

# **Smartphone 3D Reconstruction for Physical Interactions in Augmented Reality**

by

**Manuel Gil Martinez, B.Sc.**

**Dissertation**

Presented to the

University of Dublin, Trinity College

in fulfilment

of the requirements

for the degree of

**Master of Science in Computer Science**

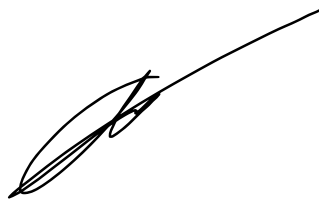
**(Graphics & Vision Technologies)**

**University of Dublin, Trinity College**

September 2018

## Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

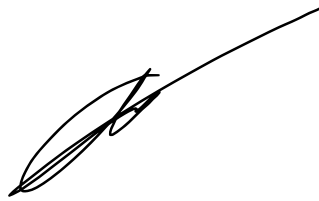
A handwritten signature in black ink, consisting of a stylized 'M' and 'G' followed by 'Martinez', written over a horizontal line.

Manuel Gil Martinez

August 27, 2018

## Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

A handwritten signature in black ink, appearing to read 'Manuel Gil Martinez', written over a horizontal line.

Manuel Gil Martinez

August 27, 2018

# Acknowledgments

Firstly, I would like to express my sincere gratitude to my supervisor at Trinity College of Dublin, Dr. John Dingliana, for his invaluable support and guidance through this project.

I am also grateful to the Trinity College of Dublin for putting at my disposal all the resources and facilities, and to the teaching staff and fellow classmates who made this Master of Science possible. Thank you for providing me with such an enjoyable learning experience.

Special thanks to Agnese, for her inspiring help and encouragement in both good and not-so-good moments.

Finally, I would like to thank my family for the motivation and immense support during the last year, with a special mention to my sister Adriana.

MANUEL GIL MARTINEZ

*University of Dublin, Trinity College  
September 2018*

# Smartphone 3D Reconstruction for Physical Interactions in Augmented Reality

Manuel Gil Martinez, Master of Science in Computer Science  
University of Dublin, Trinity College, 2018

Supervisor: Dr. John Dingliana

Over the last couple of years, Augmented Reality (AR) has reached millions of consumers with the help of Visual-Inertial Odometry systems. This key process has been developed to such extent that it can now run on handheld smartphones. Research in the field has been focused on creating usable and entertaining experiences, but there is still a need to design solutions addressed at some crucial technical obstacles of Augmented Reality systems. Nowadays, frameworks such as ARKit or ARCore are available in millions of devices and allow a basic interaction between virtual objects and real-world planes, lacking any other 3D understanding of the rest of objects present in the scene.

The project hereby presented aims to design and implement a real-time 3D reconstruction system for a richer environmental understanding of the physical world, focusing on allowing an efficient collision between real and virtual objects. Some solutions have been proposed using specific hardware. This work, however, is developed on top of monocular SLAM systems which are available in today's mobile devices, thus, the constraints that these low-end devices have must be taken into consideration. This dissertation seeks to describe how to make use of a cloud of visual features in order to build a meaningful 3D representation of the scene. The results of this research extend the alternatives currently available, and show how real-time can be achieved by taking advantage of a volumetric data structure with adaptive levels-of-detail and a multithreaded processing pipeline.

# Contents

<b>Acknowledgments</b>	<b>iii</b>
<b>Abstract</b>	<b>iv</b>
<b>List of Tables</b>	<b>vii</b>
<b>List of Figures</b>	<b>viii</b>
<b>List of Abbreviations</b>	<b>x</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 Objectives . . . . .	3
1.3 Dissertation Roadmap . . . . .	4
<b>Chapter 2 State of the Art</b>	<b>6</b>
2.1 Augmented Reality . . . . .	6
2.1.1 Mobile Augmented Reality . . . . .	8
2.1.2 Immersion . . . . .	13
2.2 3D Scene Reconstruction . . . . .	13
2.2.1 Spatial Hashing . . . . .	15
2.2.2 Voxel Hashing . . . . .	16
2.2.3 Hierarchical Data Structures . . . . .	18
2.3 Collision Detection . . . . .	19
2.3.1 AR Collisions . . . . .	20
2.4 Level of Detail (LOD) . . . . .	22
<b>Chapter 3 Adaptive Volumetric Representation</b>	<b>23</b>

3.1	Scene Understanding in AR Frameworks . . . . .	23
3.2	Basic Voxel-Hashing . . . . .	25
3.3	LOD Data Structure . . . . .	26
3.3.1	Confidence Value . . . . .	29
3.4	Hash Operations . . . . .	29
3.5	Reducing Tracked Objects . . . . .	30
<b>Chapter 4 Object Reconstruction Pipeline</b>		<b>32</b>
4.1	Point Cloud Input . . . . .	32
4.2	Processing the Point Cloud . . . . .	33
4.2.1	Building different Levels of Detail . . . . .	33
4.3	Boosting real-time interaction . . . . .	35
4.4	Reconstruction Output . . . . .	36
<b>Chapter 5 Implementation</b>		<b>37</b>
5.1	Application . . . . .	37
5.1.1	Physical Interaction . . . . .	39
5.2	Framework . . . . .	39
5.3	Rendering . . . . .	40
5.3.1	Occlusion of virtual objects . . . . .	40
<b>Chapter 6 Results &amp; Discussion</b>		<b>42</b>
6.1	Quality of the Reconstruction and Interaction . . . . .	42
6.2	Performance . . . . .	47
6.2.1	Unity3D . . . . .	49
6.2.2	Memory Footprint . . . . .	50
6.3	Discussion and Limitations . . . . .	51
<b>Chapter 7 Conclusions &amp; Future Work</b>		<b>54</b>
7.1	Conclusions . . . . .	54
7.2	Future Work . . . . .	55
<b>Bibliography</b>		<b>57</b>
<b>Appendices</b>		<b>64</b>

# List of Tables

Table 6.1	Results of processing a point cloud into a adaptive hash table of voxels. . . . .	46
Table 6.2	Comparison of anchor update times between our hierarchical data structure and plain Voxel-Hashing. . . . .	48
Table 6.3	Comparison between processing 100 points with one thread and with multiple threads. . . . .	49
Table 6.4	Memory allocated with our Adaptive Data Structure activated and deactivated. Number of voxels in different resolutions and initial feature points processed are also listed. . . . .	51



# List of Figures

Figure 2.1	Virtuality Continuum by Milgram in 1994 [44] . . . . .	7
Figure 2.2	a) IKEA’s new app Place [18]. b) Augmented Reality information overlaid in healthcare applications [31]. . . . .	8
Figure 2.3	VIO systems are possible thanks to the combination of SLAM and sensors. . . . .	11
Figure 2.4	Google’s ARCore processing pipeline. Our dissertation topic deals with “Object Detection”, highlighted in red. . . . .	12
Figure 2.5	a) Example of a 3D-reconstructed scene with Kinect Fusion [46]. b) Truncated signed-distance function using a Kinect as scanning device [17]. . . . .	15
Figure 2.6	a) The Voxel-Hashing data structure proposed in [47]. b) Results of 3D reconstruction with Voxel-Hashing [47]. . . . .	17
Figure 2.7	a) 3D reconstruction of a scene using Chisel [34]. b) Chisel application based on <i>Project Tango</i> [40]. . . . .	18
Figure 2.8	a) A wireframe model is overlaid on a real object after registration [7]. b) The model is used for occlusion and collision of digital objects (two chairs and one lamp) [7]. . . . .	21
Figure 3.1	Our Adaptive LOD data structure for storing different resolutions of the scene. Feature points are the starting data needed for building a hierarchical hash table of voxels. . . . .	28
Figure 3.2	Features points contained in the same LOD-0 (grey voxel) can be in different subvoxels for the different resolutions. . . . .	29
Figure 4.1	Our multithread processing pipeline organises threads based on the hash values of the points in order to avoid hash table collisions. . .	33

## LIST OF FIGURES

---

Figure 5.1	By moving the phone around the scene, visual features are detected and voxels start to appear to represent the 3D scene. . . . .	38
Figure 5.2	Custom Unity3D user interface for configuring the reconstruction process. . . . .	40
Figure 6.1	a) Number of voxels and subvoxels reconstructed when scanning a scene with a single object. b) Number of voxels and subvoxels reconstructed when scanning multiple objects. The results are divided into the three different resolutions that were scanned. . .	43
Figure 6.2	a) Level of detail 0 with $10\text{cm}^3$ voxels. b) Level of detail 1 with $5\text{cm}^3$ voxels. c) Level of detail 2 with $2.5\text{cm}^3$ voxels. d) Real object with only a plane being rendered. e) All resolutions rendered simultaneously. . . . .	44
Figure 6.3	a) Level of detail 0 with $10\text{cm}^3$ voxels. b) Level of detail 1 with $5\text{cm}^3$ voxels. c) Level of detail 2 with $2.5\text{cm}^3$ voxels. d) Only table plane is being rendered. e) All resolutions for the reconstruction rendered simultaneously. . . . .	45
Figure 6.4	Duration of the update of all the anchors in the scene by frame. .	47
Figure 6.5	Duration of the calculations of the physics update per frame by Unity3D based on how many LOD-0 objects are reconstructed. . .	49
Figure 6.6	Duration of the frame rendering process in relation to the number of LOD-0 voxels reconstructed. . . . .	50

# List of Abbreviations

<b>3D</b>	—	Three Dimensions
<b>AABB</b>	—	Axis Aligned Bounding Box
<b>AR</b>	—	Augmented Reality
<b>ATAP</b>	—	Advanced Technology and Projects
<b>BVH</b>	—	Bounding Volume Hierarchy
<b>CAD</b>	—	Computer Assisted Design
<b>CPU</b>	—	Central Processing Unit
<b>CV</b>	—	Computer Vision
<b>DOF</b>	—	Degrees of Freedom
<b>FOV</b>	—	Field of View
<b>GPS</b>	—	Global Positioning System
<b>GPU</b>	—	Graphics Processing Unit
<b>HMAR</b>	—	Handheld Mobile Augmented Reality
<b>IMU</b>	—	Interior Measuring Unity
<b>iOS</b>	—	Iphone Operating System
<b>KD</b>	—	K-Dimensional
<b>LOD</b>	—	Level of Detail
<b>MAR</b>	—	Mobile Augmented Reality
<b>MR</b>	—	Mixed Reality
<b>OBB</b>	—	Oriented Bounding Box
<b>OS</b>	—	Operating System
<b>PC</b>	—	Personal Computer
<b>RGB-D</b>	—	Red Green Blue - Depth
<b>RGB</b>	—	Red Green Blue

## LIST OF ABBREVIATIONS

---

<b>SDF</b>	—	Signed Distance Function
<b>SDK</b>	—	Software Development Kit
<b>SLAM</b>	—	Simultaneous Localisation and Mapping
<b>TSDF</b>	—	Truncated Signed Distance Function
<b>VIO</b>	—	Visual-Inertial Odometry
<b>VR</b>	—	Virtual Reality

# Chapter 1

## Introduction

Augmented Reality (AR) is not a new concept anymore. Numerous books and movies of the last decades have shown us the once utopic immersion into a computer-generated environment. Even if we still have a long road ahead to achieve this vision, where the overlaid information presented to the user will be impossible to discern from the real world, we are getting closer day after day. Researchers, in some cases maybe inspired by science fiction, usually try to find solutions to create a seamless integration between the real and the virtual world. In general, most of these solutions seek to improve the current hardware, the visual perception of the virtual objects, or the physical interactions, in order increase the believability of any AR experience.

Thanks to the progress made in the Computer Vision and Machine Learning fields, Augmented Reality has been brought to million of commercial mobile phones. It is now more accessible than ever, and end-users are able to experiment and have a sneak-peek of how the future will look like through their hand-size devices.

This dissertation presents a novel approach that allows interaction between real-world environments and virtual objects in Mobile Augmented Reality, focusing on improving the current 3D understanding of the scene around us. Our approach is based in two main contributions: an adaptive data structure and a multithreaded processing pipeline.

## 1.1 Motivation

In recent years, the industry of Augmented Reality has experienced an exponential growth; predictions say this increase will not stop any time soon [5]. This technology is currently being used in commercial sectors such as telecommunication, healthcare, education, and entertainment among others. Since device manufactures are able to pack more power into smaller systems, we have seen the increment of new AR-capable hardware in the last couple of years. In 2016, HoloLens was presented by Microsoft as a self-contained, holographic computer, that enables the user to engage with digital content and interact with holograms placed around them [43]. Further examples of headsets include the Meta II or Magic Leap One, which are mainly focused on enhancing work productivity by overlaying information relevant to the user. Even if these devices are capable of an impressive spatial mapping thanks to their power and sensors, their price ranging from \$1,500 to \$5,000 make them an unnecessary whim for the general public.

While we wait for these headsets to be more affordable and useful, Google and Apple have released in the last months frameworks that enable augmented reality capabilities in their mobile operating systems, i.e. Android and iOS respectively. Both frameworks, called ARCore and ARKit, have brought augmented reality to millions of consumers and developers world-wide, removing some of the barriers that other types of hardware have. However, making AR as accessible as it is right now entails some drawbacks: mobile devices are very limited when it comes to power and sensors.

Fundamentally, mobile Augmented Reality is capable of two things: tracking the pose –rotation and location– of the device in real-time and have a basic understanding of the scene around the user. Due to the lack of depth sensors in modern smartphones, the 3D reconstruction of the real world is carried out by only representing the close planes available and completely omitting the remaining objects present in the scene. Therefore, when a virtual object is thrown into a mobile AR experience it will only collide with planes that have already been detected previously, limiting the overall immersion of the user.

Although immersion in AR can be lost by the visual quality of the rendered objects, or in other cases by the hardware used, the fidelity of physical interactions also play a big role on the quality of the experience. Likewise, reconstructing the scene is not a trivial problem, nonetheless proposing solutions is crucial for enhancing the user perception by providing accurate collisions with the real world. Having a richer scene understanding will not only increase the quality of the experience, but it will also be helpful for developers to build apps and games that make use of the totality of the 3D space they are run at.

Even though 3D reconstruction is an established research area within the Computer Vision field, most of the methods proposed in the past relied on GPU-based pipelines and depth maps as input. Nevertheless, these features are not available on regular smartphones, hence brand new approaches need to be implemented. The constraints of the type of device used do not limit our research, but help us find new and creative solutions that can be extrapolated to other devices later on.

## 1.2 Objectives

The primary aim of this dissertation is to describe, analyse and build an online 3D reconstruction system that is capable of efficiently representing the scene for physical interactions. The device used for development is a consumer-grade smartphone, therefore the solutions proposed should respect the constraints of these kind of devices. The current capabilities of handheld smartphone Augmented Reality are limited to detection and understanding of planes in the scene. Extending that functionality will improve the perception and therefore the overall quality of the user's experience.

