National and Kapodistrian
UNIVERSITY OF ATHENS

MEDICAL PHYSICS LABORATORY – SIMULATION CENTER

Augmented Reality for Simulation Based Training and Assessment in Minimal Invasive Surgery

Vasileios A. Lahanas

PhD Thesis

Supervisors: Evangelos Georgiou
Pantelis Karaiskos
Konstantinos Loukas

Η παρούσα διδακτορική διατριβή εκπονήθηκε υπό τον ελληνικό τίτλο «Ανάπτυξη τεχνολογιών επαυξημένης πραγματικότητας στην ιατρική εκπαίδευση με προσομοιωτές», στο Εργαστήριο Ιατρικής Φυσικής και Ιατρικής Προσομοίωσης του τμήματος Ιατρικής Πανεπιστημίου Αθηνών.

Abstract

Minimally invasive surgery (MIS) has been widely introduced into standard surgical practice, illustrating significant benefits for both patients and healthcare providers as compared to traditional open surgery. The advantages of MIS however, are accompanied by a special set of requirements on behalf of surgeons, both in terms of psychomotor as well as cognitive skills. In this context, the traditional training curricula have progressed, utilizing surgical simulation for providing a controlled environment where surgeons can safely and efficiently acquire and enhance their skills. The great technological evolution of computer science during the past three decades, and most importantly the tremendous advancements of Virtual Reality (VR), have played a crucial role towards this progression. Nowadays, acquisition of cognitive, psychomotor and procedural surgical skills is performed using state-of-the art VR simulators that allow replication of real-world scenarios into a purely synthetic environment. Augmented Reality (AR), a novel technology allowing the realistic mixture of real and virtual worlds into a common environment, is considered as an alternative to VR. During the past two decades, this technology has gradually evolved from a "toy of scientists" to an efficient tool, substituting VR in various application fields. While these also include some medical and surgical applications, the potentials of utilizing AR for training and assessment of MIS skills have not yet been exploited.

In this thesis we present what is, to the best of our knowledge, the first framework for training and assessment of fundamental psychomotor and procedural laparoscopic skills in an interactive AR environment. The proposed system is a fully-featured laparoscopic training platform, allowing surgeons to practice by manipulating real instruments while interacting with virtual objects within a real environment. It consists of a standard laparoscopic box-trainer, real instruments, a camera and a set of sensory devices for real-time tracking of surgeons' actions. The proposed framework has been used for the implementation of AR-based training scenarios similar to the drills of the FLS® program, focusing on fundamental laparoscopic skills such as depth-perception, hand-eye coordination and bimanual operation. Moreover, this framework allowed the implementation of a proof-of-concept procedural skills training scenario, which involved clipping and cutting of a virtual artery within an AR environment.

Comparison studies conducted for the evaluation of the presented framework indicated high content and face validity. In addition, significant conclusions regarding the potentials of introducing AR in laparoscopic simulation training and assessment were drawn. This technology provides an advanced sense of visual realism combined with a great flexibility in training task prototyping, with minimum requirements in terms of hardware as compared to commercially available platforms. Thereby, it can be safely stated that AR is a promising technology which can indeed provide a valuable alternative to the training modalities currently used in MIS.

