the authoring of a pre-order algorithm, which makes it the perfect candidate for traversing any R-tree during an Insertion operation.

```

N = R-tree root;
E; //Entry to be inserted in a given R-tree instance
WHILE (N != typeof LeafNode) {
  FOR (each Node child of N) {
    xMBR = MAX(N.xMax, E.xMax) - MIN(N.xMin, E.xMin); //Axis X boundary
    yMBR = MAX(N.yMax, E.yMax) - MIN(N.yMin, E.yMin); //Axis Y boundary
    zMBR = MAX(N.zMax, E.zMax) - MIN(N.zMin, E.zMin); //Axis Z boundary
    newMBRArea = (xMBR * zMBR) * 2 +
                 (yMBR * zMBR) * 2 +
                 (xMBR * yMBR) * 2;
    enlargedArea = newMBRArea - originalMBRArea;
  }
  N = leastEnlargedAreaNode; //Follow least enlarged Node to next level
}

```

**Table 5-1 Traversing R-tree to find out the best leaf node for Insertion**

The implemented pre-order traversal starts from the 1<sup>st</sup> level of the R-tree, which is none other than its root. If the root node is also a leaf node, then the entry is assigned to it and the Insertion operation is terminated. At this point, the insertion of an entry may violate the maximum number of allowed entries for the selected node, a property defined by the variable *M* and attributed to the R-tree structure during its design stage. If such a thing happens, then a Splitting operation is initiating for that particular node, an operation described in the upcoming subchapter. However, in contrast to this extreme scenario, the root node can alternatively contain a finite number of internal nodes which can be seen as subtrees. These subtrees are checked one by one in order to find out which one needs the least area enlargement to include the new entry.

The most optimal subtree amongst them is returned and the same algorithmic process is addressed to its internal nodes. This operation is iteratively repeated for each R-tree level, returning each time a single internal node pointing to a new subtree. The overall procedure finishes when the best matching leaf node is reached and the relative entry is indexed into it. Afterwards, this newly enlarged area has to be propagated till the root, updating the MBRs of all ancestor nodes one by one. This propagation of changes starts from the leaf node and terminates to the root, following the entire subtree in an opposite route to reform the space utilization and to ensure the integrity of the generated R-tree instance.

In this study, the Insertion algorithm was executed numerous times for various use cases, bringing forward a special feature of the implemented R-tree structure. These tests revealed that the space utilization which is clearly depending on the nodes' MBRs, it is also directly related to the order in which entries are indexed to an R-tree instance. However, García et al in [42] proved that there was no trade-off between the chosen node to insert an entry and the performance of the R-tree. For that reason, they focused into developing an incremental refinement strategy to accelerate the Splitting operation, leaving aside the Insertion algorithm's functionality. On the other hand, a different approach was followed in [43], where insertion and splitting algorithms had both to be re-authored and optimized for the efficient management of 3D virtual geographic environments. In the following subchapter is thoroughly described the important role that such splitting algorithms play in order to maintain the balance of R-tree instances.

### *Splitting*

The Splitting operation can be definitely designated as the most important component found in any R-tree data structure. All splits that take place on an R-tree instance occur when a node is about to overflow, after reaching its maximum number of allowed entries. At first, this special occasion results from the necessity of inserting a new entry to an already full leaf node. Since this specific leaf node has been selected from the Insertion algorithm as the best fitting node, there is no other option than partitioning it into two distinct nodes. In this way, not only the requested space is successfully created, but the primary node's MBR has been also demarcated and must be recalculated for the new leaf nodes. So, the already existing entries and the new one are distributed amongst these two nodes according to strict splitting policies. Such policies take into account a set of parameters and try to minimize the area coverage and the overlapping MBRs. The first one guarantees that the time complexity for the construction of the tree is kept

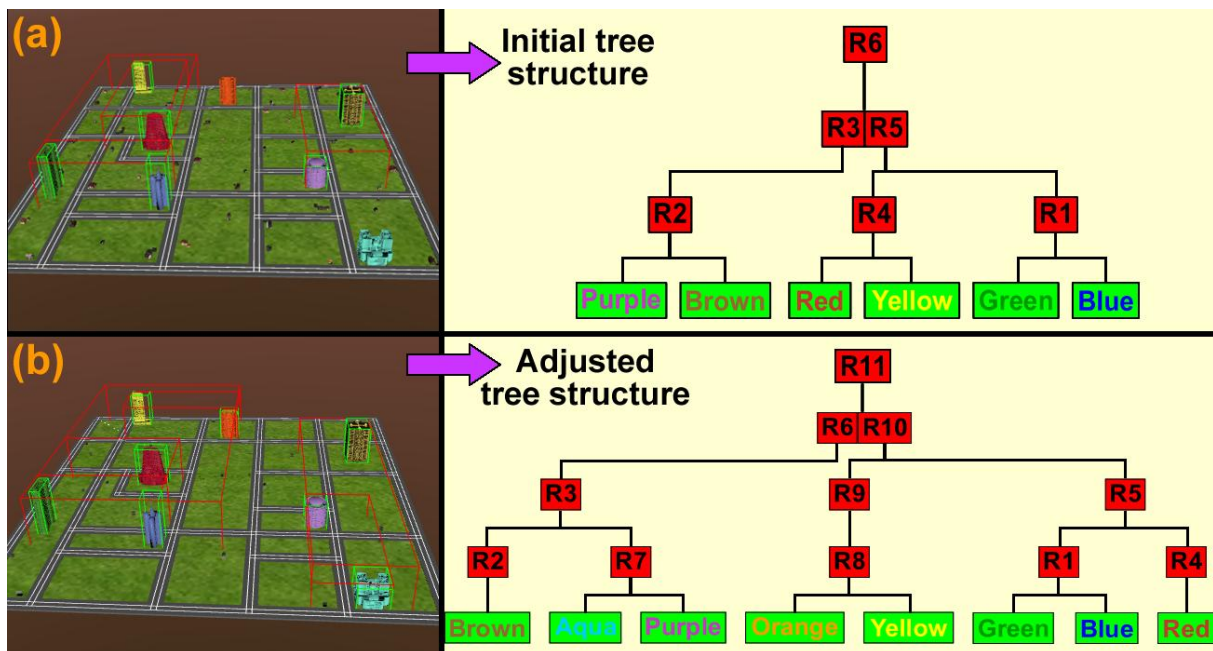
steadily at low levels, while the second one reduces as much as possible the number of visited nodes during the execution of queries. In short time after the introduction of R-trees, a couple of remarkable strategies [44] [45] were proposed to improve these two contradicting policies. However, there were applications where the algorithms beneath these strategies performed bad splits between nodes, resulting in turn to higher area coverage and slower query response. This thing demonstrated that a multitudinously parameterized splitting algorithm is not the only criterion which has to be taken into consideration, in order to wield the best possible space utilization and query performance. Instead, external factors like the domain which is going to be spatially indexed and the dataset's size are also deemed crucial to be known beforehand for the selection of the appropriate splitting strategy.

In this work, the underlying framework and its provided functionalities remain the same, despite the fact that the domain may vary according to the user's desires. Moreover, since this study is focused into the spatial annotation of VR environments on the Web, the dataset size will rarely exceed a few hundred when the latest splitting strategies target on tens of thousands of samples. So, the implemented splitting algorithm is based on *Quadratic* methodology, one of the three foremost and most frequently used splitting strategies proposed by Guttman back in 1984. As its name suggests, it takes quadratic time to split and readjust the tree after a split operation, guarantying satisfactory bipartition utilization and query performance. Each time a split at a leaf node is performed, Quadratic algorithm picks over the most wasteful pair of entries from this node in terms of area coverage. The first entry of that pair is inputted as the primary entry in the split node, while the second paired-entry is inputted in the newly created node. Afterwards, all remaining entries are checked one by one against the area covered from these two nodes. This involves the addition of the entry's and node's MBRs for the calculation of the area enlargement, denoted in the algorithm by a special preference value. This value is used to determine the least expanded node, which is the one who absorbs the inquired entry and updates its primary MBR dimensions. At this point, it was deemed necessary to define a set of alternative solutions in order to successfully cover the range of diverse paths that arise from the comparison computations made upon the nodes' MBRs:

- ❖ A non-expanded node points out that the entry totally falls into the MBR boundaries of the node, which in turn implies that entry's MBR is smaller than or equal to node's MBR.

- ❖ In case both nodes have been evenly expanded, then it is chosen the node with the smaller area coverage amongst them.
- ❖ In case both nodes have been evenly expanded and their area coverage has the same size, then it is chosen the node with the fewer entries.
- ❖ In case both nodes have been evenly expanded, and their area coverage and number of entries is the same, then the entry is randomly assigned to any node.
- ❖ Finally, if the minimum filled requirement is successfully satisfied for a node, then the remaining entries are led to the other node.

Of all these cases, only the last one can cause unreliable distribution of entries, leading to redundant increment of the search space, since such entries are inputted into a single node without any geometry checks. However, if a node reaches the maximum number of possible entries indicated by the equation  $M-m+1$ , there is no other option left for this greedy splitting strategy. The same tactic is repeated recursively from the leaf node to the root of the tree, following in that way the entire subtree which was chosen in the previous stage of the R-tree algorithm. This process is necessary in order to update the MBR dimensions of the parent node and check for available space in case both nodes are full. The overall procedure is known as *Adjustment* and preserves the integrity of any R-tree data structure after the completion of a split operation. This is rendered feasible by verifying that each R-tree property is successfully fulfilled at any given instance of its structure. In Fig. 5-6 below is displayed the bipartition criterion that takes place during a split operation, along with the appliance of an auto-balance trait used on every R-tree's leaf node. Both of these operations are concealed to the user and they are silently running into the background of the relative application. The next paragraph describes in detail the use case being deployed and tested on account of this subchapter. Moreover, its relative splitting and adjustment methodologies are followed step by step in order to demonstrate their practicality and usefulness into the implemented R-tree data structure. A virtual environment mapping a plain 3D city has been authored for the purposes of this use case. The landscape layout is composed of a grassy area with several streets leading to inhabitable areas. Each of these areas contains a set of standard residences which may coexist with a skyscraper building.



**Figure 5-6 Splitting & Adjustment operations on tree structure**

In this hypothetical scenario, we want to index such skyscrapers in an R-tree data structure in order to conduct spatial queries regarding their topology. In Fig. 5-6(a) is depicted the schematic model and the tree hierarchy at this specific stage of indexing procedure. The red-colored boxes represent an internal or leaf node, while the green-colored boxes represent the spatial object being indexed. In favor of simplicity, the color of each skyscraper is used as the unique identifier between the recorded spatial objects. Thus far, the majority of skyscrapers has been indexed apart from the *orange* and *aqua* colored ones. In Fig. 5-6(b) both of them are sequentially inserted into the R-tree, modifying its initial tree structure to the one illustrated in the same subfigure. It is evident that a few splits took place in order to rearrange the spatial objects according to the least enlargement area criterion. Such splits were boosted due to the fact that the maximum number of allowed entries per node was set to *two*, while the minimum number was retained to *one*. In this way, *R2* and *R4* nodes lose their pairs since the newcomer skyscrapers are in closer distance with both of them. So, the *R7* and *R8* nodes are created from scratch through the Quadratic splitting methodology, while at the same time, an adjustment operation takes care of balancing R-tree's leaf nodes. The latter one is based on the propagation of the affected MBRs to their corresponding ancestor node in a bottom-up strategy. Wherever is deemed necessary more splits occur in order to update these changes and maintain the integrity

of the data structure. The internal nodes  $R3$ ,  $R9$  and  $R5$  in the second subfigure suggest a typical example of this process. All these splitting and adjustment operations immediately cease after such changes are successfully propagated to the root of the tree, forming in this way different clusters of nodes each time a new skyscraper is inserted.

For sake of reference, the other two rejected splitting methodologies are the *Exponential* and *Linear* methodologies. The first one is the most optimal solution that can be found on R-trees -since it iteratively checks for the best possible combination of nodes- but it comes with the worst complexity times due to its brute force nature. The second one tries to maintain a uniform distribution amongst entries, outpointing the rest methodologies in terms of running time, at the expense of memory management and bipartition optimality. So, Quadratic methodology not only wields the best ratio between time complexity and space utilization compared to the before mentioned approaches, but it is also the ideal partner for static memory-confined virtual environments. Lastly, despite the employment of the latter methodology for the implementation of the splitting algorithm, R-tree can be always extended to support additional parameters found on other R-tree variants, like a perimeter-based split axis [46] and/or the reduction of pruning overheads [47]. However, such sophisticated heuristic solutions tend to find appliance only in very specific domains, due to the fact that R-tree's construction and maintenance rates are seriously suffer from the complexity of these algorithms.