Along with the primary goal, the reconstruction system pursues the following secondary objectives:

- To extend the current understanding functionalities of mobile AR, so that more objects apart from the already detected planes are spatially mapped on the scene allowing physical interaction with digital objects.
- To manage CPU in a performant way taking into account the limitations that

GPUs on mobile devices have. This will be done by designing a multi-thread implementation to reduce the workload of the main thread.

- To suggest an efficient, real-time system able to run simultaneously with the internal AR processes that track the motion, rotation and location of the device as well as the rest of digital objects present on the scene.
- In order to meet all the above, the data structure needed will be designed. This structure allows the storage of a volumetric representation of the world with various levels of detail (LODs), focusing on adaptiveness, and increasing the overall accuracy of the physical interactions between real and virtual world.

The aforementioned challenges are explained and reasoned throughout this dissertation and a solution for each of them will be presented. The project and all its code has been designed taking into account extensibility, and has been exported afterwards into a Unity3D package that can be used by other developers to encourage further research and applications. This dissertation and its design choices aim not only to improve the quality of the experiences in the fields of entertainment and videogames, but also to expand new areas of knowledge that take advantage of Augmented Reality and spatial mapping.

### 1.3 Dissertation Roadmap

This dissertation is arranged as follows:

- **Chapter 2 - State of the Art** overviews the background needed to understand the research being done and reviews the current state of the art by taking a look at previous related work. Then, it will examine why the already existing potential solutions do not solve the challenges mentioned in Section 1.2, and therefore why new approaches should be encouraged.
- **Chapter 3 - Adaptive Volumetric Representation** offers an in-depth view of the design choices that are the foundation of our adaptive data structure.
- **Chapter 4 - Object Reconstruction Pipeline** describes the algorithm and multithread process behind the 3D reconstruction.



- **Chapter 5 - Implementation** analyses the implementation possibilities of the data structure and pipeline presented in Chapters 3 & 4. The tools and libraries used are also described and discussed in this section.
- **Chapter 6 - Results & Discussion** evaluates the outcomes achieved in the implementation of an adaptive 3D reconstruction and exposes some of the limitations that the system has. The results are compared in regards to the current capabilities of smartphone-based Augmented Reality.
- **Chapter 7 - Conclusions & Future Work** summarises the proposed contributions and suggests ideas for future work.

# Chapter 2

## State of the Art

This chapter describes the current state of the art as well as the background knowledge needed to understand the underlying concepts that shape this dissertation. It starts by exploring Augmented Reality and more specifically the technologies that enable AR experiences on handheld smartphones. Next, an in-depth review of the related work being carried out in three-dimensional reconstruction is made, including the different ways of representing the scene data volumetrically. Finally, we look briefly into collision detection in Physics, and how efficient physics systems are developed.

### 2.1 Augmented Reality

Augmented Reality (AR) is usually defined as the technology that seamlessly overlays information on top of the real world to enable new interactive experiences [15]. Moreover, Augmented Reality is mentioned in literature as a variation of Virtual Reality (VR) [2]. While VR immerses the user inside a synthetic environment, AR superimposes the virtual objects into the real world, supplementing reality instead of replacing it. Milgram and Kishino in 1994 [44] formulate the taxonomy of the different ways in which real and virtual worlds can be combined, proposing clear terminologies for the Mixed Reality concepts. In their paper *A Taxonomy of Mixed Reality Visual Displays*, Augmented Reality is referred as all the cases in which “a display of an otherwise real environment is augmented by means of virtual (computer graphic) objects”. They classify Augmented Reality as a subset of Mixed Reality (MR), where real world and virtual

world objects are presented together within a single display. Figure 2.1 below exposes the relations between the different terminologies inside the “virtuality continuum”.

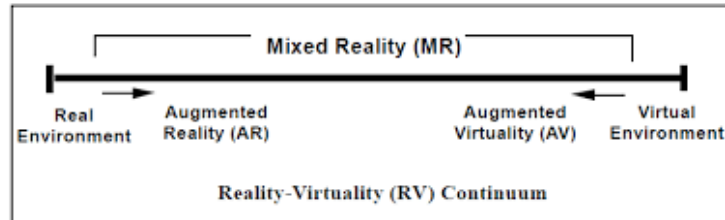


Figure 2.1: Virtuality Continuum by Milgram in 1994 [44]

Even if the first combination of virtual and real world views appeared in 1968 with the use of helmet-mounted video screens [59], the term Augmented Reality was not used until 1992. That year, Boeing Computer Services [10] designed and prototyped a see-through, head-mounted display (HUDSET) and first used the term “augmentation”, which referred to the enhancement of the visual field of view (FOV) of the user by overlaying relevant information for their current task. Overtime, Augmented Reality hardware has evolved, increasing the quality of immersion of the user and therefore, making it possible for the emergence of additional consumer, industrial, and commercial applications.

It is worth mentioning that Augmented Reality is radically changing the general perception and prior considerations about Human-Computer Interaction. Technology is now able to understand our most human inputs like hand gestures [63], gaze [51], or even voice commands [43] and therefore, is enabling a more natural way of computing. In the near future, Augmented Reality will play a big role on how we interact with computers, impacting human life and extending the boundaries of what are we able to do with technology [5].

AR is not limited to a set of specific applications; in fact, several studies [2, 6, 61, 39] investigate about the various use-cases for Augmented Reality in several industries. Researchers and developers constantly try to find fields in which overlaying virtual information in the real world can be beneficial for the users. Although two of the most common applications for AR are, undoubtedly, entertainment and videogames, other

areas such as healthcare, Computer Assisted Design (CAD) or maintenance among others also take advantage of this technology. For example, the display of information relevant to a surgery procedure above the patient can help nurses and doctors in their everyday labour [31]. As another example, further research in this field has led to the application of AR in devices such as the Google Glass, that could be used in surgical settings to indicate the doctor how to perform certain procedures [11]. Lately, literature is also exploring how AR can be used as a marketing tool [8, 55] that enhances customers' experiences and therefore influences their decisions concerning the purchase of a product. One of the most popular examples of this use-case is IKEA's new app called Place [18]. Under the slogan "Try before you buy", this mobile application lets you place true-to-scale photo-realistic furniture in any space, interact and walk around it.



Figure 2.2: a) IKEA's new app Place [18]. b) Augmented Reality information overlaid in healthcare applications [31].

### 2.1.1 Mobile Augmented Reality

Mobile Augmented Reality (MAR) and Handheld Mobile Augmented Reality (HMAR) are subsets of Augmented Reality, and refer specifically to any type of technology a user can carry to visually augment the world [2]. Throughout this dissertation, these concepts will be used interchangeably to talk about smartphone Augmented Reality, which is the main focus of this work. This section explores some of the background behind Mobile Augmented Reality and the technologies that make possible to experience it in our smartphones. The first implementations of smartphone AR applications date

from early 2004, with some demonstrations of real-time tracking of 3D markers [53]. Since then, and thanks to the technical advancement of mobile computing, AR can now be distributed to a large number of users and the potential use-cases are unlimited. Three main techniques are usually described in the literature [9] to create Mobile Augmented Reality experiences:

- SLAM (Simultaneous Localisation and Mapping): The position of the device and its motion is tracked through computer vision and machine learning algorithms.
- Marker Recognition: It tracks visual markers or objects to know the camera's perspective. Virtual objects are placed on top of the recognised markers. This method is limited to knowing the marker location at all times, and as soon as it goes away from the view frustum the tracking stops.
- Location Recognition: Uses GPS technology and/or a digital compass to track the device on a specific location. This type of Augmented Reality is usually helpful for overlaying information that refers to directions or nearby services.

Thanks to the increase of computing power in mobile phones, SLAM, and therefore markerless tracking, can now be run on a hand-held format. At the moment, HMAR it is mainly being developed by Apple and Google. In the late 2017, they both released Software Development Kits (SDKs) for their respective Operating Systems (OS) to enable developers build applications that leverage the use of Augmented Reality. However, Google's Advanced Technology and Projects (ATAP) team was already developing its AR technology since 2014 when they announced their project Tango, a device with multiple cameras and depth-sensing capabilities. The Tango platform was presented as a real-time location and mapping AR platform, solving most of the problems needed for a basic HMAR. Google Tango, with the help of Computer Vision, is able to have an adequate knowledge of the device location relative to the real world around it [35]. The Tango platform is fundamentally capable of three things: tracking motion, understanding the scene, and perceiving distances.

In March 2018, Google decided to shut down Project Tango [32], since the requirements and price of the hardware were not being well-accepted by consumers. However, all the principles behind Tango are now used in Google's new ARCore, the Android

Augmented Reality platform that can run on most commercial smartphones. With the help of just one RGB camera, and the on-board sensors, i.e. accelerometer and gyroscope, ARCore is able to track the device pose on real-time, have a basic understanding of the scene and estimate the lighting conditions. Under the same principles, Apple released the ARKit platform for their iPhone Operation System (iOS). Both platforms by the two U.S. technology giants are the reason why Augmented Reality has become so popular over the last couple years.

The following sub-sections will review some of the current features ARKit and ARCore come with, in order to understand the available functionalities, but also their limitations.

### **Motion Tracking**

Motion tracking can not be explained without first analysing the concept of Simultaneous Localization and Mapping (SLAM). SLAM is the process of understanding the device position and rotation by mapping the surroundings. Before landing on smartphones, it was a popular topic among the mobile robotics community. There are different algorithms which make SLAM possible, but essentially they all incrementally build a consistent map of the environment while simultaneously determining the device location within this map [20]. Thanks to the growth of computing power in mobile devices, SLAM algorithms can be calculated with significant consistency on smartphones. The first implementation of smartphone-based SLAM was presented by Klein and Murray in 2009 [33]. They proposed a keyframe-based SLAM approach that could be run on regular smartphones using the camera data as input. Nonetheless, using only images for 3D tracking can be limited and imprecise, and therefore SLAM algorithms can be enhanced by using sensor fusion technology [37]. This concept relies on combining the data of different sensors in order to have more accurate and detailed information about the device and hence better motion tracking. Lynen et al. [37] extend the basic SLAM algorithms with sensor fusion increasing robustness and accuracy of mobile robots navigation.

Understanding the device position and rotation on real-time starts with the data gath-

ered from the Inertial Measuring Unit (IMU), which is formed by a gyroscope and an accelerometer. From these two sensors we are able to obtain the rotational angle, acceleration, velocity and position of the smartphone. Due to the fact that IMUs are not accurate enough to track the device on their own, the image stream from the camera is also necessary in the process. By identifying characteristic visual features on every image and tracking them overtime using SLAM, the camera pose can be recovered with respect to these features. The process that computes IMU data and images for motion tracking is called Visual-Inertial Odometry.

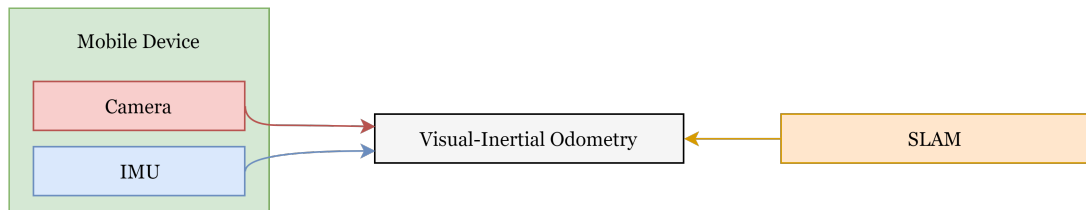


Figure 2.3: VIO systems are possible thanks to the combination of SLAM and sensors.

Nevertheless, Visual-Inertial Odometry poses an evident obstacle: it drifts overtime and objects are moved from the original position they were detected at. Thanks to SLAM optimisation processes (Figure 2.4), VIO is able to increase the accuracy of the scene map in the background and detect loop closures [58] that stabilise the previously detected poses. Due to the memory limitations of smartphones, once the scene map becomes too big, the system needs to remove some of the information which will no longer be used for the reconstruction.

### Light Estimation

Smartphone AR detects information about the lighting of its environment by processing a given frame. The light estimation is calculated by filtering the intensity of a subset of pixel data for every single frame. Once computed, it returns a value of the average brightness of the scene so the rendering style of the objects based on it can be changed. The quality of the immersion is improved by adapting the virtual objects to the surroundings and merging their aspect into the real world.

## Scene Understanding

In order to place virtual content on the scene, a 3D representation of the physical world is needed. Once the camera pose has been tracked efficiently, the HMAR frameworks (i.e. ARCore and ARKit) improve their knowledge of the scene by detecting planes, vertical and horizontal. Due to the lack of depth-sensing capabilities in commercial smartphones, the scene understanding is limited to a few planes and a cloud of points. Every plane is defined by a polygon and helps creating the illusion of virtual objects resting on top of real-world surfaces.

Robust plane detection is achieved through detection of subsets of coplanar feature points from the monocular camera image [3, 16]. On the other hand, the data of the detected point cloud is improved by estimating surface normals [14], an expensive method that is only helpful when the visual points are abundant.

### Mobile AR Processing Pipeline

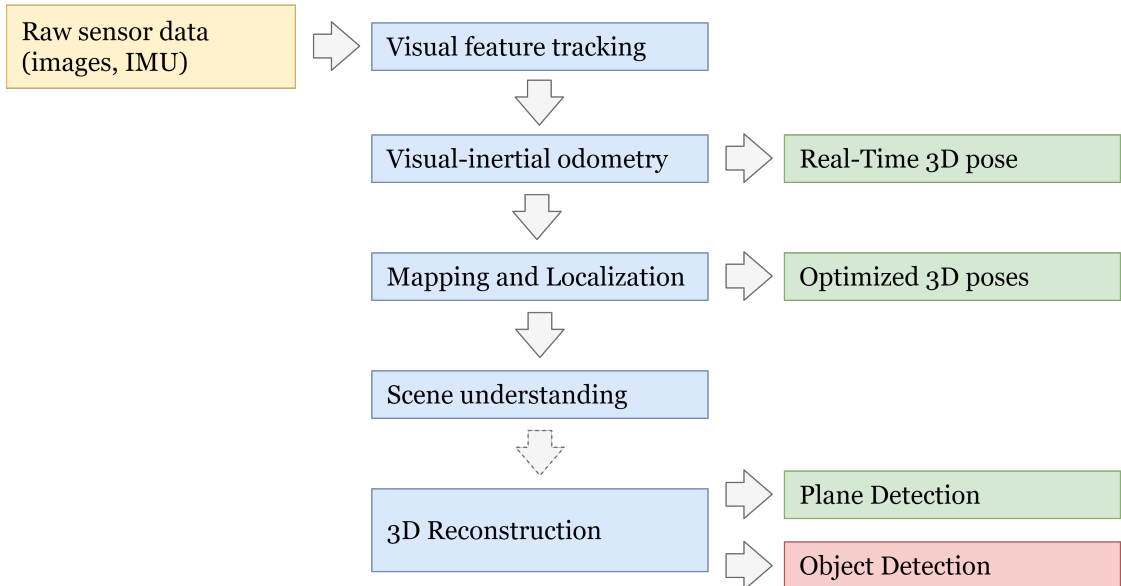


Figure 2.4: Google’s ARCore processing pipeline. Our dissertation topic deals with “Object Detection”, highlighted in red.