This page has been intentionally left blank

Περίληψη

Η Λαπαροσκοπική Χειρουργική και οι Ελάχιστα Επεμβατικές Επεμβάσεις (ΕΕΕ) αποτελούν αναγνωρισμένες μεθόδους πραγματοποίησης χειρουργικών πράξεων, κομίζοντας σημαντικά οφέλη τόσο προς τους ασθενείς όσο και τους παρόχους υγείας. Τα πλεονεκτήματα των ΕΕΕ όμως συνοδεύονται και με μια σειρά νέων απαιτήσεων που αφορούν τις ψυχοκινησιακές και γνωστικές δεξιότητες των χειρουργών. Στοχεύοντας στην κάλυψη των εν λόγω απαιτήσεων, το κλασικό μοντέλο της χειρουργικής εκπαίδευσης εξελίχθηκε, εντάσσοντας την ιατρική προσομοίωση (ΙΠ) στα υπάρχοντα πρωτόκολλα εκπαίδευσης. Η ΙΠ προσφέρει ένα ελεγχόμενο περιβάλλον όπου οι χειρουργοί αποκτούν και εξασκούν τις απαιτούμενες δεξιότητες με ασφαλή τρόπο. Σημαντικό ρόλο σε αυτή την εξέλιξη διαδραμάτισε η ραγδαία ανάπτυξη του κλάδου των υπολογιστών και της τεχνολογίας της Εικονικής Πραγματικότητας (ΕΠ). Σήμερα, η διδασκαλία της χειρουργικής γνώσης και η εξάσκηση των δεξιοτήτων πραγματοποιούνται σε τελευταίας τεχνολογίας συστήματα ΕΠ, τα οποία επιτρέπουν την προσομοίωση πραγματικών σεναρίων εκπαίδευσης σε ένα απολύτως συνθετικό περιβάλλον. Η Επαυξημένη Πραγματικότητα (ΕΑ) είναι μια σχετικά νέα τεχνολογία που επιτρέπει την ρεαλιστική επιπροβολή ψηφιακών στοιχείων σε μια πραγματική σκηνή. Κατά τις τελευταίες δύο δεκαετίες, η εν λόγω τεχνολογία έχει χρησιμοποιηθεί ως εναλλακτική της ΕΠ σε πολλούς τομείς, μερικοί εκ των οποίων αφορούν και ιατρικές εφαρμογές. Η πιθανή χρήση όμως της ΕΠ για την εκπαίδευση και αξιολόγηση δεξιοτήτων στην Λαπαροσκοπική χειρουργική δεν έχει μελετηθεί επαρκώς.

Στην παρούσα διδακτορική διατριβή παρουσιάζουμε ένα πρωτοπόρο σύστημα εκπαίδευσης και αξιολόγησης βασικών δεξιοτήτων λαπαροσκοπικής χειρουργικής σε περιβάλλον ΕΠ. Το προτεινόμενο σύστημα αποτελεί μια πλήρως λειτουργική πλατφόρμα εκπαίδευσης η οποία επιτρέπει σε χειρουργούς να εξασκηθούν χρησιμοποιώντας πραγματικά λαπαροσκοπικά εργαλεία και αλληλεπιδρώντας με ψηφιακά αντικείμενα εντός ενός πραγματικού περιβάλλοντος εκπαίδευσης. Το σύστημα αποτελείται από ένα τυπικό κουτί λαπαροσκοπικής εκπαίδευσης, πραγματικά χειρουργικά εργαλεία, κάμερα και συστοιχία αισθητήρων που επιτρέπουν την ανίχνευση και καταγραφή των κινήσεων του χειρουργού σε πραγματικό χρόνο. Χρησιμοποιώντας το προτεινόμενο σύστημα, σχεδιάσαμε και υλοποιήσαμε σενάρια εκπαίδευσης παρόμοια με τις ασκήσεις του προγράμματος FLS®, στοχεύοντας σε δεξιότητες όπως η αίσθηση βάθους, ο συντονισμός χεριού-ματιού, και η παράλληλη χρήση δύο χεριών. Επιπλέον των βασικών δεξιοτήτων, το προτεινόμενο σύστημα χρησιμοποιήθηκε για τον σχεδιασμό σεναρίου εξάσκησης διαδικαστικών δεξιοτήτων, οι οποίες περιλάμβανουν την εφαρμογή χειρουργικών clips καθώς και την απολίνωση εικονικής αρτηρίας, σε περιβάλλον ΕΠ.