### ***Searching***

The implemented R-tree algorithm takes as input a set of geometric objects which are organized in differential rectangular areas to answer various types of spatial queries. Its versatile structure not only guarantees its interoperability with large datasets of spatial objects, but it is also capable of representing any kind of geometry. In this way, any X3D object can be spatially indexed and queried, from the most widely used primitives (*Box*, *Sphere*, *Cylinder*, etc.) to the most complex *IndexedFaceSet* shapes. The supported queries can be divided according to their scope of use and their functionality, which may involve intersection, overlapping or nearest neighbor distance calculations. These data mining algorithms are summarized into three distinct subcategories and are eventually made available as point, region or k-NN queries [48]. On each occasion, a specific formula serves as the selection criterion that has to be satisfied, taking into consideration the spatial relationships between the engaged indexed objects. However, compared with other data structures, R-trees cannot be attributed with good and/or worst case searching



complexity, since the overall performance of their queries is heavily depending on the distribution of the objects in the search space and their relative geometry. Nonetheless, an average complexity for search algorithms is estimated to be around  $O(\log Mn)$ , where  $M$  is the maximum number of records allowed into a single node. In the following subchapters are described in detail the implemented queries, accompanied by their corresponding formulas that serve as the catalyst for their successful execution.

### ***Point Query***

A point query is the procedure of searching for and mapping a specifically located point to one or more spatial objects. The coordinates of the query point are translated in R-tree's index structure, where the MBRs of the latter one are successively checked for possible matching entries. In this work, such single points are defined by the classic three dimensional Cartesian coordinate system and the distance metric system used is the Euclidean distance. However, other distance metrics can be also authored for this category of queries, according to the needs of the underlying application. Moreover, it is worth mentioning that this type of queries -compared to NN queries- do not have to wage unnecessary and heavy distance calculations, but to simply satisfy the following inequality equation for each dimension:

$  \begin{aligned}  &(Point.x \geq Rect.xMin \ \&\& \ Point.x \leq Rect.xMax) \ \&\& \\  &(Point.y \geq Rect.yMin \ \&\& \ Point.y \leq Rect.yMax) \ \&\& \\  &(Point.z \geq Rect.zMin \ \&\& \ Point.z \leq Rect.zMax)  \end{aligned}  $
---

**Table 5-2 Inclusion condition in 3D space between point and rectangular parallelepiped**

Besides the axiomatic variables Point and Rect which respectively stand for a query point and a rectangular parallelepiped area, the coordinates' variables  $x$ ,  $y$ ,  $z$ ,  $xMin$ ,  $yMin$ ,  $zMin$ ,  $xMax$ ,  $yMax$  and  $zMax$  are used as upper and lower boundaries for nodes' MBRs. In this way, the given query point is simply comprised into a set of internal nodes, which are further traversed until their corresponding leaf nodes. This traversal is done with the assistance of a Depth-First algorithm that iterates R-tree's nodes level by level. However, making use of the aforementioned equation, DFS visits only those nodes that meet such inequalities, skipping a large subset of nodes at each level. This procedure keeps repeating until the algorithm reaches the lowest level MBRs, which are the ones that contain various spatial objects that may intersect the query point.

In case there are higher level nodes that have to be visited due to the existence of overlapping nodes, then DFS traverses their relative subtrees to scan for possible extra intersecting objects. The overall procedure finishes when there are no more paths to be searched for in algorithm's list.

***Region Query***

A region query -which is also known as a window query- is the procedure of searching for index records in a particularly located and shaped area of the 3D space. The placement of such an area is simply done by setting up its Euclidean coordinates, while its representation is feasible with various 3D geometric shapes. In this work we have implemented the most widely known search area, which is none other than the rectangular parallelepiped (a common rectangle in the corresponding 2D space). The coordinates resulting from the vertices of this rectangular area are inputted into a DFS algorithm that descends the R-tree in a certain order starting from the root in a top-down strategy. The algorithm's traversal order is based on a depth first tree search, where the appropriate node is purely chosen by this set of coordinates. The same set is also the responsible one for determining the rectangular parallelepiped overlap criterion in 3D space, which ultimately takes the form of the following formula:

$  \begin{aligned}  &(SearchRect.xMin < Rect.xMax \ \&\& \ SearchRect.xMax > Rect.xMin) \ \&\& \\  &(SearchRect.yMin < Rect.yMax \ \&\& \ SearchRect.yMax > Rect.yMin) \ \&\& \\  &(SearchRect.zMin < Rect.zMax \ \&\& \ SearchRect.zMax > Rect.zMin)  \end{aligned}  $
---

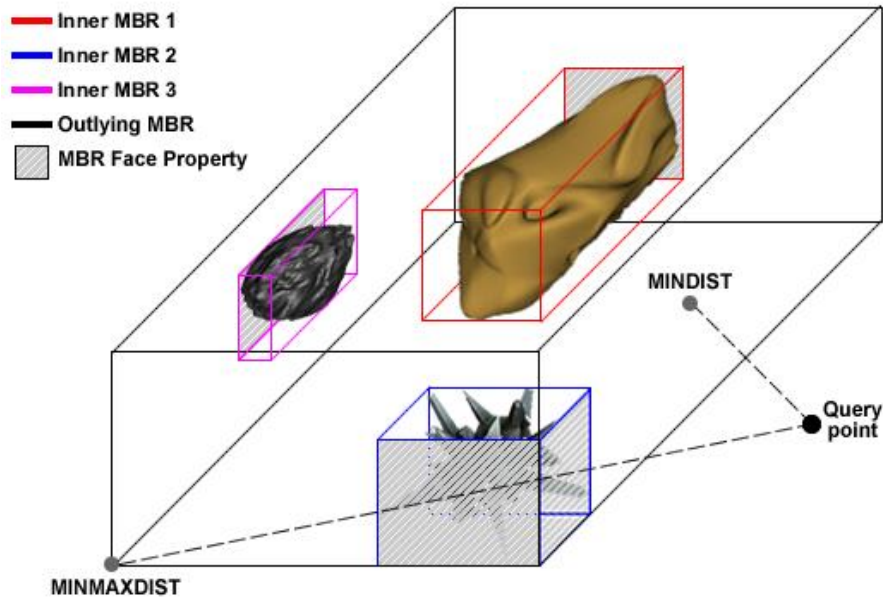
**Table 5-3 Overlap condition in 3D space between two rectangular parallelepipeds**

Based on this formula, DFS algorithm is capable of checking if the current node's MBR overlaps with the rectangular parallelepiped search area. In case it does, the search algorithm descends even further, to the children of the relative R-tree node. This procedure is continuously repeating until a leaf node is found, something which directs the search algorithm to focus into this node's entries. At this point, the validity of each entry is once more determined by their successful overlap with the predefined search area. Only those records that satisfy this requirement will be indicated as qualified, unmasking and returning their corresponding spatial objects. These objects can be further used for future processing according to the underlying application's needs or any way the end-user desires. Despite that the results coming from any

query would remain the same and integrally correct, the number of paths visited by DFS algorithm may change dramatically depending on  $m$  and  $M$  values of the R-tree instance. In case the virtual environment being indexed does not contain thousands of objects, it is not necessary to deal with these values. However, there are applications where query execution speed or space utilization are deemed especially crucial and these two values have to be checked for optimal results. Last but not least, it has been also eliminated the possibility of returning duplicate records of the same spatial object. Such a feature is feasible by implementing R-tree algorithm in such a way that although it allows the reflection of each object in many nodes, its indexing is exclusively done into a unique only leaf node.

### ***k-NN Query***

k-NN can be defined as the optimization problem of finding the nearest  $k$  points from a specific query point, amongst a finite set of points calculated by a distance measurement system. Today, numerous applications adopt a nearest neighbor or nearest neighborhood algorithm for the efficient manipulation of their datasets. Some of the latest research works in 3D point cloud domain came up with a set of novel methodologies for the space division, aiming either at the definition and decomposition of a cubic area [49], or an arbitrary usage of cell grids for the reduction of computational and reduction costs [50]. On the other hand, even though that various k-NN techniques have been presented through the passage of time, only a few of them have been widely adopted in R-trees. One of the most worth mentioning techniques is the MBR Face Property proposed in [51]. This technique consists of a branch and bound algorithm relying on the fact that every face of any MBR in an R-tree data structure contains at least one point of a spatial object. During a k-NN search the MBR face property makes use of two interconnected metrics to discover and order possible nearest neighbors of a query point  $P$ . Moreover, the algorithm is also capable of pruning unnecessary nodes, further improving its performance. The image below displays a query point and a MBR which encloses a set of smaller MBRs that point to either internal nodes or spatial objects. In the same image becomes also evident the usage of the two metric variables, *MINDIST* and *MINMAXDIST*, for the calculation of the faces' distance between the query point and the given MBR.



**Figure 5-7 The MBR face property**

In contrast with MBR Face Property, G. R. Hjaltason and H. Samet proposed the most optimal k-NN technique that can be found today on this area [52]. Their Global Order point of view traverses only the closest node at each round skipping unnecessary branches of R-tree, while at the same time, maintains a priority queue with the distances of the already visited nodes. Based on their work, a novel pathfinding Best-First algorithm has been developed to match the needs of the implemented R-tree version and be applicable to a 3D search space. The only metric being used is the MINDIST, which can be defined as the minimum distance between a query point  $P$  and a MBR. If the query point is overlapped by -or intersects with- the given MBR, then MINDIST equals to  $0$ . This checking is based on the hypothesis that point's coordinates have to be between the coordinates given by the upper left corner and lower right corner of the rectangular parallelepiped's perimeter. In any other case MINDIST denotes the minimum distance from the query point to either the MBR's perimeter or a nested spatial object. In this way, MINDIST guarantees a lower bound for every spatial object, discarding MBRs that come with higher bounds than the current best NN candidate. Such candidates are stored into a priority queue which allows only the highest priority nodes to be visited first. The queue utilizes an efficient binary min-heap data structure, which partially orders the already visited nodes according to their distance correlation between the given query point. By doing so, the queue does not contain any duplicate record and algorithm's greedy nature is overlooked by minimizing

its backtracking. As concerns the selection criterion for which node to visit next, this is overseen by the calculation of the lowest path cost between node's distances. The code segment in Table 5-3 below takes into consideration all the above mentioned facts and displays the methodology being used to determine the distance between a given query point and the rectangular parallelepipeds of an R-tree data structure.

```

priorityQueue = NewBinaryMinHeap(); //Priority queue starts at position 1
enqueue(priorityQueue, R-tree root, Infinity);
WHILE (k != 0) { //Return such records as the number of sample k
    element = dequeue(priorityQueue);
    IF (element typeof InternalNode) {
        FOR (each Node child of element) {
            enqueue(priorityQueue, Node, Node.MINDIST);
        }
    }
    ELSE IF (element typeof LeafNode) {
        FOR (each SpatialObject child of element) {
            enqueue(priorityQueue, SpatialObject, SpatialObject.MINDIST);
        }
    }
    ELSE { //If it was a SpatialObject, return it as the best NN
        RETURN (element);
        k--; //Decrease k by 1 and search for the next NN
    }
}
MINDIST(Node, Point) {
    x = MAX(Node.xMin - Point.x, Point.x - Node.xMax); //Axis X differential distance
    y = MAX(Node.yMin - Point.y, Point.y - Node.yMax); //Axis Y differential distance
    z = MAX(Node.zMin - Point.z, Point.z - Node.zMax); //Axis Z differential distance
    dist = SQRT(x^2 + y^2 + z^2); //Or another square root approximation formula
    RETURN dist;
}

```

**Table 5-4 k-NN BFS algorithm pseudocode**

At first, a binary min-heap data structure is initialized in order to keep track of the visited nodes, while the user-defined sample of  $k$  nearest neighbors is inputted to the BFS algorithm. Each time an R-tree node is visited, it is also inserted into the heap data structure. All these nodes are sorted in ascending order based on their distance correlation between the closest node's MBR and the given query point's coordinates. This distance approximation is backed by a second

algorithmic function, which is none other than the MINDIST formula presented in the same segment. The latter formula is responsible for the calculation of any requested distance between two points in the Cartesian coordinate system. Moreover, the same formula is supplied with the necessary *invoke* and *return* modules for the smooth cooperation with these fields of the priority queue that asked for its assistance. Ultimately, the resulting distance returns back to priority queue for storing and indexing purposes. The implemented MINDIST metric had to adopt a fast and efficient heuristic for the distance measurement between data points. The most widely known is the Pythagorean formula which not only guarantees accurate results, but it is also easy to comprehend and be applied in a 3D space. So, BFS algorithm takes advantage of the classical Euclidean distance and slightly modifies it to calculate the distance of a given query point and its nearest MBR. However, the rest of the distance calculations that were taking place during the construction of the tree and the execution of queries, made use only of inequality equations and/or squared distances. In this way, the usage of the classical squared root metric is kept at the minimum, saving both computational power and time.