### 2.1.2 Immersion

The concept of immersion has been addressed in numerous contexts; however, it has recently been used to a greater extent in the area of Augmented and Virtual Reality. Immersion is defined as the user sensation of involvement in the virtual world [56] and it is linked to the mental sensation of engagement. Apart from the research that has been conducted on psychological and emotional effects of Virtual Reality [54, 55], digital immersion can be influenced by mainly three technical factors [56]:

- **Hardware:** it plays a big role on how the user is able to engage with the virtual reality experience. Headsets and glasses provide nowadays the most immersive experience, as they increase the field-of-view (FOV) and therefore minimise the visual boundaries between physical and digital world. This is in contrast to smartphone AR, which only covers a small amount of real world and subsequently leads to the need of other ways to enhance immersion.
- **Physical interactions:** the virtual objects need to interact with the world in a lifelike manner. Otherwise, the immersion effect may be lost if they do not follow the laws of physics the user is accustomed to.
- **Visual perception:** how the real world and the virtual objects are rendered can completely change the effect that the experience has on the user [57]. Augmented information can be rendered as realistic as possible to be seamlessly integrated with the environment, or on the other hand, the real-world can be stylised to match the virtual objects [12].

## 2.2 3D Scene Reconstruction

While scene understanding has already been mentioned previously in this dissertation, the current capabilities of consumer HMAR three-dimensional reconstruction are very limited. So far, ARCore and ARKit only include plane detection and in order to extend this feature we need to review some of the work that has been done in the 3D reconstruction research field.

Point cloud data is commonly used in image analysis for computer-aided design (CAD), physical simulations and medical research [1], among others. One simple method for 3D reconstruction is saving multiple point clouds directly as a representation of the physical space. These so-called point-based methods use points [4, 48] as their primitive shape being an alternative to polygonal meshes, in particular when the models are highly complex or dynamic. However, when the input data is very sparse and noisy, these methods fail to accurately reconstruct the real world: areas without available points are omitted thus generating gaps on the three-dimensional representation, and negative (non-surface) information is never captured [34]. A different approach for 3D reconstruction was introduced in Elfes [21] alongside the term “Occupancy Grid Mapping”. This method divides the world into a voxel grid that contains *occupancy possibilities* generating maps from sparse and uncertain input data. Even if the original paper was focused on robot perception and navigation, this algorithm has been implemented in the Tango [40] platform, achieving impressive results.

Curless and Levoy [17] first introduce the concept of signed distance functions to volumetrically reconstruct 3D objects from range images to overcome the limitations of occupancy grids. The scanned information is converted to a weighted signed distance function (SDF) of each point to the nearest range surface along the line of sight of the scan device. Therefore, the real world data is discretised into a grid of values: data inside a surface will be negative and data outside of it will be positive. Later on, thanks to the use of a truncated signed distance function (TSDF) the surface is reconstructed at subvoxel accuracy, removing the outliers present in the voxel grid. This method has been extensively used since then in most of the depth-sensing cameras in the market, and other image-based capture devices.

*Kinect Fusion* by Microsoft Research [46] tracks the 3D pose of the Kinect sensor and geometrically reconstructs precise 3D models of the physical world. However, *Fusion* uses a fixed voxel grid, depth sensing and a heavy GPU-based pipeline for its reconstruction process, not being suitable for mobile devices with small memory and only RGB cameras. Later research such as *Kintinuous* tries to solve this limitation by efficiently removing data not visible from the GPU. Hence, reconstruction data has to be gathered and computed again if we point the sensor to an already deleted space.

Some of the principles of *Fusion* were lately applied by *MonoFusion* [52]. Pradeep et al. present how to build dense 3D reconstructions of the environment in real-time thanks to the use of a standalone camera without depth information. Compared to existing approaches, the live image stream is feed into a feature tracking algorithm that estimates the camera pose in 6 degrees of freedom (6DoF). Their method is GPU computational-intensive, and requires a desktop or high end laptop to perform efficiently.



Figure 2.5: a) Example of a 3D-reconstructed scene with Kinect Fusion [46]. b) Truncated signed-distance function using a Kinect as scanning device [17].

It is also worth mentioning some under-development mobile applications that use locally processed photogrammetry for 3D scanning objects and scenes. Photogrammetry is defined as the science of taking measurements from photographs and is the principle behind some monocular RGB scan devices. Tim Field from Abound Labs [22] published, on July 31<sup>st</sup>, 2018, an early-access beta app for 3D scanning in Augmented Reality using real-time photogrammetry with impressive results. After a scan phase using mobile GPU computing for speed up post-processing, the reconstructed mesh can be used for collision detection and occlusion in ARKit.

### 2.2.1 Spatial Hashing

Spatial hashing is an optimisation technique in which objects in a 2D or 3D domain space are projected into a 1D hash table allowing for very fast queries on objects in the domain space [28]. The look-up complexity of  $O(1)$  make spatial hashing faster in large-scale environments, but slower than other data structures like quadtrees when it comes to small sets. Spatial hashing is widely used in several areas, such as collision

detection and 3D reconstruction, to spatially distribute data into a grid-divided space.

In Teschner et al. [60], spatial hashing is used for efficient collision detection of deformable objects. By distributing the data into a hash table, we can easily narrow down which collection of objects are intersecting in what is called the *broad phase*. Afterwards, the collision of that set of objects is calculated with more precision during the *narrow phase*. In [60], during the first pass of the spatial hashing algorithm, the hash values are computed to discretise the space into a uniform-sized grid by using the following hash function:

$$\text{hash}(x, y, z) = (x \cdot p1 \oplus y \cdot p2 \oplus z \cdot p3) \bmod n \quad (2.1)$$

Where  $p1$ ,  $p2$  and  $p3$  are large prime numbers and  $n$  is the hash table size. Experiments indicate that the size of the hash table directly affects the performance of data manipulation; the higher the value, the more efficiency achieved. Once we have our unique hash value, data is saved into a hash table where it can be retrieved at any time by performing a simple query. This equation is the starting point for multiple papers [47, 30, 29] that research about three-dimensional representations of the physical world. Later research [36] explores solutions for making spatial hashing compatible with GPU parallelism, removing unused entries from the hash table by pre-computing the hash value. However, this method is tightened to a fixed-size grid with uniform resolution.

### 2.2.2 Voxel Hashing

In Nieet al. [47], the plain spatial hashing data structure is extended for volumetric reconstruction in what they call “Voxel Hashing” (Figure [47]). They show interactive real-time reconstruction of different scenes using this data structure to efficiently organise the input obtained from consumer depth cameras. In contrast to other methods, voxel hashing does not require neither a hierarchical data structure nor a memory-constrained voxel grid. Their pipeline uses a hash function to map the integer world coordinates to hash buckets, in which pointers to voxel blocks are stored, accessed and updated dynamically. The data structure they presented can be seen in Listing 2.1.

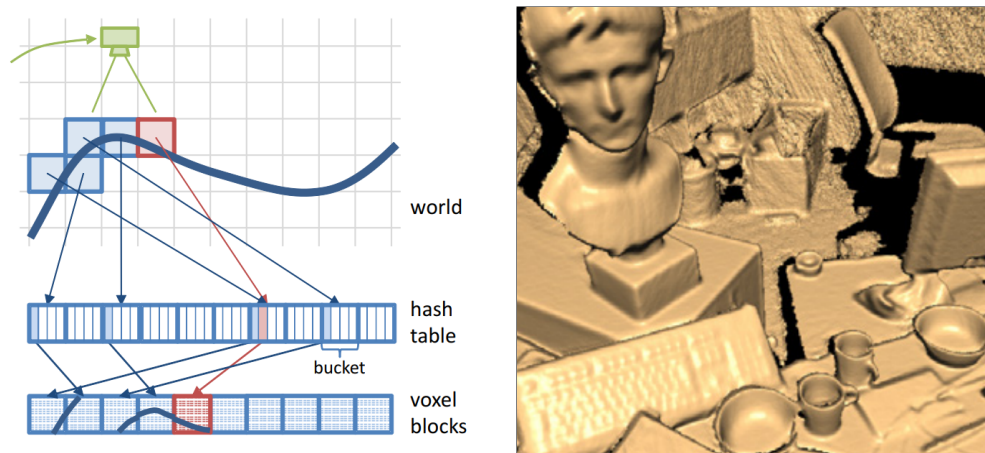


Figure 2.6: a) The Voxel-Hashing data structure proposed in [47]. b) Results of 3D reconstruction with Voxel-Hashing [47].

```

struct HashEntry {
    short position [3];
    short size;
    int pointerToVoxel;
};

```

Listing 2.1: Hash entry for the voxel hashing data structure [47].

In Klingensmith et al. [34], voxel hashing is applied to mobile augmented reality in the Tango platform avoiding needless computation and wasted memory. Their implementation achieves a real-time 3D reconstruction with a resolution of 2-3 cm without having to use GPU computing. Their pipeline pre-optimises the input data, only storing the relevant one in the GPU and throwing away the rest. Other works such as Hui et al. [29] and Muratov et al. [45] propose minor extensions of the general voxel hashing data structure by merging voxels to build a multi-scale 3D reconstruction, and by post-processing the data with OpenCL after a 360° scan respectively.

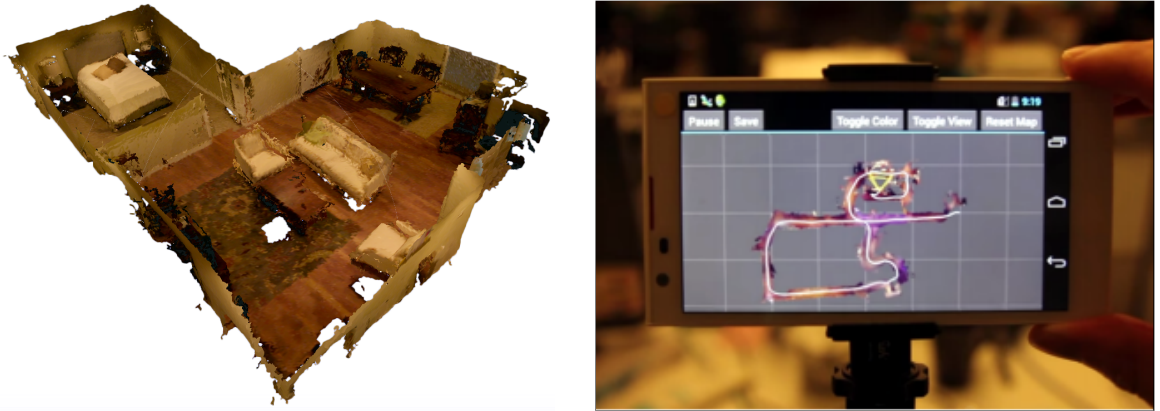


Figure 2.7: a) 3D reconstruction of a scene using Chisel [34]. b) Chisel application based on *Project Tango* [40].

### 2.2.3 Hierarchical Data Structures

For some specific implementations, TSDF information has been saved into custom hierarchical data structures such as octrees or KD-trees due to the vast amounts of data that can be obtained from sensors nowadays. A balance between resolution and efficiency has to be found to store data. One of the most common examples of hierarchical data structures are octrees. An octree subdivides the space recursively into eight equally-sized volumes until the final partial volumes represent a unique voxel [64], increasing the memory footprint but allowing a more detailed reconstruction resolution. In an octree, each new level results on an increase of resolution by a factor of two impacting performance due to the need of a deep tree structure. In Zeng et al. [64], Octrees are used to organize the sensor data from *Kinect* extending the original *Kinect Fusion* [46] research. The results show a 2 times faster reconstruction with an increase of only 10% of memory usage.

Kähler et al. [30] presents a novel approach for hierarchical voxel-hashing. Avoiding unnecessarily deep trees, they store a limited number of levels depending on the resolution of the scanned area. Every entry of the voxel hash table contains points to the data blocks of  $8^3$  voxels each. Depending on the depth and resolution, more data is represented in lower levels, and a special marker indicates if additional information can be found at a deeper level. In order to efficiently obtain the hash values or indexes of

the hash table, they use the aforementioned hash function (2.1)[60].

## 2.3 Collision Detection

Collision Detection is the process of determining the geometric contact of virtual objects with respect to others or themselves. Despite the large amount of existing work in this research field, further study is needed in the new context of Smartphone Augmented Reality. A wide range of solutions are already available for computational geometry, physics simulations or robotics among others, but there is always room for improvement in this area due to new hardware and research fields such as AR. In collision detection, there is no perfect solution for every system, fluids, rigid bodies, particles, or waves require completely different approaches to this problem. When testing a collision detection algorithm, performance can be determined by measuring its complexity, speed, accuracy and difficulty of implementation [38].

In order to overcome the computational cost of collision calculations there are two general approaches: to simplify the objects that are colliding or to simplify the space in which they collide. Usually, collision detection and response are studied in basic primitive shapes such as spheres, boxes and cylinders. Computing the contact points of these shapes is cheap, and more complex polyhedra can be modeled by combining these small pieces. Some algorithms [24, 42] make use of multi-resolution techniques that calculate the collisions with lower resolution versions of the original model. However, these systems can run into inefficient memory footprint because of the need to store different representations of the same shapes, what is usually known as Level of Details (LODs).

The practical approach to collision detection most widely used divides the computation into two different stages:

**Broad-Phase:** determines which objects are close enough to collide and need to be checked. The rest of objects are removed from the collision detection loop hence the computation cost is highly reduced.

**Narrow-Phase:** once the subset of colliding objects has been obtained from

the Broad-Phase, a more expensive calculation is done for those specific objects. This phase usually processes exact geometry collision.

Collision detection algorithms are boosted with two main Broad-Phase methods: Spatial Partitioning and Bounding Volume Hierarchy (BVH). This latter method achieves impressive results when used on rigid bodies and deformable objects by pre-processing their simpler representation. BVH was first introduced in 1976 by Clark [13], who explored a hierarchical algorithm for accelerating object render processes. By testing collision with a simpler shape instead of the object itself, Bounding Volumes reduce the cost of computation. There are multiple ways of structuring the data spatially, for example, AABB (Axis Aligned Bounding Box) compares efficiently the overlapping coordinate values [19]. Other popular method is Oriented Bounding Box (OBB) [27], allowing a tighter collision than AABB because of the orientation information, despite being less efficient. Some of the drawbacks of OBB and AABB are overcome by using sphere bounding volumes, due to the fact that they are rotationally invariant, transformation is defined by their center points and overlaps are faster to calculate [49].