Τα αποτελέσματα συγκριτικών μελετών μεταξύ έμπειρων και αρχαρίων χειρουργών που πραγματοποιήθηκαν στα πλαίσια της παρούσας διατριβής υποδηλώνουν την εγκυρότητα του προτεινόμενου συστήματος. Επιπλέον, εξήχθησαν σημαντικά συμπεράσματα σχετικά με την πιθανή χρήση της ΕΑ στην λαπαροσκοπική προσομοίωση. Η συγκεκριμένη τεχνολογία προσφέρει αυξημένη αίσθηση οπτικού ρεαλισμού και ευελιξία στον σχεδιασμό εκπαιδευτικών σεναρίων, παρουσιάζοντας σημαντικά μικρότερες απαιτήσεις από πλευράς εξοπλισμού σε σύγκριση με τις υπάρχουσες εμπορικές πλατφόρμες. Βάσει των αποτελεσμάτων της παρούσας διατριβής μπορεί με ασφάλεια να εξαχθεί το συμπέρασμα πως η ΕΠ αποτελεί μια πολλά υποσχόμενη τεχνολογία που θα μπορούσε να χρησιμοποιηθεί για τον σχεδιασμό προσομοιωτών λαπαροσκοπικής χειρουργικής ως εναλλακτική των υπαρχόντων τεχνολογιών και συστημάτων.

This page has been intentionally left blank

Ευχαριστίες

Ευχαριστώ πολύ τον κύριο επιβλέποντα τη διατριβή μου και διεθυντή του Εργαστηρίου Ιατρικής Φυσικής και Ιατρικής Προσομοίωσης Καθηγητή κ. Ευάγγελο Γεωργίου, ο οποίος μου προσέφερε την ευκαιρία να εργαστώ σε ένα εξόχως δημιουργικό περιβάλλον, και να επωφεληθώ από την γνωριμία σπανίου ταλέντου, γνώσεων και επιστημονικής επάρκειας ανθρώπων.

Ευχαριστώ επίσης τον δεύτερο επιβλέποντα τη διατριβή μου Αν. Καθηγητή κ. Παντελή Καραΐσκο, ο οποίος προτείνοντάς μου το παρόν Εργαστήριο κατ'αρχάς, εν συνεχεία μου προσέφερε διακριτικά μια αίσθηση ενδιαφέροντος, προστασίας και ασφάλειας κατά τη διάρκεια της παρουσίας μου σε αυτό.

Ιδιαίτερα θα ήθελα να εκφράσω την ευγνωμοσύνη μου προς τον τρίτο επιβλέποντα τη διατριβή μου, Επ. Καθηγητή κ. Κωνσταντίνο Λουκά, ο οποίος συμμετείχε ενεργά σε όλα τα στάδια της προσπάθειάς μου αφιερώνοντας μεγάλο μέρος του πολύτιμου χρόνου του, προσφέροντάς μου εξαιρετική καθοδήγηση, και δείχνοντας υπέρ το δέον ανοχή και σεβασμό προς το «αυτόνομον» και «αυτοδιάθετον» του χαρακτήρα μου. Η παρουσία του κ. Λουκά στην τριμελή επιτροπή καθώς και η συνεισφορά του στην ερευνητική διαδικασία διαδραμάτισαν πολύ σημαντικό ρόλο στην εκπόνηση της παρούσας διατριβής.

Ευγνώμων επίσης είμαι προς τον Επ. Καθηγητή κ. Παναγιώτη Παπαγιάννη, παράδειγμα άρτιου επιστήμονα και ανθρώπου, για πολλούς λόγους αλλά κυρίως διότι με συμπεριέλαβε στην ομάδα του δίνοντάς μου τη δυνατότητα να διευρύνω τους ερευνητικούς μου ορίζοντες.

Ξεχωριστά ευχαριστώ τον Επ. Καθηγητή κ. Ευάγγελο Παντελή, έτερο παράδειγμα άρτιου επιστήμονα και ανθρώπου, για την καθημερινή μας επαφή, την αγαστή συνεργασία καθώς και τις συνεχείς ασκήσεις εξύψωσης του ηθικού μου. Του επιστρέφω εις πολλαπλούν τον χαρακτηρισμό «θησαυρούλης».