At this point is evident that MINDIST can anytime be replaced by other distance metric systems, as long as they are capable of providing sufficient precision and satisfying computation speed. For example, MINDIST metric could be attributed with the capability to imitate a specific reciprocal square root computation known as Fast Inverse Square Root. Its functionality is analytically described in [53], proving to be a bit faster than Euclidean distance -while at the same time- remaining quite accurate. However, even though that the main concepts behind this methodology can easily be conserved, there are a few parts of the original function that have to be modified in order to run seamlessly under a JavaScript environment. Since the introduction of Typed Arrays in the latest JavaScript specifications and their corresponding affiliation by today's modern browsers, it is now possible to exploit binary data in raw memory. Yet, and in contrast with the majority of programming languages, JavaScript does not differentiate numeric arguments amongst its variants (integer, short, etc.), but it always treats them as 64-bit floating numbers of the international IEEE 754 floating point representation. So, initializing the relative Array buffers and setting the mathematically optimal constant for 64-bit number size to the hexadecimal `0x5fe6eb50c7b537a9`, a first guess for the reciprocal square root of a given floating point number is feasible. Moreover, there is always the option to use successive Newton-Raphson steps [54] to further improve the pre-calculated approximation at the expense of

performance. On the other hand, if performance comes with higher priority cost, then alternative solutions like Heron's method [55] or Bakhshali approximation [56] can be adopted to enhance the application's speed at the expense of approximation. No matter the case, it still remains a flavor of precision against speed, denoting that parameters like the underlying platform, the programming language and the application's needs have to be seriously taken into account for adopting the best possible solution amongst them. Finally, it is noteworthy that DFS and R-tree algorithms have been developed in such a way that are capable of deducting optimal solutions from repetitive nearest neighbor queries with slight only modifications. Doing so, the range of applicable domains is expanded much more, compared to the ones presented in the examples of the current chapter.

### **A computational model for spatial relations**

Spatial relations can be seen as a data mining procedure capable of inferring additional knowledge from a spatially indexed dataset. Such knowledge comes in the form of a linguistic vocabulary, where each one of its words defines a different spatial relationship between an object and another reference object of this dataset. The reasoning process is comprised of various factors (like the size, volume, position, etc.), which are ultimately used to structure a unified framework of relationships. This framework borrows concepts from psychology and computer science theories according to their perspective semantic views, contributing in that way to the specification of its underlying relations. However, no matter the composed framework or the type of standard being adopted, the general idea behind the spatial representation of a space of interest, lies to the abstract correlation of its objects' position, or this space segmentation into a finite set of regions.

Today, spatial relations can be applied into various domains, where each one of them tends to exclusively deal with a specific only subset of these relations. The most widely known classification scheme of the latter ones relies on the representation needs of the underlying space and its corresponding objects. In this way, three separate but closely affiliated categories were generated for the sufficient spatial annotation of a geometric space, namely the topological, directional and –the less paid attention- distance metric category, respectively. The majority of this work deals with directional relations, although there are a few topological relations which are occasionally used in specific stages of the R-tree algorithm. Although such topological

relations are briefly described in the upcoming subchapter, any spatial relation that is tested between two objects of an X3DOM scene is based on the directional type. The participating objects are associated with the Cartesian coordinate system, a three-dimensional distance metric which specifies an object's position with a signed triplet of numerical coordinates. These coordinates denote the distance of this object from a fixed location of three mutually vertical axes in the 3D space. Moreover, the minimum and maximum values from this set of coordinates are used to define the boundaries of a bounding container. The latter one is known as MBR and comes in the form of a rectangular parallelepiped. The vertices deriving from this type of geometry provide the necessary semantics for the deduction of spatially directional relationships. In the following subchapters, each spatial relation that takes place into the implemented data structure is comprehensively described, accompanied by the appropriate mathematical formula that results from the variable parts (i.e. object's geometry, algorithm's bounding container, number of planes, etc.) of the participating entities and the 3D space itself.




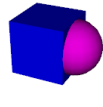

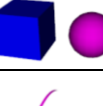

### ***Topological relations***

Nowadays, numerous applications [57] [58] come with a dedicated set of spatial operations for the implication of topological relationships between their objects. Most of them make use of *Oracle Spatial* and *Oracle Locator* extensions for the spatial reasoning of their dataset, adopting various intersection patterns defined in [59]. The latter one can be considered as a specification sheet which complies with the principles of a nine-intersection model for the topological classification of a geometric region. On the other hand, many commercial and open source RDBMS provide a mechanism for the spatial indexing of their stored records. Despite the fact that their spatial operators and integration level may vary from one implementation to another, their relative data mining mechanism is based on either OGC or SQL/MM spatial standards [60]. Both of these standards support a wide range of geometry types, which are attributed with a fixed set of properties for the manipulation of spatial information. These properties define and delimit three different regions for each spatial object (namely the boundary, interior and exterior regions), where their relative topological interconnection is inferred based on a spatial predicate representation method [61].

Such predicates, however, had to take into consideration the human spatial cognition in order to sufficiently define this kind of relations. In [62], human subjects were assigned with



miscellaneous tasks that involved the identification and grouping of topological spatial relations between two objects. The results of these case studies showed that the linguistic representation of space concepts used by the majority of subjects, concurred with the topological factors and geometry criteria used by the 9-Intersection model. In Table. 5-4 below are delineated all these spatial predicates between two predefined geometries, along with their semantic representation that corresponds to either a *Equals*, *Disjoint*, *Intersects*, *Touches*, *Crosses*, *Overlaps*, *Contains*, *Covers*, *CoveredBy* or *Within* human linguistic predicate. Even though that such topological relations are coming with various forms depending on the application domain, their classification methodology remains the same and is extensively described in DE-9IM [27] standard. In this way, it was granted an interoperable and practical topological model for a wide range of domains. In this work, the implemented R-tree data structure takes advantage of intersecting *within* and *overlap* formulas to perform spatial queries upon an R-tree instance, while an area coverage criterion is silently used during the construction of the tree. These extensions have been thoroughly presented in previous subchapters and they are similar to the topological predicates provided by DE-9IM.

 <b>Geometry A</b> <b>Geometry B</b>		
<b>Use Case</b>	<b>Associated Relations</b>	<b>Implicated Relations</b>
	Geometry A <b>contains</b> Geometry B	Geometry B is <b>within</b> Geometry A
	Geometry A <b>covers</b> Geometry B	Geometry B is <b>covered by</b> Geometry A
	Geometry A <b>intersects</b> Geometry B	Geometry B <b>intersects</b> Geometry A
	Geometry A <b>touches</b> Geometry B	Geometry B <b>touches</b> Geometry A
	Geometry A <b>overlaps</b> Geometry B	Geometry B <b>overlaps</b> Geometry A
	Geometry A <b>equals</b> Geometry B	Geometry B <b>equals</b> Geometry A
	Geometry A <b>contains</b> Geometry B	Geometry B is <b>within</b> Geometry A
	Geometry A and B are <b>disjoint</b>	Geometry B and A are <b>disjoint</b>
	Geometry A <b>crosses</b> Geometry B	Geometry B <b>crosses</b> Geometry A

**Table 5-5 DE-9IM topological relations**

Besides these MBR-related functionalities, the developed computational model is also capable of inferring the most commonly used topological relations in 3D space. As it has been stated before, the R-tree data structure consists of a multi-level hierarchy of rectangular parallelepiped containers known as MBRs. Their contents may be more lower-level MBRs or a family of spatial objects. The placement of such objects in 3D space derives from the leaf nodes of any R-tree instance, where their MBR coordinates are being recorded down, in order to proceed into an efficient spatial reasoning between them [63]. The spatial correlation procedure comes in pairs of MBRs and involves the appliance of a basic set of topological relations upon them. These relations come with a carefully designed taxonomy in order to prevent incorrect or nonessential implication of spatial annotations, since only one of the former can hold at a given time and space. For that reason, the implemented topological relations are serially tested one by one against every MBR pair, where the first valid occurrence amongst them points to the best fitting topological relation. This series of relationships and their corresponding allocation formulas between *RectA* and *RectB* rectangular parallelepipeds are presented below:

- ❖ At first, it is tested the possibility that these two MBRs are *disjoint*. Such a thing implies that neither the boundaries nor the interior regions of these MBRs are in contact. However, for the purposes of this work, the equality operator has been included into the formula shown in Table 5-5, since the intersection of a part of their boundaries alone does not affect the result set of the upcoming directional relations. In this way, we can also safely deduct a *touch* spatial predicate from the same category of relations.

$  \begin{aligned}  & (RectA.xMax \leq RectB.xMin \ \parallel \ RectA.xMin \geq RectB.xMax) \ \parallel \\  & (RectA.yMax \leq RectB.yMin \ \parallel \ RectA.yMin \geq RectB.yMax) \ \parallel \\  & (RectA.zMax \leq RectB.zMin \ \parallel \ RectA.zMin \geq RectB.zMax)  \end{aligned}  $
---

**Table 5-6 Disjoint or Touch condition for rectangular parallelepipeds in 3D space**

- ❖ In case that the above mentioned formula fails to satisfy a topological relation between two given MBRs, it is initiated the next closest spatial relation which is none other than the *equal*. In this occasion, the participating rectangular parallelepipeds must have in common not only their relative boundaries, but also their entire interior region. However, this kind of relation is rarely met in applications due to its strict constraints, which are shown in Table 5-6. So, the

majority of objects immediately proceed to the *within* criterion, where its formula has been displayed in Table 5-1 and thoroughly explained in a previous subchapter.

$$\begin{aligned} & (RectA.xMin == RectB.xMin \ \&\& \ RectA.xMax == RectB.xMax) \ \&\& \\ & (RectA.yMin == RectB.yMin \ \&\& \ RectA.yMax == RectB.yMax) \ \&\& \\ & (RectA.zMin == RectB.zMin \ \&\& \ RectA.zMax == RectB.zMax) \end{aligned}$$

**Table 5-7 Equal condition for rectangular parallelepipeds in 3D space**

- ❖ In contrast to the latter topological relation, there is a chance that the first MBR totally *contains* the second one, both in terms of its boundary and interior regions. This encasement also takes into consideration the possibility of intersecting boundaries, since the onus of containment focuses into comprising the relative interior region. Table 5-7 displays the inequalities operations that take place in order to determine if two rectangular parallelepipeds fall into this category of relation.

$$\begin{aligned} & (RectA.xMin \leq RectB.xMin \ \&\& \ RectA.xMax \geq RectB.xMax) \ \&\& \\ & (RectA.yMin \leq RectB.yMin \ \&\& \ RectA.yMax \geq RectB.yMax) \ \&\& \\ & (RectA.zMin \leq RectB.zMin \ \&\& \ RectA.zMax \geq RectB.zMax) \end{aligned}$$

**Table 5-8 Contains condition for rectangular parallelepipeds in 3D space**

However, there is one option left for a specific pair of MBRs that does not satisfy any of the afore-mentioned topological relations. This option lies to the validation of the overlapping criterion presented in Table 5-2, which denotes that a part only of their interior regions are successfully met in 3D space. At this point, it is worth mentioning that the rest of non-implemented relations (*Intersects*, *Crosses*, *Covers* and *CoveredBy*) are complementary to some of the implemented relations, providing in that way an efficient spatial representation of the 3D virtual environments.

This part of reasoning finishes when each spatial object has been uniquely attributed with a specific topological relationship. In cases of evenly arranged or overlapped entries becomes evident that this spatial representation is sufficient enough from a semantic spatial mapping of space. However, the *disjoint* topological relation is a special type of annotation which unlocks more spatial features on the underlying dataset. These features are represented by a different directional relation and their application is deemed necessary for any objects that are classified as disjoint. The following subchapter describes in detail the implicated directional relations, along

with the conditions that have to be met in each occasion, for the adequate provision of access to this kind of annotations.

### ***Directional relations***

Directional relations refer to another major type of spatial analysis, where the reasoning predicates incorporate direction constraints between two objects. From early times, there were works that took into account the human perception in directional management, involving either context sensitivity or psychological factors for the spatial reasoning of a space. In [64] was observed that this linguistic representation sometimes contradicted not only with the spatial relation term chosen by an artificial intelligence system, but also with various comparative concepts, like the orientation plane, the objective technique used and others. This fuzzy state of perception was based on the selection of a specific subset of vertices from a pair of objects. These vertices were used to determine the appropriate directional relations between these two objects. Over the passage of time, the latest technological advances allowed the automated generation of such relations for indoor scenes with the utilization of a robotic mechanism [65]. The entire process was based on a novel computational model, where its leading spatial quantities were the size and position of room's objects. These quantities gave birth to a set of principles for the generation of directional relations between all possible pair of objects, while at the same time unreliable relations were eliminated from the result set. The presented model was assisted by a 3D sensor capable of dealing with alternative configurations of indoor scenes, which have been loaded from a specific rooms' database.