Another way for efficiently calculating collisions is by culling away whole objects that are too distant from each other, and therefore they will be removed from the computing loop. Instead of dividing the object into different regions, the spatial partitioning approach splits the virtual space into several sections. Three different types of spatial structures are mentioned in literature: grids, trees and hashed storage [38]. Grids divide the space into uniform-sized sections, lacking of higher resolution representation when it is needed. To overcome this issue, trees recursively go to lower levels, usually allowing a finer resolution but increasing the memory overhead. In order to reduce the memory usage, hashed storage (hash tables) help organise the data with a potentially infinite limit.

### 2.3.1 AR Collisions

Apart from the abovementioned general collision background knowledge, it is important to go deeper into the problem and see what solutions are proposed in the literature for collision detection in Augmented Reality environments.



In Breen et al. [7], several techniques for interactively performing collision detection between real objects and dynamic virtual objects in Augmented Reality are addressed. The authors bring attention to the idea of seamlessly merging real and virtual objects in order to increase the believability of the AR experience. In their implementation, they propose two different techniques for acquiring models of the real world based on computer vision algorithms. The first one consists in model-based methods and uses pre-designed 3D models to align them to the real world geometry. The pose of the 3D objects is usually set by using visual features from the computer vision algorithms, or manually set by the user. This method registers the object's landmarks and calculate their transformation based on the camera's coordinate system, reducing the input of the user and automating the procedure. The second type of techniques for scene understanding falls into the category of depth-based methods, in which the depth of the real environment is acquired through a depth-sensing camera. The depth map generated with this method does not include full information of the object and is tied to the viewpoint of the camera. However, depth-based methods allow to have a more general knowledge about objects that are not available in the collection of models.

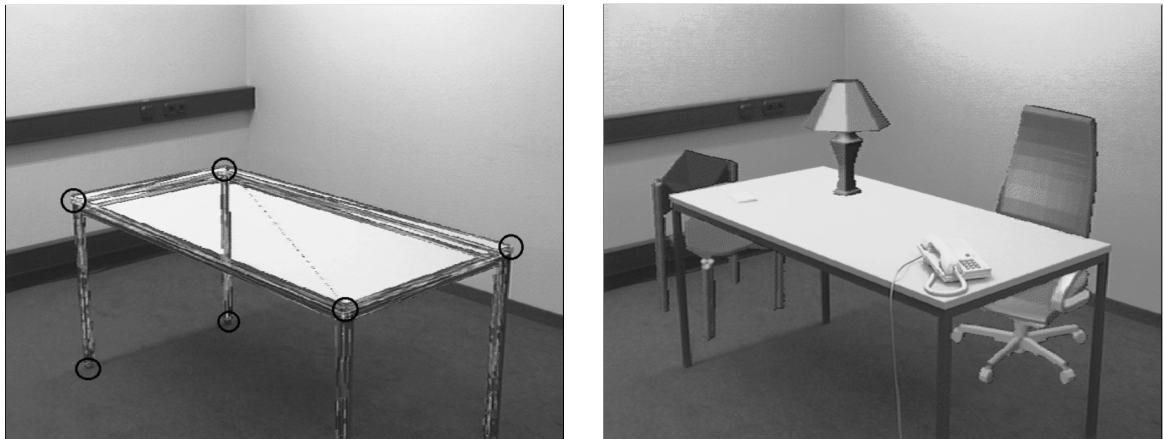


Figure 2.8: a) A wireframe model is overlaid on a real object after registration [7]. b) The model is used for occlusion and collision of digital objects (two chairs and one lamp) [7].

Later in their research, Breen et al. investigate collision and occlusion for both of these techniques. Virtual objects are hidden when they are behind the real object representations, and collision between them is achieved by comparing 3D positions of their

bounding boxes [19]. The authors conclude by stating that the model-based method is highly limited in those cases when the scene is large, or the objects are not present in the repository. On the other hand, depth-based techniques do not need prior knowledge of the scene but have a higher computational cost for real-time applications. For the time being, their approach showcased admirable results that notably improve the physical interactions of the user in AR.

Since then, the various solutions that appear in the literature [46, 52, 29] tend to follow similar *modi operandi*: improving the scene understanding (mesh reconstruction [46]) and applying a general knowledge of physics calculations and collisions (as described in Section 2.3) in order to enable interactions between virtual and real objects in AR.

## 2.4 Level of Detail (LOD)

Level of Detail (LOD) is the concept of decreasing the complexity of a 3D model as it moves away from the viewer or according to other variables, such as object relevance, speed or resolution. In Computer Graphics, LOD techniques decrease the computational cost of graphic pipeline stages by reducing the number of polygons that need to be rendered at a given time. When it comes to textures, multiple levels of detail are usually called mipmap [62]. Thanks to cross-platform rendering engines [23], dynamic geometry level-of-detail (LOD) algorithms have become more popular and help run the same source code in different GPU architectures with adaptive render qualities. In addition, LOD techniques are not only helpful to relieve heavy rendering, but also to optimise the computation of expensive tasks that can be omitted. In O’Sullivan et al. [50] crowd simulation, it proves to be cost-effective due to level-of-detail artificial intelligence (LODAI). For example, if an agent is not in the current view-frame, its complex behaviour is simplified to only make it appear believable. However, they not only use LOD techniques for crowd behaviour, but also for geometry rendering, collision and motion. One of the problems they mention in crowd simulation is the lack of realism due to the use of coarse collision approximations. A hierarchy of rigid bounding volumes is used in their system, and the different levels of resolution are switched as required for collision detection. Further information on LOD used in different areas such as collision detection or animation, can be consulted in [25, 41].

# Chapter 3

## Adaptive Volumetric Representation

The overall goal of this research project is to design an efficient pipeline that extends the current scene understanding of smartphone AR. The approach in order to achieve this objective is a novel data structure that stores key-frame independent point clouds into a hash table of voxels that can be used for physics interactions. Through the use of an extended version of Voxel-Hashing [47], different levels of detail for each voxel are created to freely adapt the resolution of the reconstruction based on any given factor. This chapter explains in detail all the design choices that have been taken to build an adaptive level of detail data structure for scene understanding in smartphone AR.

### 3.1 Scene Understanding in AR Frameworks

ARCore and ARKit frameworks for smartphone Augmented Reality are only able to recognise 3D planes in the scene. As explained in Section 2.1.1, subsets of coplanar feature points are extracted from the monocular RGB camera data and combined to build a polygon that represents each plane. Moreover, AR frameworks have limited access to their internal processes, and only allow developers to access the detected planes and a collection of feature points called “point cloud”. Point clouds are local visual features on the current frame with a significant change of an image property (e.g. intensity, colour, texture). When the device is moved around the scene, correspon-

dences of the visual features can be found and mapped into the 3D space. However, the point cloud extracted by AR frameworks is frame independent, hence there is no built-in way of knowing how visual features change from frame to frame. Taking this into consideration, the need arises to design a custom approach in order to efficiently store those feature points. Every point extracted from the point cloud is stored as an array of four floats, where the first three represent the  $x$ ,  $y$  and  $z$  3D coordinates of the point measured in meters. The last float ( $w$ ) indicates a confidence value given by the framework in the range of  $0 - 1$ , where 1 corresponds to full confidence in the measured position values. Point locations are based in world coordinates, consistent with the camera pose for the specific frame that provided the point cloud. Apart from the collection of points, each point cloud has a timestamp which indicates in nanoseconds the moment it was observed.

These factors contribute to several alternatives in order to represent the scene. One possible way to do this with the help of the point cloud data would be to use point-based scene reconstruction methods (Section 2.2). However, from a monocular scan device, computer vision algorithms are not able to detect high-confidence visual features in textureless or reflective surfaces, resulting on a very sparse point cloud data. While point-based reconstruction shows impressive results when the models are complex or highly dynamic, it works poorly with scattered data [48]. In addition, the lack of computational power and memory on smartphone devices proves to be inefficient for the storage and 3D tracking of thousands of points.

A second technique is mesh reconstruction, despite leading to problems similar to the ones found in point-based methods. Gaps in the reconstruction appear when there is not enough information available from the point cloud data and the quality of reconstruction will not allow accurate physics interactions (i.e. virtual objects going through real ones). Moreover, mesh reconstruction relies on normal data from the surfaces, which is not available in the current versions of smartphone AR frameworks.

## 3.2 Basic Voxel-Hashing

In order to efficiently store on the device the three-dimensional volumetric representation, the Voxel-Hashing [47] data structure has been extended in our proposed design. In the classic Voxel-Hashing data structure, every hash entry is used to store TSDF values into a hash table. Each occupied entry in the hash table refers to an allocated voxel block, and each block stores a TSDF value along with its weight and colour. By using the hash function described in Equation 3.1, voxel-blocks can be efficiently found in the hash table by simply having the world coordinates. Nonetheless, this data structure was designed for a fixed resolution 3D representation, where the resolution can not be changed over time or by any given factor such as the density of the point cloud or the distance to the scanning device.

Conceptually, the world space is divided into a infinite uniform grid. Every visual feature detected by the framework is located inside the grid and mapped to a block. Likewise, each block or cell that forms the grid can contain multiple points based on the scale of the subdivisions, which will cause redundant voxels to be reconstructed. In order to retrieve and avoid duplicate entries, every block in the space is identified with a unique hash value calculated using the following function:

$$\text{hash}(x, y, z) = (x \cdot p1 \oplus y \cdot p2 \oplus z \cdot p3) \bmod n \quad (3.1)$$

where  $p1$ ,  $p2$ , and  $p3$  are large prime numbers (in this specific case, 73856093, 19349669, and 83492791 respectively based on [Teschner et al. 2003] and [Nieet al. 2013]) and the  $\oplus$  symbol represents the exclusive OR operator. The size of the hash table ( $n$ ) is set to the prime number 9973, since experiments demonstrate that large numbers have a positive effect on the hash table operations [60]. This function is designed to work with integer values, but for this specific purpose the point cloud data is measured in meters. If data is used without scaling, each voxel and hash entry will have a fixed size of  $1\text{m}^3$ . To overcome this obstacle and allow more flexibility, the world coordinates of each point in the cloud are converted to a integer value by scaling them based on the cell size of our grid. Flooring the point positions and removing their decimal part after scaling converts them to a unique hash value when computing the hash function.

### 3.3 LOD Data Structure

By repeating the aforementioned procedure for all the data in the point cloud, a collection of hash entries with a unique hash value is obtained. All the entries are saved into the hash table where points inside the same grid cell will be discarded and therefore not duplicated. This approach presents a main problem, since the size of each item of the grid is fixed and there are cases where more detailed information of the scene is needed. For instance, in cases when the user is closer to an object, a more detailed reconstruction should be used. On the contrary, if the user is far away, the whole object will be represented as a unique voxel reducing the cost of physical calculation. With that aim in mind, Listing 3.1 and Figure 3.1 show a hierarchical data structure for efficiently storing point clouds into a hash table of voxels. Points in the world space are hashed and then saved to a adaptive representation that can handle different level-of-details for different use-cases. Multiple resolutions of the same reconstruction can be retrieved and efficiently used when required. The basic Voxel-Hashing procedure previously described is employed to achieve the first level of detail of the reconstructed scene (LOD-0) by having the block/cell size in  $\text{cm}^3$  as unique parameter.

```
struct HashEntry {  
    float position [3];  
    float size;  
    float confidence;  
    float distanceToCamera;  
    Voxel voxel;  
    HashTable LOD[n];  
};
```

Listing 3.1: Hash entry data structure representing LOD-0.

Once the high-level voxel representation of the point has been obtained (LOD-0), the rest of resolutions for that same voxel can be built. For that purpose, the algorithm starts transforming the point in world space coordinates gathered from the point cloud into voxel space coordinates ( $P_{voxel}$ ). This process occurs by using the inverse transformation of a point (Equation 3.2), that takes as parameters the voxel transformation

matrix  $M_{voxel}$  and the point transformation matrix in world coordinates  $P_{world}$ . The measure units of the point inside of the voxel space are normalised from meters to percentage, facilitating how the data is handled. The parent voxel is then divided into a grid of equally-sized subvoxels. The number of subvoxels (e.g.  $2^3$ ,  $3^3$ ,  $4^3$ , ...,  $n^3$ ) for each resolution is pre-defined by the developer of the application and indicates how much detail will each voxel have.

$$P_{voxel} = P_{world}^{-1} \cdot M_{voxel} \quad (3.2)$$

In order to ensure that each point is converted to a unique subvoxel for the levels of detail, spatial hashing is to be computed once again. Since the point position is now normalised and measured in the voxel space, the coordinates are scaled and floored (Equation 3.3) based on the subvoxel grid size and then fed to the hash function (Equation 3.1). This process is repeated for every LOD of the reconstruction before storing the subvoxel data in the hash table. Each voxel entry has an array of hash tables referring to the various resolutions that have been reconstructed for that space in the world. Hash tables are very well suited for representing each level of detail and can be efficiently accessed in modern programming languages [60]. The same hash function as in LOD-0 is employed to calculate the hash value, mapping the point to a subvoxel inside of the parent voxel.

$$P_{scaled} = \left\lfloor \frac{P}{GridSize} \right\rfloor \cdot GridSize \quad (3.3)$$

This approach allows a 3D representation of the scene that overcomes the problems which point cloud sparsity entails. The areas with small point density will still be represented, whereas the areas with higher density will be reconstructed in further detail. TSDF information is not stored as in other related work [47, 30]; the point from the point cloud is instead directly converted to a set of voxels and subvoxels. For example, an object will be reconstructed with  $10 \text{ cm}^3$  voxels for the LOD-0 representation. Two more levels of detail are set in the pipeline, LOD-1 and LOD-2, containing  $2^3$  and  $4^3$  subvoxels of sizes  $5 \text{ cm}^3$  and  $2.5 \text{ cm}^3$  respectively. LOD-1 and LOD-2 are hashtables saved in the LOD array (Listing 3.1) and their information can be retrieved through

a unique hash value. Fundamentally, copies of the same voxel with different details are saved into the memory thus enabling flexibility of use. When it is required, the resolution of the scene reconstruction is increased for more accurate collision. In our particular case, a float variable —called `distanceToCamera`— is updated every time the camera view frustum changes and used to update the level of detail represented.