Ιδιαίτερο ευχαριστώ προς τους διδακτορικούς πλέον, και φερέλπιδες νέους επιστήμονες, Λουκά Πετροκόκκινο, Αργύρη Μουτσάτσο και Κυβέλη Ζουράρη, για την ζεστασιά και την ευπροσηγορία με την οποία με υποδέχθηκαν στο Εργαστήριο αυτό. Παρά τις προσπάθειες που κατέβαλαν να καθυστερήσει η εκπόνηση της παρούσας διατριβής παρασύροντάς με συχνά και μεθοδικά σε ξενύχτια, συνεπικουρούμενοι και από τα νέα και εξίσου φερέλπιδα μέλη του Εργαστηρίου, Βάσω Πέππα, Λευτέρη Παππά και Μάνο Ζώρο, τους ευχαριστώ για την παρέα, τις ενδιαφέρουσες συζητήσεις και το όμορφο κλίμα.

Ευχαριστώ επίσης τα υπόλοιπα μέλη της επταμελούς επιτροπής και καθηγητές της Ιατρικής Σχολής Αθηνών, Καθηγητή κ. Εμμανουήλ Γιακουμάκη και Αν. Καθηγήτρια κ. Ιουλία Μαλαμίτση, για την τιμή που μου έκαναν.

Τέλος, ευχαριστώ τους γονείς μου που με τη διακριτική παρουσία τους με στηρίζουν και με εμπιστεύονται. Μη παραλείψω βεβαίως ευχαριστίες και προς την Σίρι, το απόλυτο spinoff της έρευνάς μου.

This page has been intentionally left blank

Publications in international peer-reviewed journals

Lahanas Vasileios, Loukas Constantinos, Smailis Nikolaos, Georgiou Evangelos (2014). A novel augmented reality simulator for skills assessment in minimal invasive surgery. Surgical Endoscopy.
DOI: 10.1007/s00464-014-3930-y

Lahanas Vasileios, Loukas Constantinos, Georgiou Evangelos (2015). A simple sensor calibration technique for estimating the 3D pose of endoscopic instruments. Surgical Endoscopy.
DOI: 10.1007/s00464-015-4330-7

Loukas Constantinos, Lahanas Vasileios, Georgiou Evangelos (2013). An integrated approach to endoscopic instrument tracking for augmented reality applications in surgical simulation training. International Journal of Medical Robotics
DOI: 10.1002/rcs.1485

Publications in international peer-reviewed conferences

Lahanas Vasileios, Loukas Constantinos, Nikiteas Nikolaos, Dimitroulis Dimitrios, Georgiou Evangelos (2011). Psychomotor skills assessment in laparoscopic surgery using augmented reality scenarios. In: 2011 17th International Conference on Digital Signal Processing. IEEE, pp 1–6

This page has been intentionally left blank

Table of contents

Chapter 1

Introduction

During the past decades, computers have been introduced in every aspect of our lives, changing the way we communicate, work and entertain. The constantly evolving field of computer science achieved impressive advancements in both hardware and software technologies, rapidly and drastically changing the traditional scientific methodologies in almost any field of research. From classical sciences such as Physics and Chemistry to applied sciences such as Engineering, computers have substituted the traditional means of experimenting and performing calculations, allowing the modern scientific society to reach at limits that few decades ago were considered as science-fiction. The field of Medicine is not an exception to this technological revolution. Computers have been introduced into every part of modern medicine, offering state-of-the-art diagnostics apparatus such as MRI, Ultrasound and CT machines, to highly sophisticated robotic surgery devices. The concept of simulation-based training and assessment has received growing attention during the past decades, especially in minimally invasive surgery (MIS). Although the initial steps towards this direction included Physical Reality (PR) devices utilizing inanimate models of human anatomy, the technological marvels of computer science have given a tremendous boost in the potentials of computer-based surgical simulation. The development of state-of-the-art Virtual Reality (VR) platforms allowed simulation of surgical procedures in a purely virtual environment, providing a practical and safe environment for novice surgeons to obtain and improve their psychomotor skills and gain a significant level of experience before stepping into the operating room.