All these studies pointed out that a visual representation of a space is always richer than its corresponding linguistic representation. This quickly became evident in spatial annotation applications, where qualities like *shape* and *color* were left aside compared to schematic traits like the *above*, *left*, *front*, etc. A series of experiments [66] proved that neither a qualitative or quantitative classification of a space were sufficient to capture similarities between spatial representation and spatial language. However, there were specific spatial propositions that could safely reflect a part of linguistic semantics in terms of applicability and accuracy. Such commonly used identification patterns were encoded to the *above*, *below*, *left* and *right* spatial relationships, which have been also adopted for the purposes of this work. On the other hand, a formal representation of natural language's spatial concepts was deemed quite a challenge,

mainly due to the fact that each interactive application can be addressed with complex and variant semantics that point to more than one kind of spatial relations [67]. For that reason, the composed computational model is capable of inferring various spatial relations between the objects of a virtual environment. This reasoning procedure is based on the spatial analysis of R-tree's MBRs, where a set of topological relations is applied among them, followed by the deduction of their corresponding directional relationships. These relations do not take into account the current viewpoint of the user, but instead they automatically annotate with spatial information any object of the 3D space, according to the Cartesian cardinal directions, which is the default coordinate system used by X3D standard. Each directional pattern is bestowed with a formula capable of testing spatial relationships between MBRs. In this way, complex and heavyweight spatial relation algorithms are avoided, where a simple and efficient methodology is used to significantly boost the reasoning performance. Moreover, even though that this technique is an approximation of the best fitting relation amongst the available directional relations, its anticipated outcomes are quite satisfactory in terms of accuracy.

The authored formulas act as a uniform classification rule for the categorization of an MBR pair into one or more directional relation types. This stage of spatial reasoning is exclusively accessed when this pair's topological relation is derived to be *disjoint*. In this way, it is guaranteed that the MBRs of any R-tree instance are used as the ground to indicate the underlying directional relationships between their enveloped objects. Below, *RectA* and *RectB* rectangular parallelepipeds represent the minimum bounding box of two different objects. The first one points to the object which has to satisfy the following set of directional criterions according to a second reference object, in order to be successfully categorized into the appropriate spatial category:

- ❖ When the rightmost boundary region of the tested MBR has lower value than -or equal value to- the leftmost boundary region of the reference MBR, then the first object is located on the *left* side of the second object. On the other hand, if the leftmost boundary region of the tested MBR has higher value than -or equal to- the rightmost boundary region of the reference MBR, then the first object is located on the *right* side of the second object.

<i>Left</i>	$RectA.xMax \leq RectB.xMin$
<i>Right</i>	$RectA.xMin \geq RectB.xMax$

**Table 5-9 Left & Right conditions for rectangular parallelepipeds in 3D space**

- ❖ When the lowest boundary region of the tested MBR has higher value than the highest possible boundary region of the reference MBR, then the first object is located *above* the second object. On the other hand, if the highest boundary region of the tested MBR has lower value than the lowest possible boundary region of the reference MBR, then the first object is located *below* the second object.

<i>Above</i>	$RectA.yMin > RectB.yMax$
<i>Below</i>	$RectA.yMax < RectB.yMin$

**Table 5-10 Above & Below conditions for rectangular parallelepipeds in 3D space**

- ❖ When the lowest boundary region of the tested MBR is equal to the highest boundary region of the reference MBR and these two boundaries are intersecting even at a single point in 3D space, then the first object is *over* the second object. On the other hand, if the highest boundary region of the tested MBR is equal to the lowest boundary region of the reference MBR and these two boundaries are intersecting even at a single point in 3D space, then the first object is *below* the second object, but the second object is *over* the first object.

$(RectA.yMin == RectB.yMax \parallel RectA.yMax == RectB.yMin) \&\&$ $(!((RectA.xMax <= RectB.xMin) \parallel (RectA.xMin >= RectB.xMax))) \&\&$ $(!((RectA.zMax <= RectB.zMin) \parallel (RectA.zMin >= RectB.zMax)))$
---

**Table 5-11 Over condition for rectangular parallelepipeds in 3D space**

- ❖ When the most posterior boundary region of the tested MBR has higher value than -or equal value to- the frontmost boundary region of the reference MBR, then the first object is located in *front* of the second object. On the other hand, if the frontmost boundary region of the tested MBR has lower value than -or equal to- the most posterior boundary region of the reference MBR, then the first object is located *behind* the second object.

<i>Front</i>	$RectA.zMin >= RectB.zMax$
<i>Behind</i>	$RectA.zMax <= RectB.zMin$

**Table 5-12 Front & Behind conditions for rectangular parallelepipeds in 3D space**

Once more, boundary representation methodologies have been established in order to imply the appropriate directional relations for each use case. The overall mechanism is also capable of deriving the inverse spatial relationships, wherever they can be applied without performing any needless reasoning procedures. Such a thing occurs when the second object prevails over the first, according to the requirements set by the corresponding directional relation. At this point, it's worth mentioning that the number of implicated relations is massively increased relative to the number of indexed objects. This phenomenon along with the upcoming need of parallel programming in World Wide Web [68], prompted the use of web workers for the purposes of this work. The computational model presented above has been implemented in a separate JavaScript script with the assistance of a web worker [69], which silently runs in the background and allows the user to freely interact with the rest of his platform. Last but not least, web workers are capable of avoiding the deficiencies met in case of slow response times, while at the same time, they are able to exploit multicore machines in a more efficient way than the classic JavaScript programming methodologies.

### **Semantic annotation of spatial relations**

Up until this point, the implemented 3D R-tree data structure is not only browser-independent, but it provides all of its indexing and spatial features under a single JavaScript library. In this way, no plugins or additional programs have to be installed in client side, incorporating faster response times and less resources burden on the machine. However, the implicated spatial relations between the indexed objects of a 3D scene are available to the user via a simple HTML element, instead of integrating them into some kind of Semantic Web solution. Hence, an abstract semantic layer capable of reusing such relations was deemed necessary in order to apply and reason with them from a semantic scope.

In [70] was presented a technique for the uniform application of spatial reasoning into empty or vague regions of a topology. The proposed model was an extension of *RCC5* and *RCC8* schemes and helped to the mathematical formalization of a set of qualitative spatial relations from a semantic scope. Such *RCC8* predicates were also employed for the development of new OWL-DL axioms [71] capable of dissociating ontology's concepts into spatial regions and applying a small subset of topological relations upon them. However, the implementation of such spatial axioms was not only a quite complex procedure, but their representation capabilities were

also clearly depending on RCC scheme's reasoning capacity. The latter fact comes in contrast with the implemented R-tree's relations, which are based on DE-9IM, a totally different spatial reasoning methodology. So, even though that these works are capable of building useful scene descriptions, they are far apart from this study's cornerstones. For reasons like this, it had to be authored an ontology that won't unsettle the prototype state of axioms, but it would instead take advantage of them, in order to represent the spatial relations coming from R-tree algorithm. Moreover, its property characteristics had to be able to derive inverse semantic relations and avoid needless computations, since the majority of spatial relations come with a counterpart functionality which can be represented by an OWL axiom (reflexive, symmetric, anti-symmetric and transitive). Such axioms could also be combined in a later time to express a VR environment with advanced semantics.

For the purposes of this thesis, the ontological framework presented in [28] has been utilized as the proposed semantic layer of information. The semantic composition of its ontology is based on a specific domain (interior design and decoration), designating the appropriate classes and their taxonomy, the relationships between the former, and ultimately defining possible class and property restrictions. The development of the ontology was done in OWL language, which is a W3C's recommendation with advanced capabilities in the description of classes and properties. Specifically, it is going to be used the *OWL-DL* sublanguage of OWL [72], in order to express first order logic and to ensure that the resulting statements are valid, through the restrictive nature of this sublanguage. The development of this ontology led to a detailed conceptualization of indoor scenes, identifying the essential semantic concepts for the efficient description and reuse of such environments, along with their corresponding 3D content. For the time being, DEC-O consists of the semantic spatial representation properties shown in Fig. 5-8. It is evident that a part of these object properties is directly related to the directional relations that are implicated through the presented computational model, while topological relations have not been implemented in this kind of framework. However, the semantic annotation of the latter leaves unaffected the result-set of the former, since all of them have to satisfy the *disjoint* criterion in order to proceed to such a semantic supplement. In the last section of the following chapter is employed a typical indoor scene for the deduction of spatial relations between a specific subset of objects with the assistance of an R-tree instance and DEC-O's web service.



The screenshot shows a web-based interface for editing semantic properties. On the left is a list of 'Object properties' for DEC-O, including various spatial relationships like 'is\_Across\_The', 'is\_Behind\_Of', etc. The 'DEC-O:Intersects' property is selected. The main area displays a table for this property and two panels for its domain and range.

Property	Value
rdfs:comment	
ace_lexicon:pl	Intersects
ace_lexicon:sg	Intersectses
ace_lexicon:vbg	Intersected by

Domain	Range
DEC-O:Structural	DEC-O:Structural
DEC-O:Content	DEC-O:Content

**Figure 5-8 Semantic spatial properties of DEC-O**

Summarizing, the proposed R-tree data structure and its relative operations can be used as the foundation for the construction of a formal ontology -or the cooperation with an existing one- aiming at the efficient spatial description of VR environments. This semantic representation of spatial knowledge can also be further enhanced with a query mechanism like SPARQL [73] or Linked Data, in order to conceal the R-tree reasoning process and provide the end-user of the application with a flexible and user-friendly environment. No matter the path chosen, each ontology has to be carefully designed, in order to sufficiently classify the interested domain's concepts and adequately represent the relationships between these concepts.

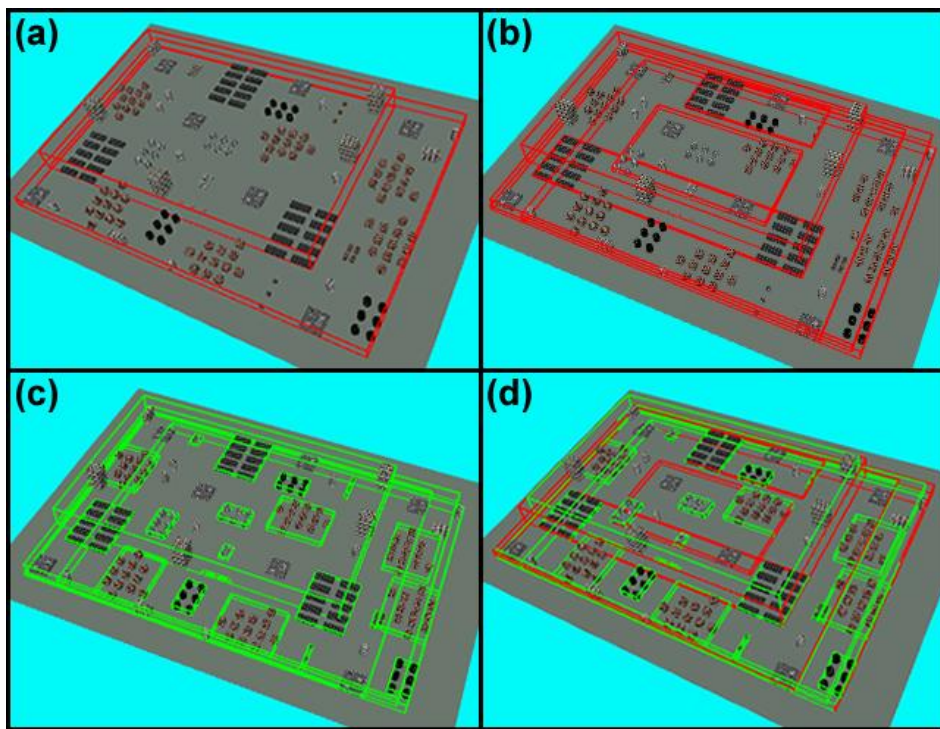
## Chapter 6 - Experimental evaluation

The implementation part was exhaustively confronted with numerous scenarios in order to certify the unremitting functionality of the authored algorithms and the validity of their corresponding results. In the previous chapter, a couple of simple examples were presented to highlight certain aspects of the OWL ontology and R-tree's data structure. In this chapter, however, it takes place a systematic monitoring and review of a scenarios' series. Each scenario stands for a disparate use case which comes in the form of an X3DOM virtual environment. This environment has been set up according to the outline of its relative scenario, while specific only objects from the subjected 3D space have been spatially indexed based on the use case being tested. The entire process of spatial annotation was carefully observed step by step, fixing a few key issues that had arisen from inconsistencies in the logical system. Moreover, the result set of each scenario was thoroughly examined and evaluated against the expected results, modifying either the original structure of the R-tree, or optimizing the performance of the underlying algorithms wherever that was deemed necessary. These scenarios have been divided according to their spatial usability into the following subchapters, where each one addresses a specific state of its runtime instance.

### R-tree taxonomy & stretch tests

At first, it will be presented a complex scenario which includes a finite set of randomly distributed boxes across a 3D space. The scenario aims at mapping a typical city structure, where the latter boxes are used as a representation mean for various city buildings. Even though that a quick glance at the authored X3DOM scene may lead to the conclusion that it is a very simple environment, it instead comes with a vague clustering sequence of about 180 objects into a set of 24 distinct *Group* nodes. This scattered distribution of objects creates many undefined areas of jurisdiction that have to be taken into consideration during R-tree's indexing and splitting operations. In order to highlight the taxonomy of nodes in this R-tree instance, a series of level-layered approaches is presented below. All of them make use of the *Insertion* and *Quadratic* algorithms to spatially index 24 groups of objects, where each R-tree node supports a fixed range of entries with lowest possible value of  $m=2$  and highest possible value of  $M=4$ .