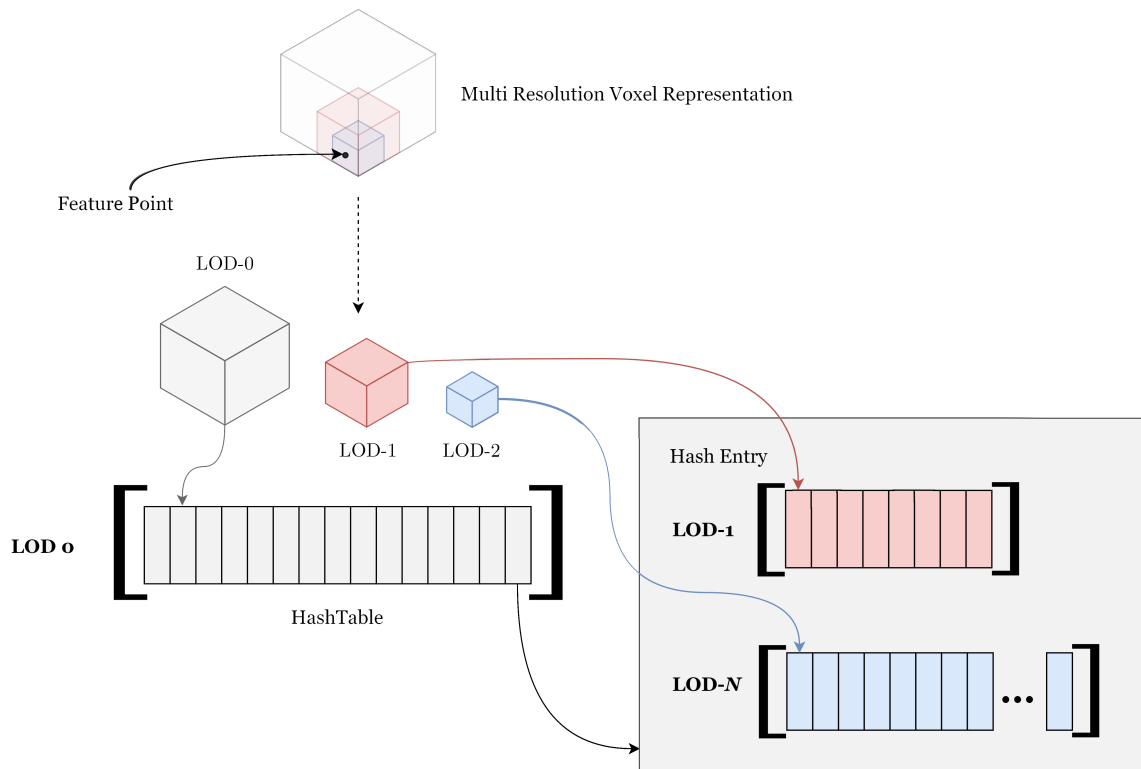


Figure 3.1: Our Adaptive LOD data structure for storing different resolutions of the scene. Feature points are the starting data needed for building a hierarchical hash table of voxels.

This custom technique for storing data does not follow the principles of conventional tree data structures with  $n$ -depths; in contrast, the  $n$ -resolutions reconstructed are children of the first LOD. Depending on the visual-features that the VIO system is able to extract for some specific area, the details (subvoxels) available will be different. Some points will be contained inside the same LOD-0 voxel, but after running the hash function for the different level of details, these points could be missing. This means that, for a specific resolution, the point has not been stored yet, therefore it is created



and the detail of that specific area is increased (Figure 3.2). The LOD-0 voxels have a parent-child relation with various levels of details, and therefore any transformation on the parent will directly affect all the contained subvoxels.

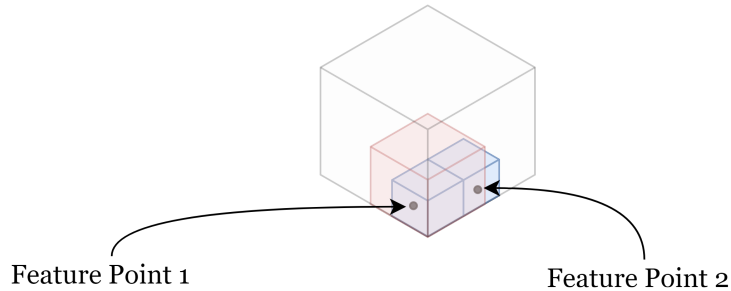


Figure 3.2: Features points contained in the same LOD-0 (grey voxel) can be in different subvoxels for the different resolutions.

### 3.3.1 Confidence Value

The data structure hereby proposed also contains a confidence value that is increased based on how many times a point has been hashed to the same hash value. Since the point cloud information from the AR framework is key-frame independent, this variable is useful to discard low-confidence voxels from the 3D reconstruction. Another way of measuring the relevance of a particular voxel is by counting the hash table entries it has in its respective levels of detail (LOD). The garbage collector checks these two parameters (confidence value and density) to identify outliers and noise and remove the amount of unwanted voxels.

## 3.4 Hash Operations

**Insertion** – To insert new hash entries, the hash value for a specific point is computed. The first step is to verify if the hash value is contained in the first-level (LOD-0) hash table, and then a second check is recursively done for each level of the detail hash tables. New voxels are only inserted if the hash value is not contained, otherwise the confidence value is increased.

**Look-up and Update** – To read and update hash entries, a linear search based on the level of detail needs to be performed. If the level of detail is 0, the search will be made on the main hash table. If the level of detail is other than zero, the specific hash entries based on the hash value are retrieved, and a linear search is made in their respective LOD hash tables. The small number of subvoxels that can be contained in every voxel increase the efficiency of this procedure.

**Deletion** – To delete a hash entry, retrieval and deletion of every single element in their LOD hash tables is needed first in order to avoid unwanted data in the device. The final stage is the removal of the high-level voxel from the memory. For cases in which the usage of a specific voxel is unwanted for reconstruction or physical interaction, the information of the voxel will be left empty but will still be present in the hash table to avoid unexpected insertions (e.g., if the voxel is too close to an already detected plane).

### 3.5 Reducing Tracked Objects

AR frameworks build their own spatial understanding of the surroundings in real-time using SLAM for mapping and localisation of the visual features. One common problem that can be expected from AR experiences is virtual objects shifting from the real position overtime. This issue is more common in the early phases of reconstruction, when the app has been recently opened, although it also occurs when the device runs out of memory and the garbage collector is activated to destroy some of the map information. In order to accurately track every object in the scene, anchors need to be placed. Anchors are tracked points in the space that ensure that objects stay in the same position and orientation in space over time, helping to maintain the illusion that they are placed in the real world [26]. When an anchor is created in a specific point, the estimated position and orientation relative to the world space is stored and added to the tracking loop. Placement of anchors leads to a trade-off: tracking them is expensive for the AR internal process and the device can easily run out of RAM memory. Therefore, anchors need to be placed effectively and efficiently, and only employing the important anchors reduces the computational cost. In the current version of ARKit and ARCore, anchors can be sent to the cloud to allow online multi-device experiences

and spatial mapping with multiple cameras at the same time. This fact justifies the creation of only the necessary anchors to avoid sending useless data to the server.

Our suggested data structure helps organise the point cloud into voxels, and also allows having several levels of detail while avoiding the placement of unnecessary anchors in the world space. Only the LOD-0 voxels are attached to an anchor, whereas all the sub-voxels contained in the different reconstructed resolutions are children to them. Every subvoxel transformation matrix is calculated in respect to its parent. Subsequently, if the anchor changes its pose (position and orientation) every element on the hierarchy tree will be affected.

## Chapter 4

# Object Reconstruction Pipeline

The fourth chapter examines the three-dimensional reconstruction pipeline, which takes the point cloud as input and processes it to obtain a hierarchical structure of voxels that can be used for physical interactions. In order to achieve real-time performance, voxels are processed in a multithreaded pipeline.

### 4.1 Point Cloud Input

The bare minimum for the algorithm we are developing in this dissertation is a set of 3D points that can be detected by any Virtual-Inertial Odometry (VIO) system. Working with the points alone fails to obtain an adequate representation of how the world looks, hence the need to process this input into a collection of voxels or primitive shapes. Our points are described by their 3D coordinates ( $x$ ,  $y$  and  $z$ ), and a confidence value ( $w$ ) that the system has about each point. Most of the 3D reconstruction algorithms reviewed in Section 2 rely on surface normals from the depth scanning devices. However, VIO systems in AR do not provide this information and therefore, we need a simplified reconstruction that meets the requirements of basic physical interactions. The points are gathered in a frame basis and directly fed to the pipeline and converted into a set of voxels that represent the scene.

## 4.2 Processing the Point Cloud

For every visual feature detected at a specific time, the hash value using hash function (Equation 2.1) is computed. Ideally, multiple threads will be created to process all the points efficiently. However, we might encounter multiple points in a single frame that are contained in a unique voxel. By multithreading the process, collisions in the hash table will occur when different threads try to insert entries with the same hash value. This issue has been proved to create unexpected results, such as the inconsistency of the different levels of detail created for each voxel with the information retrieved by the VIO system. In order to avoid this problem while still benefiting by multithreading and multiple resolutions, every hash value calculated is stored into an array that helps organise the multithreaded processing pipeline. For every unique hash value, a new thread is created (Figure 4.1). If, for a given frame, multiple points are detected to have the same LOD-0 hash value, they are inserted into a queue and run in a unique thread. Each thread for colliding hash entries computes the information necessary to add new voxels into the hash table. In any given queue, the first element will create a LOD-0 voxel, and the rest of duplicate hash values will be checked so as to add detail to the parent voxel.

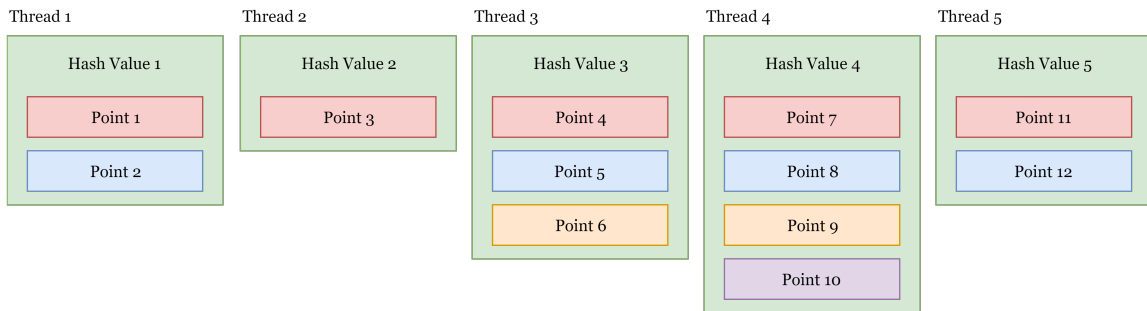


Figure 4.1: Our multithread processing pipeline organises threads based on the hash values of the points in order to avoid hash table collisions.

### 4.2.1 Building different Levels of Detail

Every thread that is created for the hash values runs the Algorithm 1 to insert the voxels into the hash table. The first step is to check if the hash value is already contained in the hash table. If it is not, a new voxel is created and saved as LOD-0 (first

level resolution) of that point. Later, the system iterates through all the pre-defined resolutions that need to be created and a subvoxel is generated and stored in the respective LOD hash table. All the subvoxels are contained inside the LOD-0 voxel and therefore all the transformations applied to the parent will be transmitted to the children. Therefore, every high-level voxel has a minimum of  $n$  subvoxels, where  $n$  is the number of LODs (different resolution copies) that are saved.

---

**Algorithm 1:** Thread processing pipeline

---

**Data:** Array of points with same pre-computed hash  $\rightarrow$  Points.

Hash table of voxels  $\rightarrow$  HT.

Resolutions to be reconstructed  $\rightarrow$  LODs.

**Result:** Modified hash table of voxels  $HT$ .

```
foreach Point  $P \in Points$  do
  Point  $P_{scaled} \leftarrow Scale(P)$ 
  Point  $P_{floored} \leftarrow Floor(P_{scaled})$ 
  int  $hash \leftarrow GetHash(P_{floored})$ 
  if  $hash \in HT$  then
     $increaseVoxelConfidence(hash)$ 
    foreach LOD  $L \in LODs$  do
      int  $subHash \leftarrow GetSubHash(L, P_{floored})$ 
      if  $subHash \notin HT[hash]$  then
        |  $addSubvoxelToLOD(L, subHash)$ 
      else
      end
    end
  else
    |  $addToHashTable(hash)$ 
    foreach LOD  $L \in LODs$  do
      |  $addSubvoxelToLOD(L, hash)$ 
    end
  end
end
```

---

If the hash value is contained on the LOD-0, the corresponding voxel is retrieved and the different levels of detail contained in the array of hash tables are checked for

collisions. The confidence value for every voxel found also increases if it has already been represented. In this project, the total confidence  $C$  for every voxel has been calculated with the Equation 4.1, where  $C_P$  is the confidence of the point  $P$  extracted from the point cloud information provided by the AR framework and  $n$  is the times that point was previously detected. A collision box is also designed for every voxel and subvoxel created, to enable collision detection from the physics system.

$$C = \frac{(n - 1) \cdot C_{n-1} + C_P}{n} \quad (4.1)$$

The scale of every subvoxel with respect its parent is calculated using Equation 4.2, where  $n_{divisions}$  is the number of divisions in which the subvoxel is split (e.g.  $2^3$ ,  $3^3$ ,  $4^3$ , ...,  $n^3$ ).

$$V_{size} = \frac{1}{\sqrt[3]{n_{divisions}}} \quad (4.2)$$

If the 3D reconstruction is enabled, new points are fed into the algorithm every frame, in order to verify if they were already mapped. Thanks to the multithreaded pipeline, verifications are done in real-time without blocking the main thread and therefore not reducing the frame rate. Our system allows the physics engine and the rendering engine to run alongside the reconstruction process.

### 4.3 Boosting real-time interaction

Once the hash table has been updated with the new entries, an optimisation stage is run to discard redundant elements. In our case, points that are contained inside any plane already detected are removed from the reconstruction. The removal can be done after checking if the point is contained inside a axis aligned bounding box (AABB) that is attached to each plane. This step reduces the number of voxels on the screen, and increases the quality of the interaction with planes. Every time a new plane is detected, this procedure is run to check if already detected voxels are contained by this new plane. If they are, voxels and all their LODs are sent to the garbage collector. Moreover, while the space is being reconstructed, a pointer to every voxel and their

subvoxels is stored inside a native array enabling high-speed access. In order to control and use each resolution, instead of recursively go through all the hash entries and either enable or disable their children, the array full of pointers for that resolution is obtained and their elements are modified. This technique optimises the treatment of the various levels of detail, and accelerates their usage.

## 4.4 Reconstruction Output

The output of our processing pipeline is a collection of voxels that are visually snapped to an invisible grid. These voxels are helpful to represent the Level of Detail 0 (LOD-0), or first-level resolution. Inside every LOD-0 voxel, different hash tables of fixed-size subvoxels are allocated to represent the several levels of detail for that area in the space. The higher the number of points detected for a specific area, the more detail it will have. Every level of detail is a uniformly distributed grid inside the highest-level voxel, where subvoxels will only be reconstructed if information is found within a specific cell. All the scene voxels and subvoxels have a box collider attached which allows physical interactions with virtual objects on the scene. Thanks to our pipeline, the hash table containing the 3D scene reconstruction includes various LODs that can be activated or deactivated for an efficient calculation of physical interactions in specific use-cases.