1.1 Thesis Statement and Goals

Augmented Reality (AR) is a relatively new technology, considered as the link between PR and VR. In AR, virtual objects are superimposed on top of real world images, producing a highly realistic visual outcome of real and virtual elements coexisting in the same environment. The concept of AR has been coined and widespread during the early 1990s and following the increasing processing power of modern computers, within two decades it has reached a level of being considered a competitor/successor of VR. A great advantage compared to VR is that it can achieve higher visual realism, mixing the real environment with virtual enhancements in a way that creates viewers with a sense of virtual objects being actual parts of a real world scene. Despite the growing evidence indicating its advantages and potentials, AR has not being yet introduced in MIS training. Contrary to the other application fields where AR is receiving increasing attention, and while computer-based training platforms have proven their value as efficient means of laparoscopic training and assessment, the benefits from introducing AR as an alternative to VR in MIS simulation training has not been investigated at the levels that is should. The primary goal of the present thesis is to

examine the potentials of AR as an alternative technology in the field of laparoscopic simulation training, aiming to bridge the gap between PR and VR simulation for training and assessment of skills in laparoscopic surgery.

1.2 Thesis Contributions

The final product of this thesis is a prototype AR simulation platform for training and assessment of fundamental surgical skills in MIS. Up to date, commercially available surgical simulators characterized as AR, only perform a limited use of its possibilities, simply enriching a PR training environment with computer-generated visual aids. On the contrary, out framework allows trainees to truly interact with virtual objects within a real scene and in real time, fulfilling all the fundamental requirements, as given in the definition of AR; Augmented reality is the introduction of superimposed graphics, audio and other sense enhancements over a real-world environment that is interactive in real-time.

Due to the diversity of the specialized fields involved in the development of a computer-based surgical simulator, our effort was put on the implementation of solutions regarding the vital parts of a computer-based surgical simulator affected by the special needs and limitations of AR. Specifically, in this thesis we dealt with and proposed solutions regarding the following technical challenges:

1. Implementation of an AR graphics engine that would allow introduction and spatial registration of virtual elements within a standard box-trainer environment.
2. Implementation of a real-time physics engine that would allow real-time interaction between virtual and real elements of the training scene.
3. Development and/or utilization of algorithms and techniques for real-time pose tracking of laparoscopic instruments within a box-trainer environment.
4. Employment of the aforementioned engines and techniques into a solid framework, which formed a basis for creating laparoscopic training scenarios that involved dynamic interaction between surgical instruments and virtual objects.

In addition to the technical challenges that we dealt with, and the solutions that were derived during the development of the presented framework, considerable work has been done in the course of this thesis for evaluating the construct validity[1] not only of the presented framework but AR technology itself. To the best of our knowledge, the training tasks presented in this thesis are the first AR training tasks allowing actual interaction between laparoscopic instruments and virtual elements of the training scene, providing a proof of concept for the potential use of it as a core technology for laparoscopic simulation training and assessment of fundamental skills such as

[1] Construct validity is one of the most valuable and mandatory assessments of laparoscopic training techniques and modalities, confirming that they can distinguish the experienced from the inexperienced surgeon based on the performance score [223].

depth perception, hand-eye coordination and bimanual operation. Finally, another significant contribution of this thesis is a calibration technique for laparoscopic instrument tracking using electromagnetic sensors attached on the instruments' handles. Utilization of the proposed technique provides real-time instrument pose tracking, which is essential in the development of training scenarios for both AR and VR platforms, as well as for the implementation of automated performance assessment methods in PR. Our method is characterized by three significant advantages:

- Supporting an instrument-sensor setup that does not affect or restrict the surgeons' freedom of motion.
- Achieving sub-millimeter accuracy in tooltip position tracking and sub-degree accuracy in instrument orientation tracking, thus allowing extraction of highly accurate metrics for objective performance assessment.
- Easy to implement, allowing non-specialized subjects to utilize it for experimental purposes.