The first level of this R-tree instance is depicted in Fig. 6-1(a), which is none other than the root of the tree structure comprising of two only nodes. This level and its relative nodes contain the most abstract layer of information, since the rest of levels come with an increased number of nodes that target specific areas of the virtual environment. Fig. 6-1(b) displays the second level of the same R-tree instance, which points this time to the internal nodes of the data structure. The latter ones do not only compose the children of the aforementioned level, but they also indicate next level's nodes. The third and final level of this data structure can be seen in Fig. 6-1(c) and refers to R-tree's leaf nodes, where it becomes obvious that the overlapping ratio of its indexed objects is based on their insertion order and abnormal initial distribution in 3D space. Ultimately, the entire R-tree taxonomy is highlighted in Fig. 6-1(d) by concatenating all the generated nodes of each individual level.



**Figure 6-1 Various levels of an R-tree instance**

This hierarchical representation of R-tree's nodes can be seen as an effective clustering methodology aiming to improve the rest of its spatial operations. However, such a thing revealed the need for a clarification of its indexing and reasoning capabilities in terms of computation speed. For this reason, a brief -but extensive- stretch test took place upon the same scenario. The

generic structure of the virtual environment remained the same on purpose, in order to sufficiently compare R-tree's performance and scalability features under a common pattern. The only thing that varies is the definition of two distinct use cases, where each one comes with an increased number of randomly distributed shapes of the original pattern. As concerns the core R-tree parameters, the minimum and maximum node sizes were set to  $m=10$  and  $M=20$ , respectively. These two parameters were kept the same across all three use cases and R-tree instances. On the same side of coding, the execution times of algorithms were validated according to the *Performance.now()* method of JavaScript specification [74], where its relative set up and functionality are shown and explained in the following table.

```

FOR (each operation in R-tree) {
    startTime = new TIMER();
    ... //Main body of the relative operation
    endTime = new TIMER();
    elapsedTime = endTime - startTime; //Return elapsed time in milliseconds
}
TIMER() {
    currentTime = performance.now(); //Make use of High Resolution Timer
    RETURN currentTime;
}

```

**Table 6-1 Time measurement pseudocode**

At this point, it's worth mentioning that there are various alternative solutions for the measurement of time in a JavaScript-based implementation, like the *Date.getTime()* and *console.time()* methods. However, the methodology presented above was tagged as the most appropriate one, since it provides more accurate timing compared to the rest of JavaScript functions. This function is used for all kinds of time measurements in this subchapter, regardless of which browser is being used. In this experimental evaluation, all use cases took advantage of *X3DOM runtime 1.6.2* JavaScript library<sup>3</sup>, which cooperated with *Firefox v47.0.2505.0* browser and *Windows 7 SP1 64bit* OS. On the other hand, the hardware side of the test environment was composed of a laptop computer with an *Intel Core i5-2430M* processor, *4GB* RAM and an *Intel HD Graphics 3000* as the graphics processor unit of the system. Besides the aforementioned

---

<sup>3</sup> <http://x3dom.org/download/>

configurations, no further changes were made to the software or hardware components of the test environment and its use cases.

At first, the number of shapes and triangles from each use case were obtained with the assistance of X3DOM environment's *showStat* parameter. Afterwards, an R-tree instance of each use case was immediately created and its relative generation time was recorded with the above mentioned JavaScript function. Then, the provided spatial queries were put to test in order to check out their relative performance against all three use cases. Each one of these queries was executed on a large volume of the given dataset, which have been indexed into an R-tree instance at an earlier time. In this way, the presented measurements are as realistic as possible, since searching operations have to visit not only overlapping entries, but also multiple branches of the same subtree. Finally, the computational model for the automatic generation of spatial relations is also put to test, recording down its implication time and the number of the returned relations. The first two R-tree instances make use of the same VR environment and spatial predicates, while in the third one is utilized a slightly modified version of a 500 components scene from X3D archive list<sup>4</sup>. So, the implemented model itself is the only one responsible for the deduction of all possible types of spatial relations that hold between the indexed objects. All three use cases along with the computation costs of their corresponding operations have been collected and are illustrated into Table 6-2.

<i>Dataset Size</i>		<i>R-tree index</i>	<i>Spatial Queries</i>				<i>Spatial Relations</i>	
<i>No. of components</i>	<i>No. of triangles</i>	<i>Index time (ms)</i>	<i>Point (ms)</i>	<i>Region (ms)</i>	<i>1-NN (ms)</i>	<i>10-NN (ms)</i>	<i>Automatic implication (ms)</i>	<i>No. of relations</i>
25	6564	15	0,53	1,53	0,57	1,32	2,1	1055
178	6564	65	0,65	3,9	0,81	2,24	2,92	75376
500	576000	132	0,45	0,15	0,72	1,61	6,03	602250

**Table 6-2 Time costs for each R-tree operation**

---

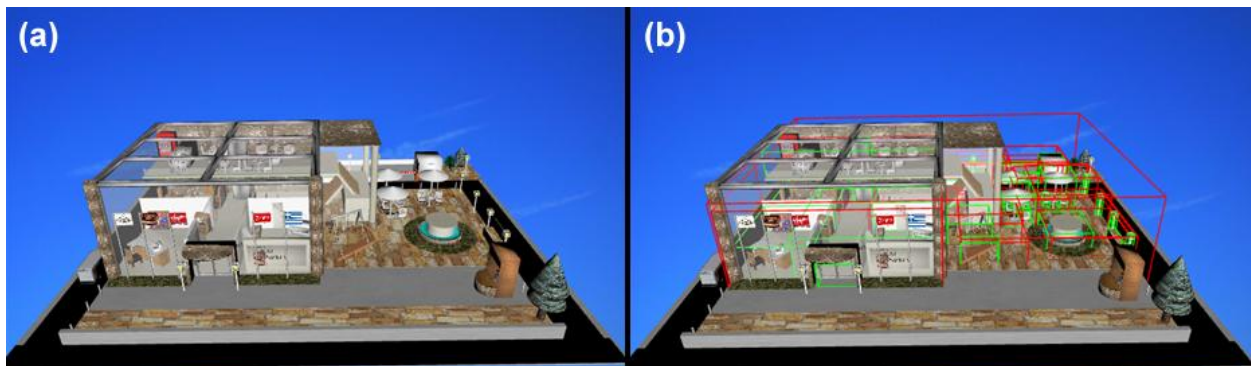
<sup>4</sup>[http://www.web3d.org/x3d-resources/content/examples/ConformanceNist/GroupingNodes/Collision/\\_pages/page01.html](http://www.web3d.org/x3d-resources/content/examples/ConformanceNist/GroupingNodes/Collision/_pages/page01.html)

As expected, R-tree's generation time is steadily increasing in each use case, due to the larger number of indexed components. The number of triangles does not seem to affect the indexing procedure or the rest of spatial operations, but is displayed as a load criterion from the perspective of X3DOM framework. Even though that the first two cases may handle a different number of indexed components, they share the same badly distributed 3D space of objects. This comes with a negative impact on R-tree's overlapping ratio and indexing performance. In contrast to these use cases, the third one is comprised of more components but a better tree structure, boosting in this way its spatial operations compared with the former. Regarding spatial queries, their performance was found outstanding on all three use cases, since their corresponding execution times were close to zero. In the first two cases, the number of overlapping entries was higher than the overlaps detected in third case, making each query algorithm to visit multiple subtree in order to reach the desired output. On the other hand, k-NN queries came in two flavors for the purposes of this test environment, where they had to discover the *first closest* neighbor and the *top ten closest* neighbors. Based on the experimental results, it was estimated that this increment into spatial reasoning process, led to the doubling of the initially recorded execution time. As concerns the computational model of spatial relations, its relative time costs remain equally low, but the number of implicated relations is rapidly growing according to the number of indexed components. It has been noticed that relations' number is the second power of the indexed components multiplied by a real number, which is around 2.5 and is slowly increasing in each use case.

Summarizing, all time costs remained in satisfying response times, even when the number of indexed components and spatial relations raised massively. It is also noteworthy that these times could be even better (or even worse) in case of different values in the core parameters of R-tree data structure,  $m$  and  $M$ . On the other hand, the test environment was composed of a moderate-performance computer for today's standards, leading to an undeniable improvement of these execution times. Lastly, it has to be noted that the main concept of these stretch tests was to assess the functionality of each implemented operation and its underlying algorithm in generic use cases. In order to provide more accurate time costs for a wide range of applications, an advanced test environment with various computers and more use cases has to be set up and spatially analyzed.

## Performing spatial queries

In a previous subchapter, three different types of queries were presented for the sufficient spatial reasoning on an indexed dataset of an R-tree instance. These queries are able to take advantage of the spatial relationships between the generated MBRs, in order to infer additional knowledge about the location or the state of an indexed object. The drawing of such conclusions is clearly depending on the geometric formation of the participating nodes, where disjoint sets of nodes are intentionally skipped to improve the query performance. In the following paragraphs are described in detail various scenarios, where each one comes with miscellaneous illustrations for maximal comprehension of its corresponding use case. In this way, it is accomplishable a conclusive demonstration of the provided searching capabilities, which are propelled by the intrinsic query algorithms.

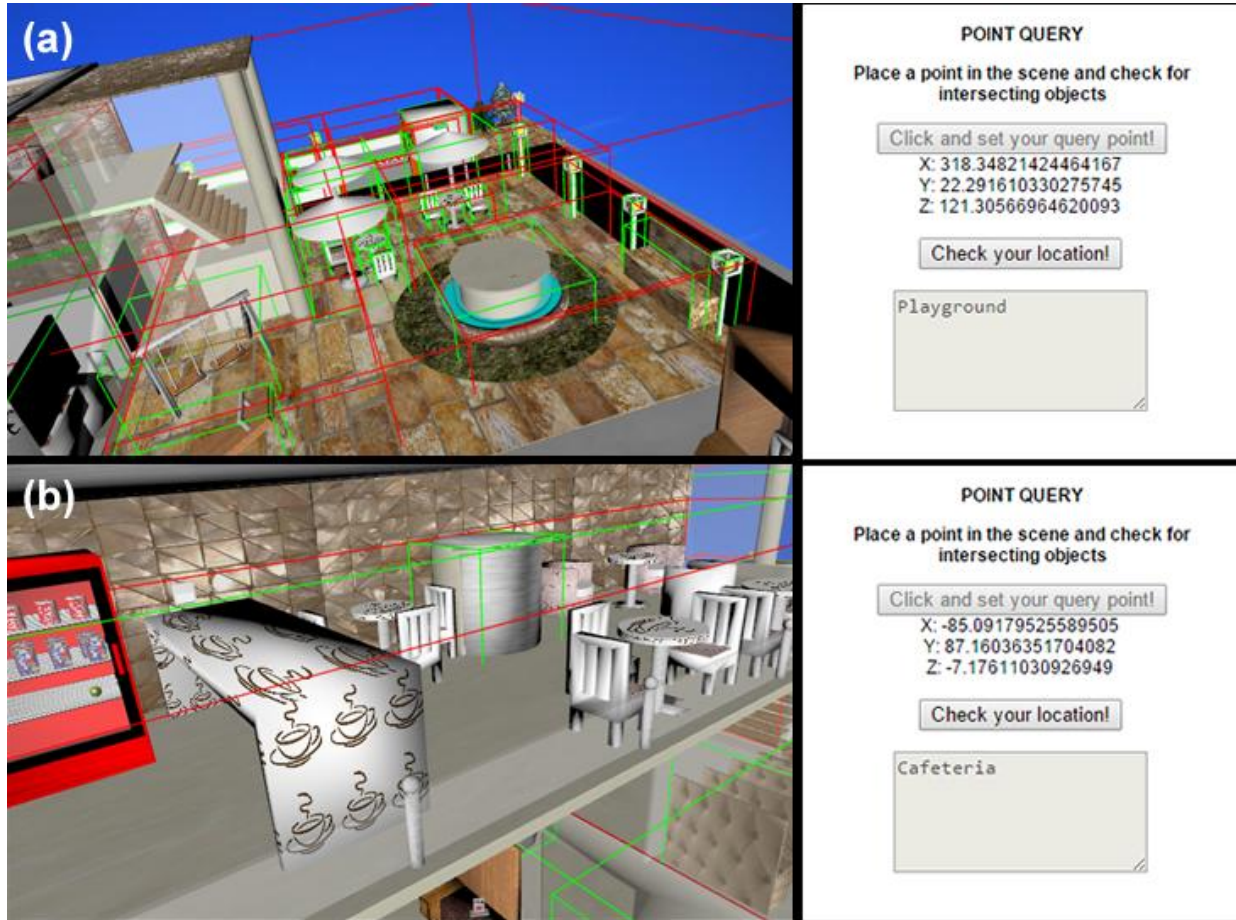


**Figure 6-2 Indexing a Shopping Mall for location identification purposes**

In Fig. 6-2(a) is displayed a complex X3D scene composed of a shopping center along with its surrounding area. This virtual environment comes with a large dataset of X3D objects that have been unified in numerous groups to ultimately form distinct activity areas. In this use case, R-tree algorithm indexes specific objects from the original dataset in order to provide location identification features. The overall mechanism is based on the definition of diverse areas of use, which are differentiated by their records hierarchy in the tree. More specifically, these areas have been attributed as Playground, Parking, Cafeteria, DVD Club, Cinema, Record Shop, Furniture Store and Elevator. The relative data structure which is going to be traversed can be seen in Fig. 6-2(b) along with a quick glimpse of the drawn MBRs. From this moment on, a point query can



be raised against this R-tree structure to retrieve the characterization of a specific location, which has been indicated by the audience of the virtual environment.