Only LOD-0 are attached to anchors on the scene, thus reducing the computational cost of localising virtual objects on the scene. When the camera moves around the scene, the reconstruction is anchored to a fixed position in the 3D space creating the illusion that is in the real world. The colliders needed for physical interactions overlap the real-world objects represented and therefore collisions between digital objects and physical ones will take place.



# Chapter 5

## Implementation

This chapter shows a sample application that has been built to experiment with the use of the data structure and the processing pipeline previously designed. Three-dimensional scene understanding can be done on real-time before interacting with the scene. Our implementation is built on top of the Unity3D engine, since the debugging tools it comes with allow for faster development in mobile devices. Moreover, Unity3D has the official ARCore and ARKit SDKs integrated, so extending our design to different operating systems can be easily done by changing a few lines in the code.

### 5.1 Application

We have aimed to demonstrate the use of our data structure by building an Android application with ARCore [26]. Once the application is launched, the first stage is to obtain the detected planes information from ARCore. On every frame, we query ARCore for new or updated planes and visualise them accordingly. When the user moves around the space, visual features on the objects start to be detected and the point cloud needed for our system is generated. Voxels with enough confidence to be represented will be reconstructed in the scene. However in order to improve immersion, voxels are not rendered and are only defined by a collider that is used for physical interactions.

Our application was designed to help us measure reconstruction statistics and results, and therefore a debugging interface can be enabled or disabled. The user can change

the maximum distance of the reconstruction and the size of the LOD-0 voxels by using some sliders. Besides those sliders, the rendering, 3D reconstruction, and updating of the distances can be toggled ON or OFF. When “Update distances” is activated, a thread is created to calculate the distances from the camera to each voxel in real-time. The distance value is used to update the resolution that will be represented at every single frame. If the camera is far away from the object, the resolution used will be LOD-0. The closer it gets, the higher will become the resolution of the reconstruction and more smaller voxels will appear to achieve a more accurate physical interaction. In our case, the framework is employed to set the variable `distanceToCamera` as the value to change the which resolution is used. This variable, however, can be set to anything for specific use-cases. For instance, the resolution could be adapted to the density or confidence value of that voxel: the more confidence a voxel has, the higher the resolution, and therefore the better physical interaction.

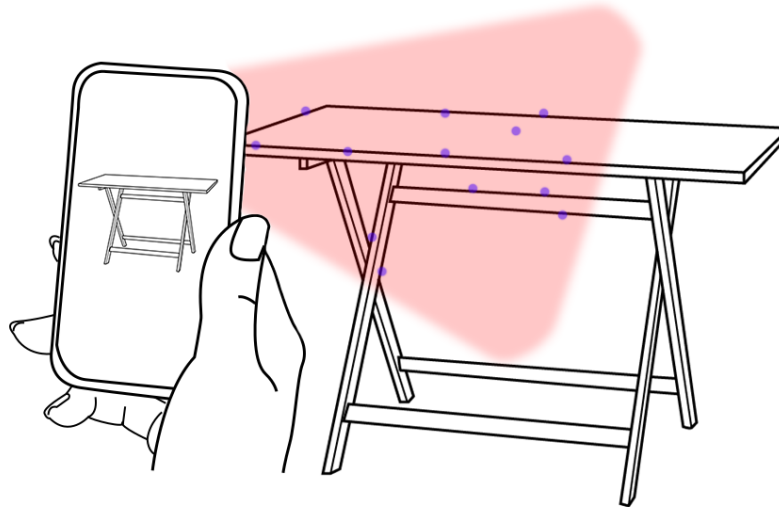


Figure 5.1: By moving the phone around the scene, visual features are detected and voxels start to appear to represent the 3D scene.

### 5.1.1 Physical Interaction

In order to demonstrate how the physical interactions can be tested with our 3D representation of the real world, digital objects can be instantiated by tapping on the screen. When tapping, the vector perpendicular to the camera image is calculated and an object is thrown with a particular force to the scene. These objects are rigid bodies with collision boxes and are affected by gravity, therefore they will bounce off the reconstructed objects and rest on the planes that are provided by ARCore.

## 5.2 Framework

Using Unity3D allowed us to create a framework that can be used for different use-cases, and that developers can plug into their applications to benefit from 3D object reconstruction. All the variables that are employed in the 3D reconstruction can be modified from our custom UI inspector.

- Confidence Threshold (%): It determines what voxels will be discarded from the reconstruction. The confidence on the voxels increases every time they are in the view frustum, and once they are higher than the threshold they will be activated for the 3D representation.
- Scale ( $\text{cm}^3$ ): It is the size of the LOD-0 voxels. This parameter will also affect the size of the subvoxels since they are children of high-level voxels.
- Camera Update Distance (cm): Distances from each voxel to the camera are only calculated if the device moves a certain distance, reducing the redundant calculations.
- Maximum Reconstruction Distance (cm): Voxels that are detected far away from the device are immediately discarded from the reconstruction.
- Levels of Detail: This variable contains the information of how many resolutions will be built during the 3D reconstruction. In our specific case, we have also set a percentage from 0 to the Maximum Reconstruction Distance that changes the 3D reconstruction from one resolution to another. Each resolution is set by

an integer number that refers to how many small cells each side of the voxel is divided into.

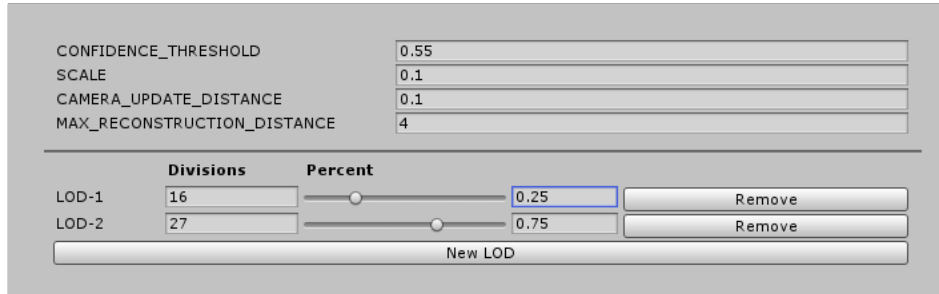


Figure 5.2: Custom Unity3D user interface for configuring the reconstruction process.

## 5.3 Rendering

In order to create the illusion that objects are placed on top of the real-world space, rendering is divided in two stages: first the camera information is rendered; secondly, the virtual objects. For the first stage, the camera image is transformed into a Unity3D texture and bounded to the shader in every frame step. The texture is rendered into a quad that covers the whole screen, creating the effect that the camera is activated. The user can now see through the device. On the second stage, we obtain the transformation matrix of the virtual objects based on the projection and view matrix that ARCore calculates ( $M_{projection}$  and  $M_{view}$  respectively) and render them afterwards (Equation 5.1).

$$M_{transformation} = M_{projection} \cdot M_{view} \cdot M_{model} \quad (5.1)$$

### 5.3.1 Occlusion of virtual objects

With the help of a transparency shader, digital objects that are behind the real ones can be occluded. Our shader sends the voxel rendering process to the back of the

rendering queue, therefore the background will be rendered on top of them even if it is located further away. In Chapter 6 we will discuss the implication of rendering multiple voxels and how it affects performance.

# Chapter 6

## Results & Discussion

This chapter showcases and describes the results of our implemented Android application, and how our adaptive data structure performs in various use-cases. The results are divided into two different sections. The first one analyses the quality of the reconstruction and the second section measures the overall performance of the system. Finally, we discuss the strengths and weaknesses detected in our system so that future research on this area can benefit from these outcomes and discoveries. The application is able to run at 60 frames per second with almost a thousand of reconstructed voxels, and multiple virtual objects being thrown to the scene. All the measurements were taken using a Samsung Galaxy S9 with Android Operating System, a device that shares architecture with most of today's smartphones. This hardware is formed by a Snapdragon 845 CPU (Octa-Core - 4x2.7 GHz and 4x1.8 GHz), 4 GB of RAM Memory and a Mali-G72 graphics card (18x850 MHz).

### 6.1 Quality of the Reconstruction and Interaction

In order to measure the quality of the interaction, we set up two different scenes to be reconstructed. The first one shows a unique object resting on a plane. In the second one, multiple unordered objects on top of a table are reconstructed. Consequently, the scene understanding results are presented with screenshots of the two abovementioned scenes, as well as attached videos that are available online (please see the Appendix A). The quality of the interaction is difficult to quantify although is directly propor-

tional to the representation accuracy: the higher the quality of the reconstruction, the better the interactions between the real world and virtual objects are. Unity3D has a high-end physics engine that handles the collisions realistically and is used by many AAA games, therefore in this specific project the quality of the interaction has not been measured by how well the engine computes collision, since we did not implement it. In summary, the quality of the interaction is related to the accuracy of the scene reconstruction, and not to the physics engine used. Figure 6.1 showcases the number of voxels and subvoxels for the different resolutions that are reconstructed in our two different scenarios. The system clusters higher-detail subvoxels into first-level voxels, reducing the number of tracked objects since only these last ones have anchors attached to them.

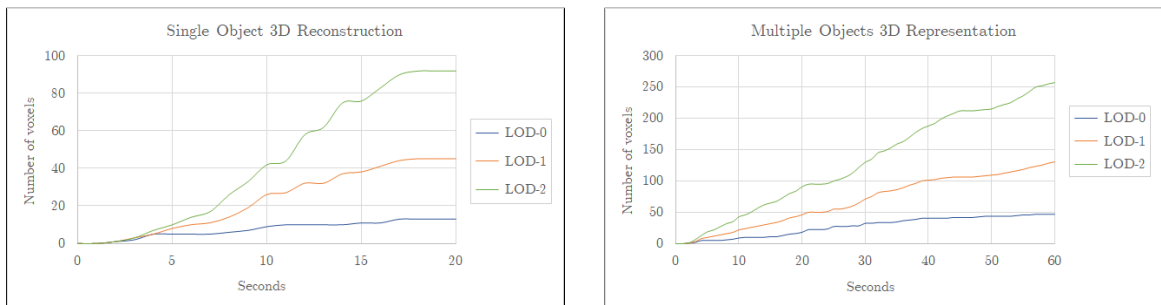


Figure 6.1: a) Number of voxels and subvoxels reconstructed when scanning a scene with a single object. b) Number of voxels and subvoxels reconstructed when scanning multiple objects. The results are divided into the three different resolutions that were scanned.

In order to scan a single object, the phone needs to be moved so that the visual features begin to be detected and voxels begin to appear. Therefore, the reconstruction is not instantaneous and takes some time to be performed, not allowing any physical interaction as soon as the app is launched. On the other hand, the plane representation provided by ARCore is detected almost immediately by calculating co-planar feature points. The reconstruction for a unique object takes approximately 12 seconds in this particular case, but it can not be precisely measured since it is highly dependent on how fast and efficient ARCore detects the feature points. Figure 6.2 shows the results of scanning the various levels of detail of a single object. Please refer to Appendix A for a complete video of the process.



Figure 6.2: a) Level of detail 0 with  $10\text{cm}^3$  voxels. b) Level of detail 1 with  $5\text{cm}^3$  voxels. c) Level of detail 2 with  $2.5\text{cm}^3$  voxels. d) Real object with only a plane being rendered. e) All resolutions rendered simultaneously.

When scanning a more complex scene, the reconstruction phase lasts around 30 seconds in which the user can not adequately interact with the scene. Once the scene has been represented in three dimensions, colliders cover the real world and digital objects can interact with them. The 3D reconstruction of a multiple object scene can be seen



in Figure 6.3 (video in Appendix A).

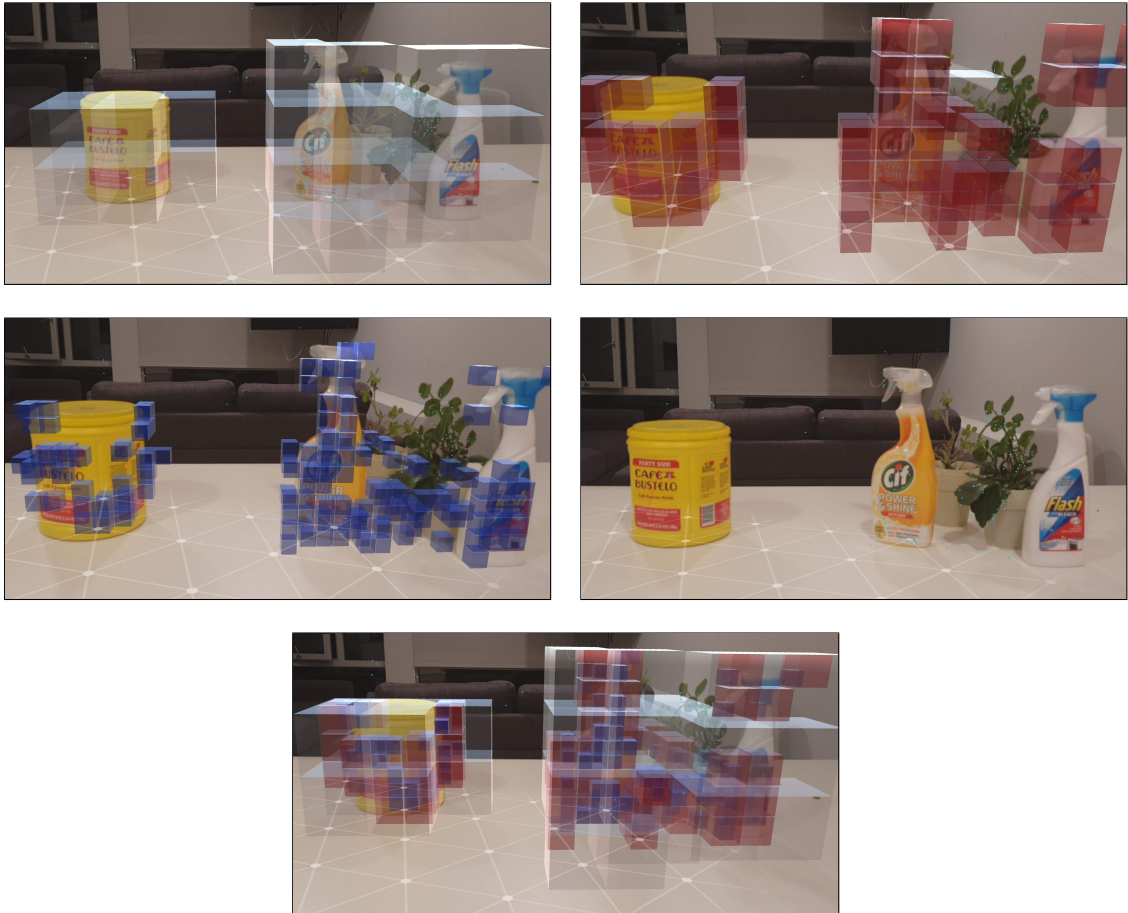


Figure 6.3: a) Level of detail 0 with  $10\text{cm}^3$  voxels. b) Level of detail 1 with  $5\text{cm}^3$  voxels. c) Level of detail 2 with  $2.5\text{cm}^3$  voxels. d) Only table plane is being rendered. e) All resolutions for the reconstruction rendered simultaneously.