1.3 Thesis Overview

We will first provide an introduction to MIS simulation training and assessment in Chapter 2, with a short reference on the existing simulation modalities. Additionally, the fundamentals of AR and the use of AR in medical/surgical applications will be presented. Chapter 3 will discuss the basic theory and methods of Computer Graphics (CG), the building block of AR and fundamental for the understanding of the work performed in the context of this thesis. Chapter 4 will provide a general description of the system architecture and principle of operation behind the proposed simulation framework. Chapter 5 will discuss the design and development of an AR graphics software engine, a core component of our framework, and give a detailed description on how certain technical challenges where addressed to realistic AR visualizations within a box-trainer environment. In Chapter 6, the design and development of a real-time physics software engine for the simulation of interactions between surgical instruments and virtual objects will be discussed. Chapters 7-10 will illustrate the experiments and results of our research process, as published in relative scientific journals. Finally, Chapter 11 will conclude the work and discuss future research directions.

This page has been intentionally left blank

Chapter 2

Minimally Invasive Surgery Simulation and the connection to Augmented Reality

The ultimate goal of the present thesis, as discusses in Chapter 1, is to introduce Augmented Reality into simulation-based training for Minimally Invasive Surgery. Between these two topics, which are relatively new in the relevant research field, there exists a direct link; the utilization of state-of-the art computer graphics tools and techniques. Despite this link however, not a solid work introducing AR in the field of MIS simulation has been yet proposed in the current literature. This Chapter provides an introduction to both topics, allowing the reader to identify this missing link that triggered the initial motivation for the fulfillment of the present thesis.

2.1 An introduction to MIS

Minimally invasive surgery (MIS), also called minimal access or endoscopic surgery has become a universally accepted method for performing certain surgical operations during the past three decades. Contrary to the traditional open surgery where operations are performed through a large incision on the area of interest, surgical procedures in MIS only require one or more tiny incisions at a size of a few millimeters (usually 0.5–1.5 cm) on the patients' skin. In general, the great benefits of minimally invasive operations compared to traditional open surgery are: Shorter recovery times, less discomfort during rehabilitation and a better aesthetic outcome, all due to the small size of the incisions that translate in less trauma. Amongst the numerous types of surgical procedures performed in a minimally invasive approach nowadays, the majority covers procedures in the abdominal cavity. The field of surgery involved with this type of operations is called Laparoscopy.

2.1.1 History

Although the concept of minimally invasive surgical intervention is relatively new in medical practice, the desire of examining a patient's inner body without significant injury that is nowadays called endoscopy first appeared more than two millennia ago. Hippocrates (460 – 375 BC), founder of an ancient medical school in Cos and nowadays considered as the father of Medicine, made a reference to a rectal speculum; a medical device that could dilate the opening of the rectum and allow physicians to look at the inner human body through a tube [1]. Similar types of medical apparatus were used during ancient years for the examination of all kinds of orifices existing on a human body, such as the rectum, ear, nose and vagina by the Greeks, Romans or Egyptians [2]. The credit of being the father of modern endoscopy however belongs to Philipp Bozzini, a German army surgeon who in 1805 invented an instrument for solving the problem of inadequate illumination during the examination of human orifices. Bozzini's invention, called the "Lichleiter" (Fig. 2.1), utilized artificial light coming from a candle placed inside a housing tube. This instrument was a primitive medical instrument that could be attached to different tube sizes, allowing the inspection of the human ear, urethra, rectum, female bladder, cervix, mouth, nasal cavity, or wounds [3]. Using artificial light and a mirror that reflected light to a certain direction, the "Lichleiter" or "Light conductor" is considered ancestor of modern endoscopes. In 1853, the French Antoine Jean Desormeaux modified Bozzini's invention, using a kerosene lamp instead of a candle and a long metal channel with lenses instead of the tube housing of the "Lichleiter" (Fig. 2.1). Desormeaux utilized his invention for urethra and bladder inspections, and referred to it as the "endoscope", coining a term that prevailed through the years.