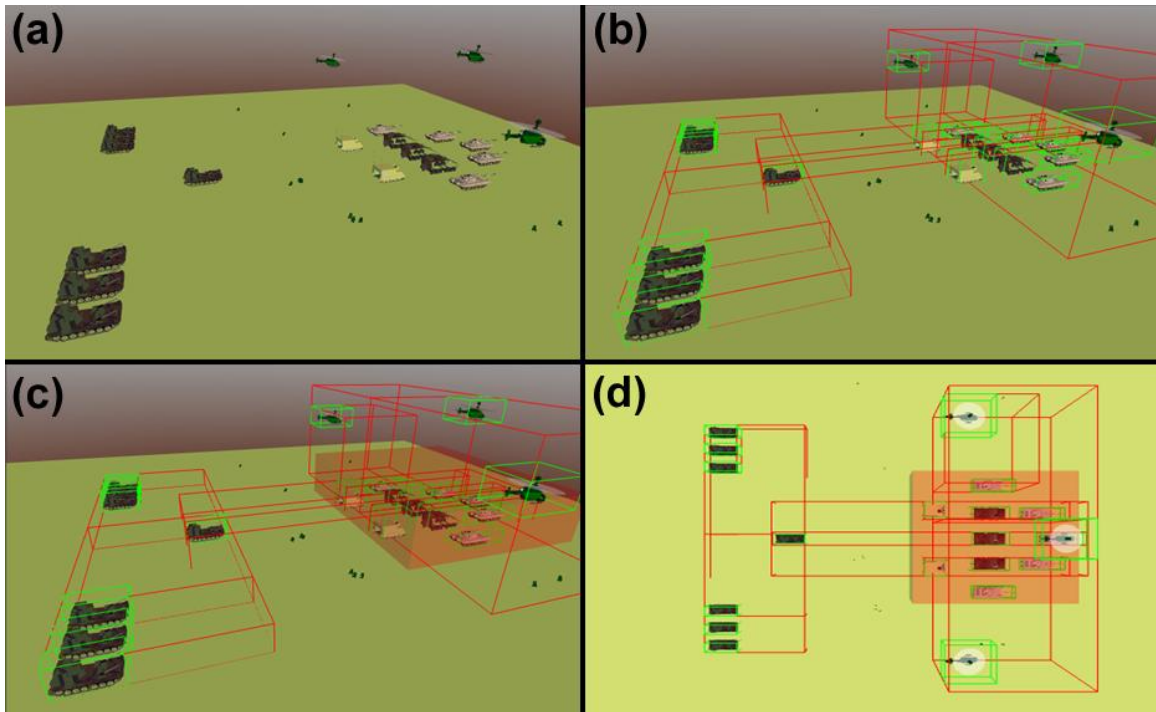
**Figure 6-3 Estimating location based on a chosen object**

At this point, it is feasible to navigate through the X3D scene and test the validity of algorithm's results, since the indexing procedure of the desired spatial objects has been successfully done. The query point is represented by a small silver-colored X3D Sphere element, which acts as the reference point for the derivation of the chosen object's coordinates. In the first scenario shown in Fig. 6-3(a), the top side of the fountain has been marked, returning its corresponding coordinates and denoting that the selected spatial object belongs to the Playground area. On the other hand, Fig. 6-3(b) displays a second scenario where the query point picks the front side of a refrigerator on the 1<sup>st</sup> floor of the shopping center. DFS algorithm traverses the data structure once more, inferring that the refrigerator belongs to the Cafeteria area of the X3D scene. Hereby,



both scenarios make use of the coordinates picking buffer proposed by X3DOM framework thanks to its ease of use and implementation simplicity. However, programmers and/or end-users have always the option to define their own unique coordinates' representation system, in order to match their application needs and guarantee its interoperability.

On the other hand, Fig. 6.4 below displays a typical example of a region query, which is followed by a brief description of this use case as concerns the algorithm's functionality in the underlying R-tree structure. For the purposes of this use case it was created the fictional war scenario shown in Fig. 6-4(a), which is composed of a finite set of military vehicles and helicopters in a tree scenery.



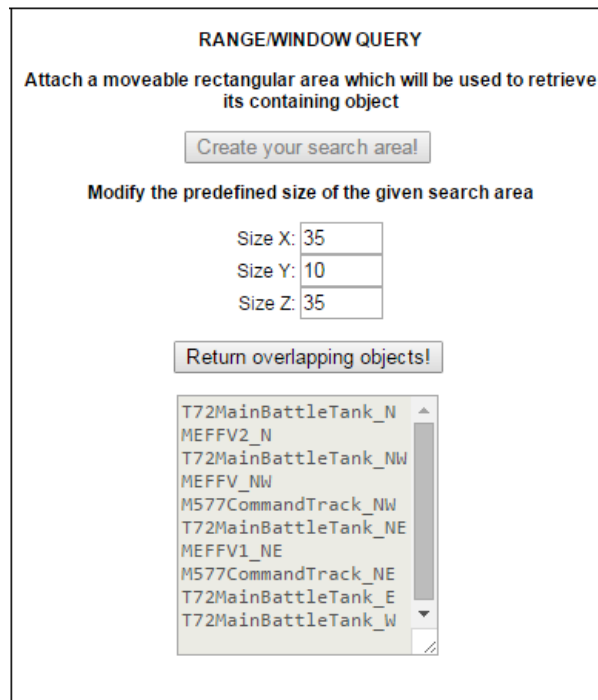
**Figure 6-4 Execution steps for a region query**

This X3D scene has been loaded into X3DOM framework and its depiction is feasible with the assistance of the added *Extrusion* and *ElevationGrid* JavaScript components. Afterwards, the R-tree algorithm initiates the suitable routines to annotate the scene's spatial objects as presented in Fig. 6-4(b). At this point, the user is able to create the necessary instance of the transparent-red

---

<sup>5</sup> X3D models have been taken from Savage X3D Examples Archive (source: <https://savage.nps.edu/Savage/>)

rectangular parallelepiped and search for the desired spatial objects. In this use case, it is assumed that the user wants to know how many -and what kind of- ground units have already marched to the front. So, the search area has to be resized and relocated as shown in Fig. 6-4(c) and Fig. 6-4(d), in order to match the requirements set by the user. Its defined dimensions for each axis are presented in the following figure, narrowing down the search space to the boundaries given by these three values.

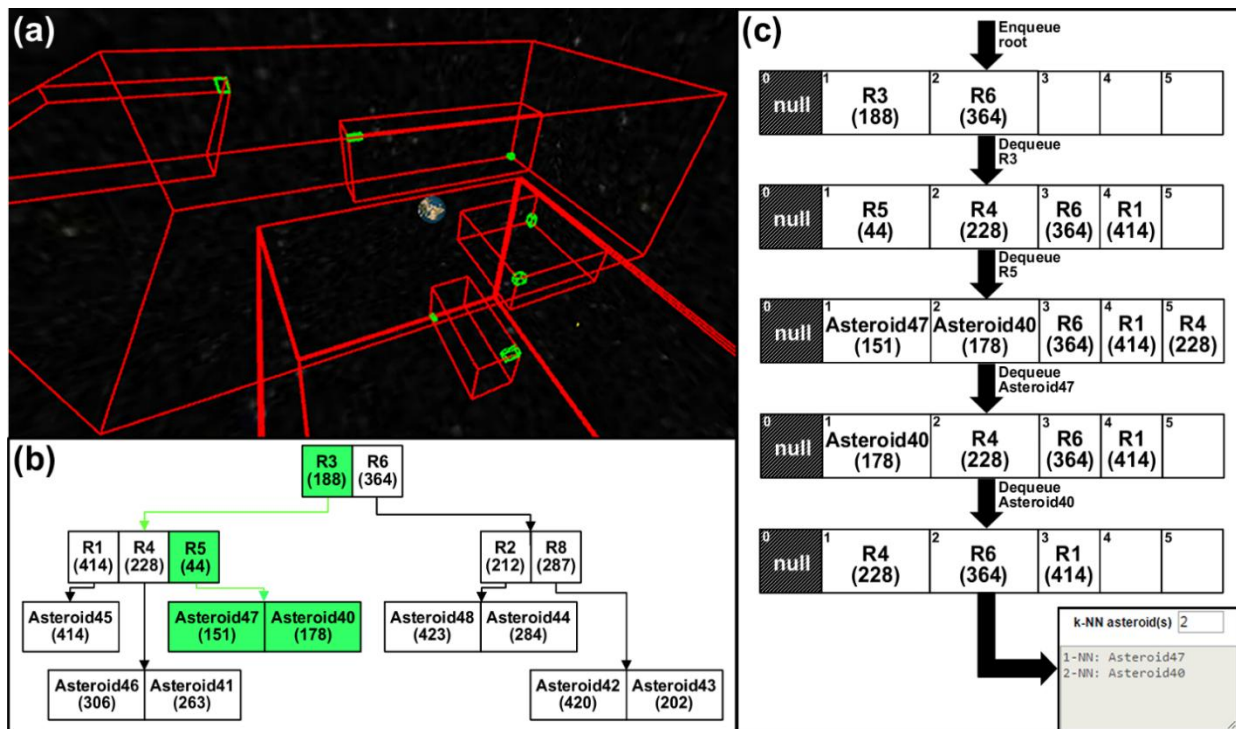


**Figure 6-5 The result set of a region query**

The same figure presents the subjected overlapping entries of the R-tree, which are none other than a subset of the spatial objects composing the X3D scene. In this use case, 5 x *T72* main battle tanks, 2 x *MEFFV2* armored vehicles and 2 x *M577* command post vehicles are considered as qualified records and they are returned to the user's query. Moreover, it is worth mentioning that the current R-tree made use of  $M=4$  and  $m=2$  for nodes' maximum and minimum filled requirement, respectively. So, there could exist various alternative solutions to the one displayed in Fig. 6-4(b), based on the fact that the geometric structure and the space utilization of the tree is clearly depending on these two values. However, any changes on nodes' size are only affect the

number of nodes that DFS algorithm has to visit, returning in that way an identical result set of spatial objects with lower or higher query complexity.

Finally, in the Fig. 6-6 below, there are various subfigures that depict the use case being spatially indexed along with its corresponding 2-NN execution steps. This use case is in fact a simple non-interactive game which has been developed to test the real-time capabilities of the implemented k-NN algorithm. The main concept of the game is defending the Earth from an incoming shower of 9 asteroids by launching nuclear bombs against the two closest of them. The defense module can be seen as a separate AI system composing of a maximum line of range and the k-NN algorithm. Each asteroid's position is ceaselessly recorded to an R-tree instance, which is latter queried in contrast with Earth's position, in order to calculate their correlated distance and deduct the two closest amongst them.



**Figure 6-6 Searching for 2-NN candidates with BFS algorithm**

The thumbnail image displayed in Fig. 6-6(a) follows a particular point of view being used for the purposes of this scenario, while the corresponding inquired R-tree instance can be seen in Fig. 6-6(b). The colored nodes of the graph denote the paths that have been followed from the k-NN algorithm during its 2-NN deduction procedure. The overall process is thoroughly described

in Fig. 6-6(c), where it becomes evident that the nodes are visited according to their distance compared to Earth's coordinates. Each one of them is further traversed until a spatial object is found, which is eventually returned as the nearest neighbor of the query point. The results were very satisfying in terms of speed and accuracy, taking into consideration not only that the scene was continuously receiving add/remove node requests, but it had also to update them in real time. Moreover, R-tree's performance and response were kept in acceptable levels, even though that this specific R-tree version has not been optimized for indexing moving objects. The latter one indicates that the implemented algorithm can also be a quite promising approach for detecting and indexing moving objects in 3D space, through slightly modifying its structure according to the requirements set by other domains.