In both cases, scanning single and multiple objects, three different resolutions are reconstructed for each scan. These resolutions or LODs are indistinctly activated or deactivated based on the distance of the smartphone to the objects. When the user is far away, the LOD-0 is activated while the rest are disabled. Physical interactions are less accurate, although this is not noticeable at a glance since the user is distant. As soon as the user begins to get closer, higher detail starts to appear and collisions with the objects thrown are more precise.

It is worth mentioning that our system is highly dependent on the point cloud and therefore we may obtain unexpected results on texture-less or reflective surfaces. In these kind of surfaces, the VIO system is not able to efficiently identify unique feature points, hence voxels will not be reconstructed and gaps in the objects may appear. When throwing virtual cubes to the scene for testing the quality of the interaction, some of them go through the real world elements where no feature points were detected. These gaps that appear on the reconstructed objects have a negative effect on the immersion perceived by the application user. Moreover, since we are using voxels (i.e. cube meshes) as the primitive shape for our reconstruction, digital objects may rest on a surface that is not existent in the real world, or bounce off a corner, changing their direction unrealistically.

The scale set by the user for the LOD-0 voxel size is relevant for the representation accuracy. A whole voxel can cover the full object and make its digital boundaries differ from the real ones. This problem can be solved by either changing the scale of the high-level voxel with the slider, or by setting more and smaller levels of detail to be reconstructed. The application designer needs an efficient way of determining which is the correct voxel size as soon as the app is launched, otherwise the user would need to set this value manually. Table 6.1 shows the voxels and subvoxels obtained in our two experiments compared to the number feature points that were processed from the point cloud. These results were taken while recording the video attached in Appendix A. However, in the next section, performance measurements were calculated with the screen recording disabled to maximise efficiency and remove the cost of video capture.

	<b>Single Object</b>	<b>Multiple Objects</b>
Feature Points processed	42931	132448
LOD0	13	48
LOD1	44	131
LOD2	92	262
Total Voxels	149	442

Table 6.1: Results of processing a point cloud into a adaptive hash table of voxels.

## 6.2 Performance

Throughout this dissertation, real-time performance was one of the main goals to be achieved. The proposed system has to be efficient enough to be computed simultaneously to an application or game, so that developers can use our work to create Augmented Reality experiences.

One of the motivations to build an adaptive LOD data structure was that the more objects smartphone AR is tracking and updating in every frame, the lower the frame rate is. After doing some experiments, we become aware of the need of an efficient way to store the 3D reconstruction, while trying to obtain a suitable amount of detail that maintains the quality level of physical interactions. In summary, anchors being computed by ARCore on a frame basis need to be reduced as much as possible, since a high number of tracked objects entails higher chances for the frames per second to be low. Figure 6.1 in Section 6.1 shows the different numbers of voxels and subvoxels that are generated by reconstructing a multi-resolution scene, and tracking all of them is extremely inefficient. In Figure 6.4 can be observed a higher update time when the number of anchors increases, hence the motivation to keep the number of anchors low.

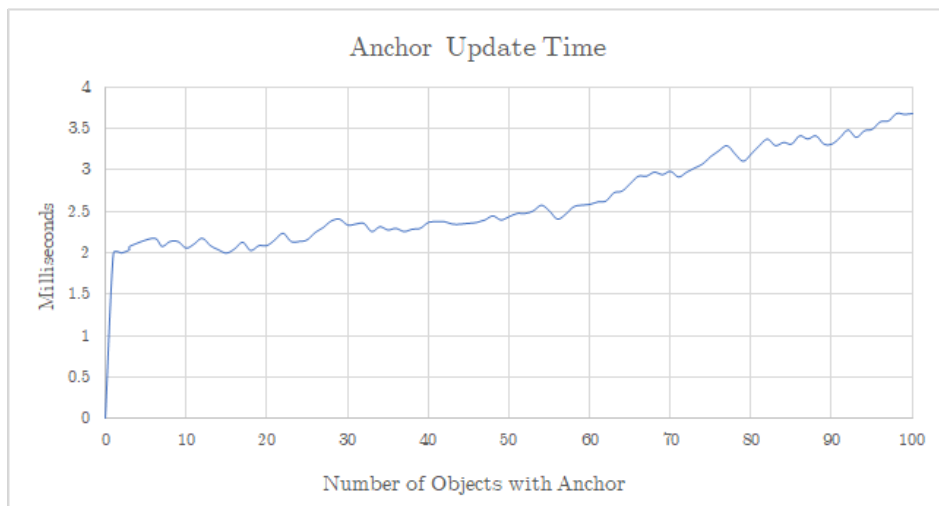


Figure 6.4: Duration of the update of all the anchors in the scene by frame.

Decreasing the computational cost does not only have an effect on the application

FPS, but also reduces the power consumption and battery drain of these mobile devices, which can be considered by developers as a key factor for using our system. In Table 6.2, we measure the time that it takes to process a frame depending on how many objects are being tracked. These measurements are taken twice, activating and deactivating the adaptive data structure we have designed. In both cases, the size of the high-level voxel is  $10\text{cm}^3$ , but when the data structure is disabled we do not create multiple LODs, and therefore the quality of the reconstruction and interaction is lower. Our system does not only allow a more detailed reconstruction, but also implies the same computational cost (anchor update time) for tracking all the digital colliders in the AR scene.

<b>Adaptative Data Structure</b>	<b>ON</b>	<b>OFF</b>
Feature Points processed	82393	82393
LOD0	25	25
LOD1	51	0
LOD2	130	0
Total Voxels	206	25
Anchor Update Time (ms)	2.2	2.1

Table 6.2: Comparison of anchor update times between our hierarchical data structure and plain Voxel-Hashing.

The multithreaded processing pipeline described in Chapter 4 helps us consistently reach 60 frames per second (FPS) while reconstructing the scene. In Table 6.3 we measure the differences in performance between a single thread processing pipeline and our multithreaded approach. Both techniques have been evaluated under the same conditions while scanning a single object resting on top of a table. The data shows how much time it takes to process and insert multiple voxels into the hash table. In the single thread pipeline every added voxel blocks the process, while in the multithreaded pipeline multiple operations are done in parallel reducing the overall computational cost.

Processing Pipeline	One Thread	Multi Thread
Time to process 100 points (ms)	1.27	0.99

Table 6.3: Comparison between processing 100 points with one thread and with multiple threads.

### 6.2.1 Unity3D

The Unity3D profiler was used in this implementation to measure performance on engine related processes that we did not implement ourselves. In this section we explain how our 3D reconstruction of the real world affected the calculations done by the physics engine and the rendering process. The results are measured by scanning multiple objects and throwing digital cubes to the scene afterwards to test physics interactions.

#### Physics Calculations

In Figure 6.6, the time per frame that Unity3D took to calculate the physics is shown. The number of voxels that compose the 3D reconstruction of the scene does not have a noticeable effect on how long the engine takes to calculate the physics and collisions.

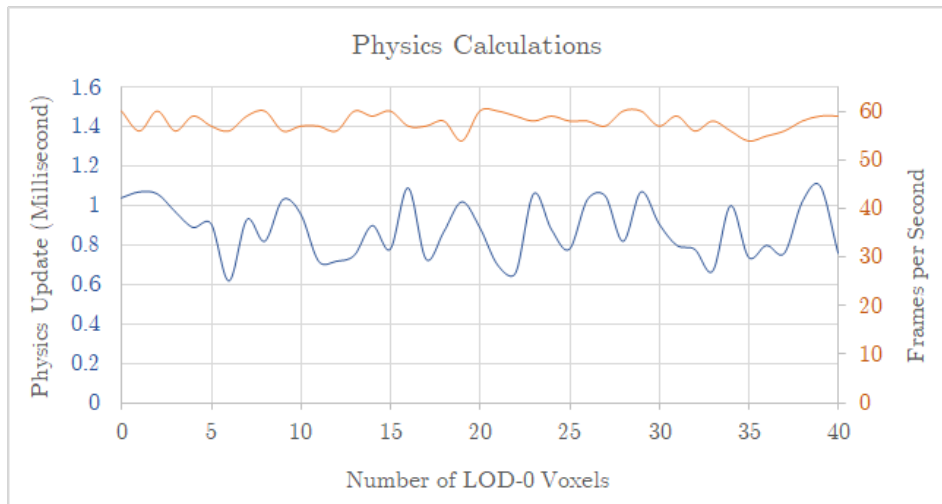


Figure 6.5: Duration of the calculations of the physics update per frame by Unity3D based on how many LOD-0 objects are reconstructed.

## Rendering

Rendering the reconstructed collection of voxels might be useful for different use-cases. Nevertheless, in this particular case, they were only rendered for debugging purposes. Later, we measured and compared the results of activating and deactivating the rendering of the objects. Even if the number of voxels that represent the scene does not have an effect on the physics calculation being performed by Unity3D, rendering causes a huge drop on the frame rate. Our implementation did not focus on rendering the voxels, since in most cases, the immersion is higher if user is not able to discern between virtual and real world. Taking into account the limitations of GPUs in smartphones, a better rendering system could be put in place by merging the voxels or using instance-based rendering, resulting on an improvement in rendering performance.

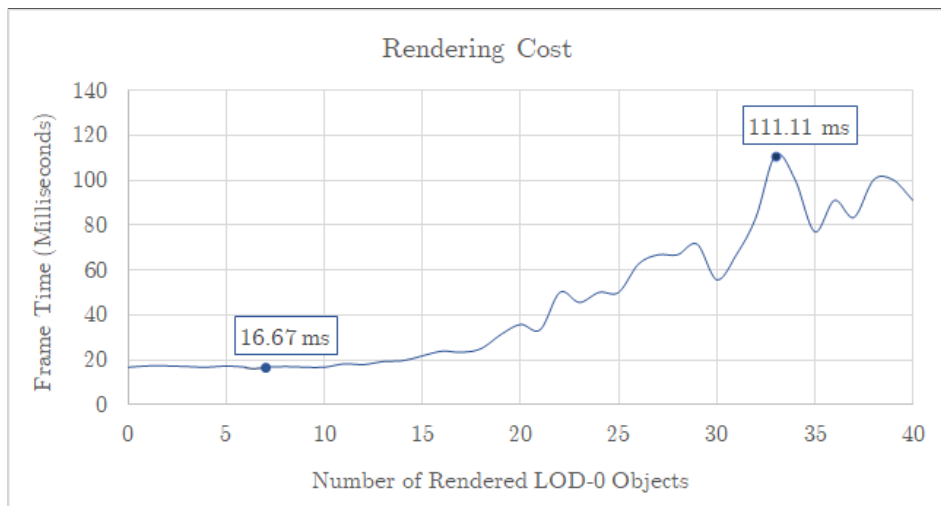


Figure 6.6: Duration of the frame rendering process in relation to the number of LOD-0 voxels reconstructed.

### 6.2.2 Memory Footprint

Considering the limitations that smartphones have in internal memory size, it is crucial to measure how many bytes are allocated for any given reconstruction. By determining the size in memory of a single voxel, the total size of the representation will be directly proportional to the number of voxels that compose it. This approach works for the regular voxel-hashing algorithm and for calculating the first LOD memory size. How-

ever, quantifying the memory footprint becomes more complicated for our adaptive LOD data structure, since the high-level voxels contain the several resolutions that the developer wants to save. The details of each block depend on the number of feature points detected for that space and it can change due to many different factors (e.g. lighting, type of surface etc.). Table 6.4 shows the results of measuring the allocated memory in two different scenarios: when our adaptive resolution mode is enabled and when it is disabled.

<b>Adaptative Data Structure</b>	<b>ON</b>	<b>OFF</b>
Feature Points processed	82393	82393
LOD-0 Voxels	25	25
LOD-1 Subvoxels	51	0
LOD-2 Subvoxels	130	0
Total Voxels	206	25
Allocated Memory (Kb)	185.48	22.05
Average bytes per voxel	922.00	903.00

Table 6.4: Memory allocated with our Adaptive Data Structure activated and deactivated. Number of voxels in different resolutions and initial feature points processed are also listed.

It is important to take into account that the size in kilobytes for every voxel was calculated by measuring the memory allocated by the application, before and after inserting a new voxel on the reconstruction. Since we are working with Unity3D for implementing our application, the size of the memory footprint includes the internal classes that are loaded by the engine for handling its processes. For example, all of our voxels are contained inside the Unity `GameObject` class, allocating unnecessary data in the memory. Nonetheless, the measurements taken can be used as reference for comparison in future work.

### 6.3 Discussion and Limitations

While 3D reconstruction and collision detection have been extensively studied in previous work, most of the research has been carried out with specific hardware like Kinect and taking advantage of the benefits of GPU computation. Our approach aims

at extending the current functionalities of smartphone AR by improving the scene understanding around the device and therefore we are constrained to the limited computational power of this kind of mobile devices. Nonetheless, our results can be used as a starting point for future work in this area, even if using alternative methods for reconstruction.

Throughout this dissertation, our main goal of improving the scene understanding of HMAR has been achieved, despite being in most cases limited to the data that could be obtained by the AR Framework, i.e. ARCore in our case. The proposed solution is based on the use of voxels, a primitive shape commonly used in 3D reconstruction, so as to build a representation of the real world objects taking as input the visual features that the VIO system finds. Even if our solution was not intended as a mechanism to obtain a realistic representation of the scene, we were able to approximate the real world for physical interactions. Instead of only colliding with the ARCore planes, collisions now occur also with real-world objects thanks to our voxels. Evidence has shown that, in cases where the surfaces are textureless or reflective, VIO fails to obtain unique feature points and our objects have gaps on their representation. When trying to interact with them, virtual elements go through these gaps and the immersion effect is lost. The quality of our technique would drastically improve by refining the computer vision algorithms behind VIO systems so that more dense point clouds can be detected. Similarly, the use of box colliders for the voxels produces unexpected collision results in some tests. The solution proposed would be to substitute the shape of the each collider from a box to a sphere, which is more often used in physical simulations. Another limitation that affected the quality of our representation is the scale of the LOD-0 voxels, which is a variable that is set by the user at the beginning of the reconstruction. This value should adapt automatically depending on the size object or scene that is being reconstructed.

In terms of performance of the reconstruction, the system is able to obtain 60 frames per second while processing and inserting our voxels into the hash table by employing our custom data structure and a multithreaded processing pipeline. For a single object scan, we were able to reduce 82393 feature points into just 25 LOD-0 voxels (Table 6.4). Rendering those 82393 points would not be efficient for a phone, and therefore



by clustering everything into voxels a real-time 3D representation is obtained. We found that there is an imbalance between the level of detail of our reconstruction and memory usage. Our system is able to create a 3D representation of the scene with different levels of details that can be changed indistinctly at runtime, but the higher the resolution, the more memory is allocated. The designed framework is also capable of efficiently reducing the number of anchors that are tracked in every frame by the AR framework thanks to the clustering of the various LODs into high-level voxels. AR-Core is still not able to use only the sensors (gyroscope and accelerometer) to track the camera pose, it needs the camera feed to find feature points. If a frame of the camera is lost, objects slide from their original position causing unexpected results. Having an efficient system is crucial in order to let the AR framework get the camera images on real-time to be processed.

Finally, even though in our application we only rendered the voxels for debugging purposes, results show that the rendering process is expensive for a high amount of voxels. On the other hand, physical calculations are barely influenced by the number of elements in the hash table since Unity3D is able to efficiently compute collisions.

# Chapter 7

## Conclusions & Future Work

### 7.1 Conclusions

Physical interactions between the real and virtual world play a crucial role in the level of immersion the user experiences in AR. The current solutions proposed by smartphone Augmented Reality lack a satisfactory scene understanding since is limited to plane detection. Similarly, the literature reviewed shows that most 3D reconstruction techniques often use special hardware, such as depth sensors and GPU-intensive computation. The purpose of this dissertation is to present a novel approach for real-time 3D spatial mapping that is specially well-suited for physical interactions in smartphone AR. The main limitations and constraints of mobile computing have been overcome thanks to the design of a custom adaptive LOD data structure and a multithreaded processing pipeline.

Our system is able to build a voxel-based representation of the scene based on the point cloud data gathered from smartphone AR frameworks, such as ARCore or ARKit. The data structure proposed allows to store several resolutions of the 3D reconstruction, reducing the number of objects computed by the AR tracking loop. All the data is saved into a hash table data structure that extends Voxel-Hashing [47], and the different resolutions or LODs generated can be activated or deactivated for changing the accuracy of physics collisions. Once the space has been represented, the user is able to interact with real-world objects, and not only with planes like in default ARCore

applications.

The point cloud is processed in real-time thanks to our multithreaded pipeline that organises threads based on the hash value of each point in the cloud. The potential of this design has been proved by prototyping an Android application that successfully reaches 60 frames per second while reconstructing a scene of multiple objects. The scene changes its resolution based on the distance from the device, and the user can test collisions by throwing digital objects to the space.

Despite the constraints found, using mobile devices did not limit our research goals, but helped us make novel and creative design choices while developing this dissertation. We hope the proposed solutions can be helpful to future work, and can be extrapolated to devices with different architectures.

## 7.2 Future Work

Despite the fact that the system designed allows us to reach promising results in 3D reconstruction for smartphone AR, there are several areas of research arising from this work which could be pursued at a later stage. Future work might address a deeper analysis of the point cloud obtained from the VIO system in order to improve the overall scene understanding. For example, clusters of points could be processed to obtain estimations of the surface normals and therefore, voxels could be rotated accordingly. A better knowledge of the point cloud would also allow reconstruction of dynamic real-world objects in real-time, something that could possibly be achieved with a data structure other than the one we implemented.

There is also room for improvement of the current model by post-processing the generated reconstruction in order to merge adjoining voxels. This will create a multi-resolution scene representation which, even though it could be more demanding to handle and store, could also be more efficient for collision detection.

A natural extension of research departing from this dissertation would be to investigate mesh reconstruction based in photogrammetry techniques. However, AR frameworks

should be expanded from their current functionalities and allow developers more accessibility to their internal computer vision algorithms in order to obtain more and richer visual features. Photogrammetry on mobile has been proved by startup companies like AboundLabs [22] with their recent demos, and seems to be a viable way of generating a mesh out of monocular RGB cameras.

Augmented Reality and Human-Computer Interaction trends [5, 63] show that there is an increasing interest in hand-tracking applications based on monocular RGB cameras. This field is very computational intensive and requires complex machine-learning algorithms, but eventually future work could explore interaction between the real world scene and the user's hand based on our line of research.

Finally, another line of research that could follow our dissertation is aimed at reducing computational cost of collision detection thanks to our data structure and the already built-in spatial hashing algorithm. Collisions with the physical objects thrown to the scene could be efficiently calculated by using the hash values of the hash table containing all the voxels in the reconstruction.

# Bibliography

- [1] Leonardo Ishida Abe, Yuma Iwao, Toshiyuki Gotoh, Seiichiro Kagei, Rogerio Y. Takimoto, Marcos de Sales Guerra Tsuzuki, and Tae Iwasawa. High-speed point cloud matching algorithm for medical volume images using 3d voronoi diagram. *2014 7th International Conference on Biomedical Engineering and Informatics*, pages 205–210, 2014.
- [2] Ronald Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6:355–385, 1997.
- [3] Adrien Bartoli. A random sampling strategy for piecewise planar scene segmentation. *Computer Vision and Image Understanding*, 105:42–59, 2007.
- [4] Matthew Berger, Andrea Tagliasacchi, Lee M. Seversky, Pierre Alliez, Joshua A. Levine, Andrei Sharf, and Cláudio T. Silva. State of the art in surface reconstruction from point clouds. In *Eurographics*, 2014.
- [5] Mark Billinghurst, Adrian Clark, and Gun Lee. A survey of augmented reality. *Foundations and Trends in HumanComputer Interaction*, 8(2-3):73–272, 2015. ISSN 1551-3955. doi: 10.1561/1100000049.
- [6] Oliver Bimber and Ramesh Raskar. Spatial augmented reality - merging real and virtual worlds. 2005.
- [7] David E. Breen, Eric Rose, and Ross T. Whitaker. Interactive occlusion and collision of real and virtual objects in augmented reality. 1995.
- [8] Marius Bulearca and Daniel Tamarjan. Augmented reality: A sustainable marketing tool?. *Global Business Management Research*, 2(2/3):237 – 252, 2010. ISSN 19475667.

- [9] Julie Carmigniani, Borko Furht, Marco Anisetti, Paolo Ceravolo, Ernesto Damiani, and Misa Ivkovic. Augmented reality technologies, systems and applications. *Multimedia Tools and Applications*, 51:341–377, 2010.
- [10] Thomas P. Caudell and David W. Mizell. Augmented reality : An application of heads-up display technology to manual manufacturing processes. 1992.
- [11] Johnny Yau Cheung Chang, Lok Yee Tsui, Keith Siu Kay Yeung, Stefanie Wai Ying Yip, and Gilberto Ka-Kit Leung. Surgical vision: Google Glass and surgery. *Surgical innovation*, 23 4:422–6, 2016.
- [12] Jiajian Chen, Greg Turk, and Blair MacIntyre. Watercolor inspired non-photorealistic rendering for augmented reality. In *VRST*, 2008.
- [13] James H. Clark. Hierarchical geometric models for visible-surface algorithms. *Commun. ACM*, 19:547–554, 1976.
- [14] James H. Clark. Estimating surface normals in noisy point cloud data. 2003.
- [15] Alan B. Craig. *Understanding Augmented Reality: Concepts and Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013. ISBN 9780240824086, 9780240824109.
- [16] Daniel Fidel Cruz-Lunagomez, Oscar Alonso-Ramirez, Antonio Marín-Hernández, Fernando Montes-Gonzalez, and Luis Felipe Marin-Urias. Floor plane recovery from monocular vision for autonomous mobile robot on indoor environments. *CONIELECOMP 2011, 21st International Conference on Electrical Communications and Computers*, pages 222–226, 2011.
- [17] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, 1996.
- [18] Daniel Dasey. Ikea place augmented reality app. <https://highlights.ikea.com/2017/ikea-place/>.
- [19] Ino Van Den and Ergen. Efficient collision detection of complex deformable models using aabb trees. 1998.

## BIBLIOGRAPHY

---

- [20] Hugh F. Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13:99–110, 2006.
- [21] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22:46–57, 1989.
- [22] Tim Field. Abound labs. *www.aboundlabs.com*, 2018. (Accessed on 08/14/2018).
- [23] Epic Games. Unreal engine. *https://www.unrealengine.com*, 2018. (Accessed on 08/13/2018).
- [24] Fabio Ganovelli, Paolo Cignoni, and Roberto Scopigno. Introducing multiresolution representation in deformable object modeling. 1999.
- [25] Thanh Giang, Robert Mooney, Christopher K. Peters, and Carol O’Sullivan. Aloha : Adaptive level of detail for human animation towards a new framework. 2000.
- [26] Google. ARCore. *https://developers.google.com/ar/*.
- [27] Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *SIGGRAPH*, 1996.
- [28] Erin J. Hastings, Jaruwat Mesit, and Ratan K. Guha. Optimization of large-scale , real-time simulations by spatial hashing. 2007.
- [29] Zhou Hui, Yi Xu, and Yuzhang Wu. Multi-scale voxel hashing and efficient 3d representation for mobile augmented reality. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.
- [30] Olaf Kähler, Victor Adrian Prisacariu, Julien P. C. Valentin, and David W. Murray. Hierarchical voxel block hashing for efficient integration of depth images. *IEEE Robotics and Automation Letters*, 1:192–197, 2016.
- [31] Xin Kang, Mahdi Azizian, Emmanuel Wilson, Kyle L. Wu, Aaron D. Martin, Timothy D. Kane, Craig A. Peters, Kevin Cleary, and Raj Shekhar. Stereoscopic augmented reality for laparoscopic surgery. *Surgical Endoscopy*, 28:2227–2235, 2014.

## BIBLIOGRAPHY

---

- [32] Jacob Kastrenakes. Google’s project tango is shutting down because arcore is already here. <https://goo.gl/BbN3eQ>, 2017. (Accessed on 08/01/2018).
- [33] Georg Klein and David W. Murray. Parallel tracking and mapping on a camera phone. *2009 8th IEEE International Symposium on Mixed and Augmented Reality*, pages 83–86, 2009.
- [34] Matthew Klingensmith, Ivan Dryanovski, Siddhartha S. Srinivasa, and Jizhong Xiao. Chisel: Real time large scale 3d reconstruction onboard a mobile device using spatially hashed signed distance fields. In *Robotics: Science and Systems*, 2015.
- [35] Emilio Lando. *How Augmented Reality Affects the Learning Experience at a Museum*. PhD thesis, 2017.
- [36] Sylvain Lefebvre and Hugues Hoppe. Perfect spatial hashing. *ACM Trans. Graph.*, 25:579–588, 2006.
- [37] Simon Lynen, Markus Achtelik, Stephan Weiss, Margarita Chli, and Roland Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3923–3929, 2013.
- [38] Francisco Madera. An introduction to the collision detection algorithms. 2011.
- [39] Federico Manuri and Andrea Sanna. A survey on applications of augmented reality. 2016.
- [40] Eitan Marder-Eppstein. Project tango. In *SIGGRAPH Real-Time Live!*, 2016.
- [41] Begoña Martínez-Salvador, Angel P. Del Pobil, and Miguel Pérez-Francisco. A hierarchy of detail for fast collision detection. In *IROS*, 2000.
- [42] César Mendoza and Carol O’Sullivan. Interruptible collision detection for deformable objects. *Computers Graphics*, 30:432–438, 2006.
- [43] Microsoft. Hololens. [www.microsoft.com/hololens](http://www.microsoft.com/hololens), 2016. (Accessed on 08/01/2018).



- [44] Paul Milgram. A taxonomy of mixed reality visual displays. 1994.
- [45] Oleg Muratov, Yury Slyngo, Vitaly Chernov, Maria Lyubimtseva, Artem Shamsuarov, and Victor Bucha. 3dcapture: 3d reconstruction for a smartphone. *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 893–900, 2016.
- [46] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew W. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011.
- [47] Matthias Nie, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Trans. Graph.*, 32: 169:1–169:11, 2013.
- [48] Evrard Freddy Ohou. Surface reconstruction: Techniques and methods from 3d points data. 2009.
- [49] Carol A. O’Sullivan. Real-time collision detection and response using sphere-trees. 1999.
- [50] Carol A. O’Sullivan, Justine Cassell, Hannes Högni Vilhjálmsson, John Dingliana, Simon Dobbyn, B. F. McNamee, Christopher K. Peters, and Thanh Giang. Levels of detail for crowds and groups. 2002.
- [51] Hyung-Min Park, Seok-Han Lee, and Jong-Soo Choi. Wearable augmented reality system using gaze interaction. *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 175–176, 2008.
- [52] Vivek Pradeep, Christoph Rhemann, Shahram Izadi, Christopher Zach, Michael Bleyer, and Steven Bathiche. Monofusion: Real-time 3d reconstruction of small scenes with a single web camera. *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 83–88, 2013.

- [53] Victor Adrian Prisacariu, Olaf Kähler, David W. Murray, and Ian D. Reid. Real-time 3d tracking and reconstruction on mobile phones. *IEEE Transactions on Visualization and Computer Graphics*, 21:557–570, 2015.
- [54] Jenny C. A. Read and Iwo Jerzy Bohr. User experience while viewing stereoscopic 3d television. In *Ergonomics*, 2014.
- [55] Joachim Scholz. Augmented reality : Designing immersive experiences that maximize consumer engagement. 2016.
- [56] Donghee Shin. How does immersion work in augmented reality games? a user-centric view of immersion and engagement. *Information, Communication & Society*, 0(0):1–18, 2017.
- [57] William Steptoe, Simon J. Julier, and Anthony Steed. Presence and discernability in conventional and non-photorealistic immersive augmented reality. *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 213–218, 2014.
- [58] Jürgen Sturm. The computer vision technology underlying arcore. <https://www.youtube.com/watch?v=1TF7esI3sMQ>, 2017. (Accessed on 08/01/2018).
- [59] Ivan E. Sutherland. A head-mounted three dimensional display. In *AFIPS Fall Joint Computing Conference*, 1968.
- [60] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H. Gross. Optimized spatial hashing for collision detection of deformable objects. In *VMV*, 2003.
- [61] D. W. F. van Krevelen and Ronald Poelman. A survey of augmented reality technologies , applications and limitations.
- [62] Lance Williams. Pyramidal parametrics. In *SIGGRAPH*, 1983.
- [63] Shahrouz Yousefi, Mhretab Kidane, Yeray Delgado, Julio Chana, and Nico Reski. 3d gesture-based interaction for immersive experience in mobile vr. *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2121–2126, 2016.

## BIBLIOGRAPHY

---

- [64] Ming Zeng, Fukai Zhao, Jiayang Zheng, and Xinguo Liu. Octree-based fusion for realtime 3d reconstruction. *Graphical Models*, 75:126–136, 2013.

# Appendix A

## Links to Videos

3D Reconstruction of a single object - <https://youtu.be/f8EGg-tjohs>

3D Reconstruction of multiple objects - [https://youtu.be/BFQM\\_Gj7twg](https://youtu.be/BFQM_Gj7twg)

Scanning a kitchen - <https://youtu.be/63mVZeER3uI>

## Source Code

Github - [https://github.com/manugildev/dissertation\\_source\\_code](https://github.com/manugildev/dissertation_source_code)

## Compiled Android Application

Application (Development Build) - <https://goo.gl/pVJyXd>