In this subchapter, various use cases have been developed in order to present the spatial reasoning capabilities of the implemented queries. Each one of these cases corresponds to a carefully planned *sui generis* scenario, which has been attributed to a different spatial query type. The reasoning process was thoroughly explained in each query, while their relative results were checked against the expected outcomes. The latter ones were closely linked to the chosen filled requirements of the R-tree instance and the indexed objects of the underlying X3DOM scene. Generally, the implemented spatial query operations were found to be very accurate in their topological and distance calculations, while at the same time, their execution times were deemed very satisfying.

### **Implication of spatial relations**

Leaving aside the provided spatial queries, the implemented R-tree data structure has been further extended to support advanced spatial reasoning features. Such features come in the form of spatial predicates that are ultimately represented by topological and directional relations. Both of these spatial reasoning categories are based on the definition of a flexible computational model, which relies on various mathematical formulas for the deduction of the appropriate spatial relations between two objects in 3D space. In Fig. 6-7 below is depicted a spatial implication scenario, which involved the development of a realistically decorated indoor virtual environment. The implemented computational model takes advantage of specific indexed objects from a given R-tree instance, in order to automatically infer their corresponding spatial relations. This model -of course- is relied on the validation of a predefined set of spatial predicates that

come with certain constraints upon these objects' location. In the following paragraph, a unique use case has been composed for the sake of experimental evaluation, where its indexing and spatial reasoning procedures are thoroughly described.



**Figure 6-7 Automatic implication of spatial relations between three objects**

It quickly becomes clear that this scene provides numerous objects for indexing and spatial reasoning. However, a small only subset of the available objects was used in order to present an - as simple as possible- use case. This lies to the fact that the spatial indexing of the entire set would return hundreds of spatial relationships, complicating thereby this scenario's purpose and demonstration. For such reasons, three different objects were chosen and indexed into an R-tree instance, a *dining table*, a *colorful orb* and a *shelf*. These objects were preferred over the others due to their special placement into the 3D space. Their corresponding locations allow a straightforward presentation of the spatial reasoning capabilities provided by the implemented computational model. At first, each object is found to satisfy the *disjoint* topological relation, something that leads them to the deduction of their directional relations. For example, the dining table is found to be on the right side of the colorful orb and the shelf, while at the same time, it is placed in front of and below both of them. The inverse directional relationships, *left*, *behind* and *above*, are automatically applied into the colorful orb and shelf without proceeding to any kind of additional reasoning formula. On the other hand, the colorful orb and shelf share a common

boundary region, indicating a *touch* topological relation which is included into the mathematical formula comprising the implemented *disjoint* set. In this special occasion, the colorful orb is attributed with the *over* directional relation when it is compared to the shelf. The latter object remains to be below the former, keeping the already defined relations unchanged for the rest of the indexed objects. However, both of these two objects are positioned in a very particular way. Even though that they are found to be *disjoint* too, there is no other directional relation that can be satisfied for them, since their coordinates collection excludes the implication of a *left*, *right*, *front* or *behind* relationship. Finally, the complete set of the automatically generated spatial relations between these three objects is displayed in the same and sole figure of this subchapter.

Besides the before mentioned topological relations, *within* and *overlap* relations are extensively used during the construction of an R-tree instance or the execution of point and region queries. Moreover, there are still left a few topological relationships that have not been presented in experimental evaluation chapter, since their placement into the current 3D space is a bit paradoxical. Such relations are the *equal*, *within* and *contains*, which have been tested and found to be fully functional for any scenario, but their use is quite limited in the majority of them due to their complexity and unnecessary reasoning correlations. Finally, despite the fact that such relations are utilized by an ontological system with SPARQL support, the GUI approach presented for the purposes of this subchapter, it will significantly suffer in case of thousands or millions of implicated relations. In case that application's users desire to maintain such a solution, they have to come up with an efficient dynamic JavaScript grid library [75] [76] [77], which provides advanced features for real time manipulation of datasets from classifying and applying CSS themes upon them, to sorting and searching a finite subset of data.

## Chapter 7 - Conclusion

This thesis dealt with the problem of spatially reasoning a VR environment in X3DOM framework. A fast and memory-efficient R-tree data structure has been implemented for the indexing of 3D content from X3D authored environments. R-tree was chosen thanks to its wide acceptance in various domains, as one of the best techniques to handle multi-dimensional datasets. The proposed algorithm has been implemented in JavaScript language to preserve the independent nature of X3D standard and the plugin-less feature of X3DOM framework. R-tree provides a range of diverse operations –i.e. spatial queries and spatial relationships- that can benefit various applications, like GIS, CAD and multimedia. Moreover, the experimental evaluation results showed that it comes with a great medium between speed, reliability and practicality, where its indexing and spatial reasoning subtasks support amortized run times based on the dataset being tested.

By the same token, the agile architecture of the indexing algorithm makes feasible the spatial registration of dynamic objects, through the provision of a propagation mechanism tailored for tree topologies. The chosen objects are maintained under a common structure which may be initialized only once, but it is automatically updated each time an operation perturbs the spatial arrangement of its corresponding AABBs. This adjustment procedure can be seen as a self-balancing feature, which not only avoids expensive reinsertion methodologies, but it also guarantees R-tree's spatial reasoning validity. The latter one is backed by various mathematical and algorithmic optimizations that aim at reducing as much as possible the execution time of spatial queries. Especially in the case of k-NN queries, various alternative solutions were presented for the approximation of the distance between two objects, while a flexible binary heap data structure was employed to serve as a partially ordered priority queue.

Lastly, despite the fact that the authored scenes have to comply with the X3D language and the web-based X3DOM framework as the presentation middleware, the generic structure of the R-tree instance remains independent of the front-end data format. In this way, there are various alternatives for the presentation of the results coming from the spatial reasoning upon an R-tree instance.

## Chapter 8 - Future work

The implemented R-tree algorithm is heavily based on the classical R-tree data structure, since its construction complexity and query performance were deemed adequate for the 3D virtual environments being tested. However, in case of static environments with a large number of moving objects or interactive environments with deformable objects, a different heuristic splitting approach has to be adopted for the improvement of space utilization and the execution of spatial queries. A couple of significant research works [38] [78] introduced new spatial parameters or proposed light modifications upon the Quadratic algorithm, respectively, aiming to minimize the overlapping entries and the coverage factor. The aspects of such variants can be used as a starting point for the authoring of an appropriate splitting methodology, which could be able to sufficiently deal with the majority of 3D virtual environments.

Finally, the spatial relationships have been optimized to favor rectangular objects in contrast to other geometric shapes, where two different but interrelated points of interest bear the responsibility of this phenomenon. The first one lies to the fact that primary target of this work was a semantic representation of indoor environments. In this type of environments, a rectangular parallelepiped is the best fitting form of shape for the majority of 3D objects that can be found in it. The second factor comes to testify this empirical evidence by assigning to each R-tree's node a bounding box container, which is used to encapsulate a specific object and its surrounding space. However, there are virtual environments that originate from delicate domains with strict specifications that perform expensive geometric operations, from the collision detection of 3D objects to the selective rendering and management of complex spatial scenes. In these occasions, a large part of the data structure has to be remodeled to support an advanced object-oriented geometry container for the unconditional performance of such applications. A typical container of this kind could be the convex polyhedron, but the authoring of its algorithm has to be done quite carefully in order to avoid enormous computation time gaps.



## References

- [1] Pittarello, F., & De Faveri, A. (2006, April). Semantic description of 3D environments: a proposal based on web standards. In *Proceedings of the eleventh international conference on 3D web technology* (pp. 85-95). ACM.
- [2] Spala, P., Malamos, A. G., Doulamis, A., & Mamakis, G. (2012). Extending MPEG-7 for efficient annotation of complex web 3D scenes. *Multimedia Tools and Applications*, 59(2), 463-504.
- [3] Zampoglou, M., Spala, P., Kontakis, K., Malamos, A. G., & Ware, J. A. (2013, June). Direct mapping of X3D scenes to MPEG-7 descriptions. In *Proceedings of the 18th International Conference on 3D Web Technology* (pp. 57-65). ACM.
- [4] Cao, X., & Klusch, M. (2013, September). Advanced Semantic Deep Search for 3D Scenes. In *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on* (pp. 236-243). IEEE.
- [5] Bilasco, I. M., Gensel, J., Villanova-Oliver, M., & Martin, H. (2006, April). An MPEG-7 framework enhancing the reuse of 3D models. In *Proceedings of the eleventh international conference on 3D web technology* (pp. 65-74). ACM.
- [6] Zhang, X., Gračanin, D., & Matković, K. (2014, August). Using linked data for interactive 3D web content integration. In *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies* (pp. 147-147). ACM.
- [7] Brutzman, D., & Daly, L. (2010). *X3D: extensible 3D graphics for Web authors*. Morgan Kaufmann.
- [8] Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific american*, 284(5), 28-37.
- [9] Otto, K. A. (2005, May). Semantic virtual environments. In *Special interest tracks and posters of the 14th international conference on World Wide Web* (pp. 1036-1037). ACM.
- [10] Bille, W., Pellens, B., Kleinermann, F., & De Troyer, O. (2004). Intelligent Modelling of Virtual Worlds Using Domain Ontologies. *IVEVA*, 97.
- [11] Kleinermann, F., De Troyer, O., Mansouri, H., Romero, R., Pellens, B., & Bille, W. (2005). Designing semantic virtual reality applications. In *Proceedings of the 2nd INTUITION International Workshop, Senlis, France* (Vol. 61).
- [12] Grüninger, M., & Fox, M. S. (1995). Methodology for the Design and Evaluation of Ontologies.

- [13] Davies, J., Duke, A., & Sure, Y. (2003, October). OntoShare: a knowledge management environment for virtual communities of practice. In *Proceedings of the 2nd international conference on Knowledge capture* (pp. 20-27). ACM.
- [14] Halabala, P. (2003). Semantic metadata creation. In *Proceedings of 7th Central European Seminar on Computer Graphics CESC* (pp. 15-25).
- [15] Bilasco, I. M., Gensel, J., Villanova-Oliver, M., & Martin, H. (2005, November). On indexing of 3D scenes using MPEG-7. In *Proceedings of the 13th annual ACM international conference on Multimedia* (pp. 471-474). ACM.
- [16] Bilasco, I. M., Gensel, J., Villanova-Oliver, M., & Martin, H. (2005, December). 3DSEAM: a model for annotating 3D scenes using MPEG-7. In *Multimedia, Seventh IEEE International Symposium on* (pp. 10-pp). IEEE.
- [17] Bilasco, I. M., Gensel, J., Villanova-Oliver, M., & Martin, H. (2006, April). An MPEG-7 framework enhancing the reuse of 3D models. In *Proceedings of the eleventh international conference on 3D web technology* (pp. 65-74). ACM.
- [18] Figueroa, P., Green, M., & Hoover, H. J. (2002, February). InTml: a description language for VR applications. In *Proceedings of the seventh international conference on 3D Web technology* (pp. 53-58). ACM.
- [19] Lenne, D., Thouvenin, I., & Aubry, S. (2009). Supporting design with 3D-annotations in a collaborative virtual environment. *Research in engineering design*, 20(3), 149-155.
- [20] Chevaillier, P., Trinh, T. H., Barange, M., De Loor, P., Devillers, F., Soler, J., & Querrec, R. (2012, March). Semantic modeling of virtual environments using mascaret. In *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2012 5th Workshop on* (pp. 1-8). IEEE.
- [21] De Luca, L., Véron, P., & Florenzano, M. (2005). Semantic-based modelling and representation of patrimony buildings. In *SVE Worksop towards Semantic Virtual Environments* (pp. 1-11).
- [22] Kosteljik, T. (2012). *Semantic annotation of urban scenes: Skyline and window detection* (Doctoral dissertation, Universiteit van Amsterdam).
- [23] Nüchter, A., Surmann, H., Lingemann, K., & Hertzberg, J. (2003). Semantic Scene Analysis of Scanned 3D Indoor Environments. In *VMV* (pp. 215-221).
- [24] Elseberg, J., Magnenat, S., Siegwart, R., & Nüchter, A. (2012). Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. *Journal of Software Engineering for Robotics*, 3(1), 2-12.
- [25] Shao, T., Xu, W., Zhou, K., Wang, J., Li, D., & Guo, B. (2012). An interactive approach to semantic modeling of indoor scenes with an rgbd camera. *ACM Transactions on Graphics (TOG)*, 31(6), 136.

- [26] Maria Del Carmen Molla Garcia. (2013). Describing scenes by qualitative spatial relations (Master's thesis, Royal Institute of Technology, Stockholm, Sweden). Retrieved from <http://www.diva-portal.org/smash/get/diva2:699637/FULLTEXT01.pdf>
- [27] Strobl, C. (2008). Dimensionally Extended Nine-Intersection Model (DE-9IM). In *Encyclopedia of GIS* (pp. 240-245). Springer US.
- [28] Kontakis, K., Steiakaki, M., Kapetanakis, K., & Malamos, A. G. (2014, August). DEC-O: an ontology framework and interactive 3D interface for interior decoration applications in the web. In *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies* (pp. 63-70). ACM.
- [29] Kontakis, K., Steiakaki, M., Kalochrsitianakis, M., Kapetanakis, K., & Malamos, A. G. (2015). Applying Aesthetic Rules in Virtual Environments by Means of Semantic Web Technologies. In *Augmented and Virtual Reality* (pp. 344-354). Springer International Publishing.
- [30] Behr, J., Jung, Y., Keil, J., Drevensek, T., Zoellner, M., Eschler, P., & Fellner, D. (2010, July). A scalable architecture for the HTML5/X3D integration model X3DOM. In *Proceedings of the 15th International Conference on Web 3D Technology* (pp. 185-194). ACM.
- [31] Limper, M., Thöner, M., Behr, J., & Fellner, D. W. (2014, August). SRC-a streamable format for generalized web-based 3D data transmission. In *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies* (pp. 35-43). ACM.
- [32] Wikipedia – The Free Encyclopedia. (2015). JavaScript. Retrieved from <http://en.wikipedia.org/wiki/JavaScript>
- [33] Teschner, M., Kimmerle, S., Heidelberger, B., Zachmann, G., Raghupathi, L., Fuhrmann, A., ... & Volino, P. (2005, March). Collision detection for deformable objects. In *Computer graphics forum* (Vol. 24, No. 1, pp. 61-81). Blackwell Publishing Ltd.
- [34] Comer, D. (1979). Ubiquitous B-tree. *ACM Computing Surveys (CSUR)*, 11(2), 121-137.
- [35] Guttman, A. (1984). *R-trees: a dynamic index structure for spatial searching* (Vol. 14, No. 2, pp. 47-57). ACM.
- [36] Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A. N., & Theodoridis, Y. (2003). *R-trees have grown everywhere*. Technical Report available at <http://www.rtreeportal.org>.
- [37] Balasubramanian, L., & Sugumaran, M. (2012). A state-of-art in R-tree variants for spatial indexing. *International Journal of Computer Applications*, 42(20), 35-41.

- [38] Beckmann, N., Kriegel, H. P., Schneider, R., & Seeger, B. (1990). *The R\*-tree: an efficient and robust access method for points and rectangles* (Vol. 19, No. 2, pp. 322-331). ACM.
- [39] AnandhaKumar, P., Priyadarshini, J., Monisha, C., Sugirtha, K., & Raghavan, S. (2010, August). Location Based Hybrid Indexing Structure-R kd Tree. In *Integrated Intelligent Computing (ICIIC), 2010 First International Conference on* (pp. 140-145). IEEE.
- [40] Li, G., & Tang, J. (2010, July). A new DR-tree K-nearest neighbor query algorithm based on direction relationship. In *Environmental Science and Information Application Technology (ESIAT), 2010 International Conference on* (Vol. 2, pp. 246-250). IEEE.
- [41] White, D., & Jain, R. (1996, February). Similarity indexing with the SS-tree. In *Data Engineering, 1996. Proceedings of the Twelfth International Conference on* (pp. 516-523). IEEE.
- [42] García, Y. J., Lopez, M. A., & Leutenegger, S. T. (1998, August). On optimal node splitting for R-trees. In *Proceedings of the 24rd International Conference on Very Large Data Bases* (pp. 334-344). Morgan Kaufmann Publishers Inc.
- [43] Zhu, Q., Gong, J., & Zhang, Y. (2007). An efficient 3D R-tree spatial index method for virtual geographic environments. *ISPRS Journal of Photogrammetry and Remote Sensing*, 62(3), 217-224.
- [44] Greene, D. (1989, February). An implementation and performance analysis of spatial data access methods. In *Data Engineering, 1989. Proceedings. Fifth International Conference on* (pp. 606-615). IEEE.
- [45] Ang, C. H., & Tan, T. C. (1997, January). New linear node splitting algorithm for R-trees. In *Advances in Spatial Databases* (pp. 337-349). Springer Berlin Heidelberg.
- [46] Achtert, E., Kriegel, H. P., & Zimek, A. (2008, January). ELKI: a software system for evaluation of subspace clustering algorithms. In *Scientific and Statistical Database Management* (pp. 580-585). Springer Berlin Heidelberg.
- [47] Kao, B., Lee, S. D., Lee, F. K., Cheung, D. W. L., & Ho, W. S. (2010). Clustering uncertain data using voronoi diagrams and r-tree index. *Knowledge and Data Engineering, IEEE Transactions on*, 22(9), 1219-1233.
- [48] Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., ... & Steinberg, D. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1), 1-37.
- [49] Sankaranarayanan, J., Samet, H., & Varshney, A. (2007). A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computers & Graphics*, 31(2), 157-174.

- [50] Zhao, J., Long, C., Xiong, S., Liu, C., & Yuan, Z. (2013). A New K Nearest Neighbours Algorithm Using Cell Grids for 3D Scattered Point Cloud. *Elektronika ir Elektrotechnika*, 20(1), 81-87.
- [51] Roussopoulos, N., Kelley, S., & Vincent, F. (1995, June). Nearest neighbor queries. In *ACM sigmod record* (Vol. 24, No. 2, pp. 71-79). ACM.
- [52] Hjaltason, G. R., & Samet, H. (1999). Distance browsing in spatial databases. *ACM Transactions on Database Systems (TODS)*, 24(2), 265-318.
- [53] Eberly, D. (2010). *Fast inverse square root (revisited)*. Technical report, Geometric Tools, LLC.
- [54] Weisstein, Eric W. "Newton's Method." From *MathWorld*--A Wolfram Web Resource. <http://mathworld.wolfram.com/NewtonsMethod.html>
- [55] Tsai, C. F., & Lin, C. Y. (2010). A triangle area based nearest neighbors approach to intrusion detection. *Pattern Recognition*, 43(1), 222-229.
- [56] Banerjee, A., Ghosh, A., & Das, M. (2015). High Performance Novel Square Root Architecture Using Ancient Indian Mathematics for High Speed Signal Processing. *Advances in Pure Mathematics*, 5(08), 428.
- [57] AUTODESK Knowledge Network. (2015). Data Model: Spatial Relationship Settings. Retrieved from <http://knowledge.autodesk.com/support/infrastructure-map-server/learn-explore/caas/CloudHelp/cloudhelp/2016/ENU/MapServer-Help/files/GUID-C628E985-0F7A-4FE3-B2C3-27630396E551-htm.html>
- [58] Koranne, S. (2011). Boost C++ libraries. In *Handbook of Open Source Tools* (pp. 127-143). Springer US.
- [59] Kothuri, R., Beinat, E., & Godfrind, A. (2004). *Pro oracle spatial*. Apress.
- [60] Piórkowski, A. (2011). Mysql spatial and postgis–implementations of spatial data standards. *Electronic Journal of Polish Agricultural Universities*, 14(1), 1-8.
- [61] Daum, S., & Borrmann, A. (2014). Processing of Topological BIM Queries using Boundary Representation Based Methods. *Advanced Engineering Informatics*, 28(4), 272-286.
- [62] Mark, D. M., & Egenhofer, M. J. (1994). Modeling spatial relations between lines and regions: combining formal mathematical models and human subjects testing. *Cartography and geographic information systems*, 21(4), 195-212.
- [63] Papadias, D., Sellis, T., Theodoridis, Y., & Egenhofer, M. J. (1995). *Topological relations in the world of minimum bounding rectangles: a study with R-trees* (Vol. 24, No. 2, pp. 92-103). ACM.

- [64] Freeman, J. (1975). The modelling of spatial relations. *Computer graphics and image processing*, 4(2), 156-171.
- [65] Johannsen, K., Swadzba, A., Ziegler, L., Wachsmuth, S., & De Ruiter, J. P. (2013). A Computational Model for Reference Object Selection in Spatial Relations. In *Spatial Information Theory* (pp. 358-376). Springer International Publishing.
- [66] Hayward, W. G., & Tarr, M. J. (1995). Spatial language and spatial representation. *Cognition*, 55(1), 39-84.
- [67] Schwering, A., & Raubal, M. (2005). *Spatial relations for semantic similarity measurement* (pp. 259-269). Springer Berlin Heidelberg.
- [68] Herhut, S., Hudson, R. L., Shpeisman, T., & Sreeram, J. (2012, June). Parallel programming for the web. In *Proceedings of the 4th USENIX conference on Hot Topics in Parallelism, HotPar* (Vol. 12, p. 1).
- [69] WHATWG HTML Living Standard. (2015). Web workers. Retrieved from <https://html.spec.whatwg.org/multipage/workers.html#workers>
- [70] Stell, J. G. (2004). Part and complement: Fundamental concepts in spatial relations. *Annals of Mathematics and Artificial Intelligence*, 41(1), 1-17.
- [71] Kong, H., Jung, K., Choi, J., Kim, W., Kim, P., & Park, J. (2003). Representing the spatial relations in the semantic web ontologies. In *AI 2003: Advances in Artificial Intelligence* (pp. 77-87). Springer Berlin Heidelberg.
- [72] McGuinness, D. L., & Van Harmelen, F. (2004). OWL web ontology language overview. *W3C recommendation*, 10(10), 2004.
- [73] O'Connor, M. J., & Das, A. K. (2009, October). SQWRL: A Query Language for OWL. In *OWLED* (Vol. 529).
- [74] W3C Recommendation for "High Resolution Time" (2012). Performance Interface. Retrieved from <http://www.w3.org/TR/hr-time/>
- [75] Manricks, G. (2013). *Instant JqGrid*. Packt Publishing Ltd.
- [76] DataTables. (2015). Table plug-in for jQuery. Retrieved from <http://datatables.net/>
- [77] Slickgrid. (2014). A lighting fast JavaScript grid/spreadsheet. Retrieved from <https://github.com/mleibman/SlickGrid/wiki>
- [78] Al-Badarneh, A. F., Yaseen, Q., & Hmeidi, I. (2010). A new enhancement to the R-tree node splitting. *Journal of Information Science*, 36(1), 3-18.

## Appendix A - Glossary of Terms

**SPARQL (SPARQL Protocol and RDF Query Language):** a Semantic Web standard and the most widely used query language for RDF datasets. Its latest protocol provides improved performance and advanced retrieval capabilities, thanks to a set of unique features like SPARQL algebra, custom filter functions, aggregation, various storage systems support, etc.

**XSLT (Extensible Stylesheet Language Transformations):** a language that comes with a very strict vocabulary for the generation of an XML document, based on a specific formatting and a source XML-based document.

**ICP (Iterative Closest Point):** a closest point approximation algorithm which returns the most optimal paths between clouds of points. Its efficiency and accuracy can be further improved by consecutively repeating its algorithmic procedure.

**RANSAC (Random Sample Consensus):** an iterative approach for the estimation of a mathematical model's outcome based on the manipulation of a given dataset. It is still an active area of research in computer vision domain.

**WebGL:** a novel OpenGL ES 2.0 approach for the rendering of 2D and 3D content, natively and plugin-less in any typical browser and device. It provides a JavaScript API and is totally independent of the underlying platform and operating system.

**BST (Binary Search Tree):** a tree data structure which boosts common tree operations by defining that each non-leaf node must have at most two child nodes. Amongst them, the left node holds a value less than the value of its parent, while the right node holds a value greater than the value of its parent.

**OBB (Oriented Bounding Box):** an arbitrary oriented bounding box which makes use of heuristic methodologies to calculate its rotation and perimeter based on its enclosed object or set of objects.

**MBR (Minimum Bounding Rectangle/Region):** is defined as the smallest possible rectangular area that successfully covers the total area of its node's entries. The same algorithmic pattern is followed for any type of entry, whether it is an internal node, leaf node, or spatially indexed record.

**Moore neighborhood:** a widely used neighborhood algorithm for games and graphics editors, which surrounds a target area with a predefined number of its cuboid siblings according to the cellular automata theory.

**Bounding Container:** a methodology which applies for the inclusion of a geometric object in a closed volume to improve the runtime speed of computationally expensive operations, like collision detection or ray tracing.

**Level-order traversal:** a breadth-first tree traversal which visits every node level by level. It starts from the root node and continues to its direct child nodes. Then, it traverses every grandchildren, great grandchildren, and so goes on until all nodes have been successfully traversed.

**Pre-order traversal:** a depth-first tree traversal which visits every parent node before its children. It starts from the root node and then are recursively traversed the nodes of the left subtrees before the nodes consisting the right subtrees.

**OGC (Open Geospatial Consortium):** an alliance of international organizations responsible for developing data mining services and implementing interface specifications, which are ultimately made available as open standards for geospatial information applications.

**Vague region:** a term which is used to imply that an object's boundary alone, does not suffice to spatially categorize it into a specific region. This uncertainty is usually overcome with the introduction of additional spatial factors.



**RCC (Region Connection Calculus):** an alternative -and contradictory to DE-9IM- spatial reasoning methodology, which segments the search space into various spatial regions based on a fixed set of topological relations.

**ElevationGrid:** a special type of geometry defined in X3D standard for the generation of polymorphic terrains. Its diversity is based on the definition of a set of grids, which are attributed with a height value on specific row and column coordinates.