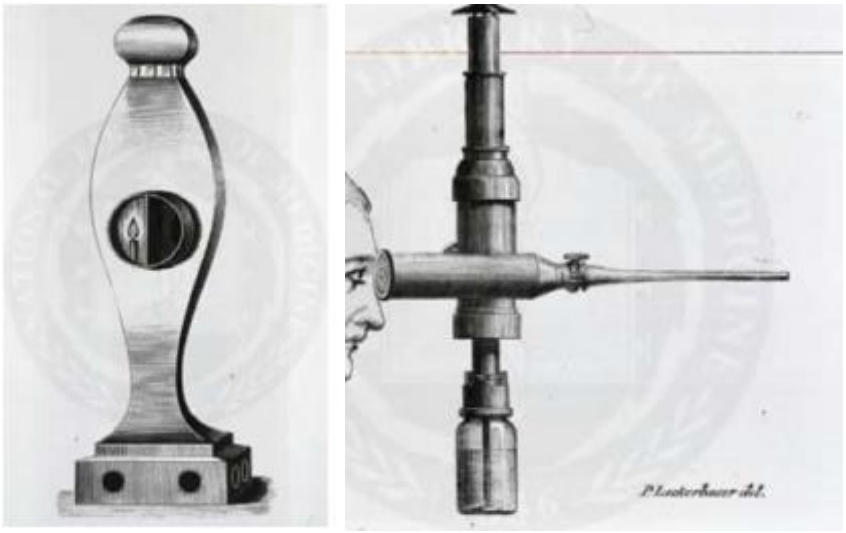


Figure 2.1 The first endoscope, called the "Lichleiter", invented by Philipp Bozzini in 1805 (left). In 1853, Antoine Jean Desormeaux used the term "endoscope" to describe his modification of the "Lichleiter" (right).

During the next decades, several inventions and innovations occurred independently and simultaneously, setting the foundations of modern endoscopy and laparoscopy. It was the invention of the solid state camera in 1980 however that ignited the world of laparoscopic surgery, allowing substitution of the traditional open surgery techniques with minimally invasive interventions, leading to the establishment of laparoscopic surgery as a standard technique in modern medical practice. As in most surgical innovations though, the answer to who is the inventor of laparoscopic surgery is debatable [4]. In most bibliographic references, Kurt Semm [5] who performed the first laparoscopic appendectomy in 1981 [1], along with Erich Mühe [6] and Phillip Mouret [4] who carried out the first laparoscopic cholecystectomies[2] in 1985 and 1987 respectively, are largely credited as the pioneers of laparoscopic surgery [1, 3, 7]. Within 5 years from the first successful laparoscopic cholecystectomy, the minimally-invasive approach was established as a feasible alternative to open surgeries [8] and in the following two decades, laparoscopy demonstrated a very rapid expansion. Nowadays, more than 98% of cholecystectomies are performed laparoscopically [9], while applied laparoscopy includes most of surgical procedures involving abdominal and chest contents [2].

2.1.2 Benefits of laparoscopy surgery

In open surgery, also called laparotomy, examinations and operations on the abdomen are performed through a large incision that exposes the abdominal cavity, providing surgeons with a direct view of the patients' inner tissues and anatomical structures. The size of such incision is large enough, not only to allow a clear view of the inner abdomen, but also to provide adequate

[2] Cholecystectomy is the surgical removal of the gallbladder

space so that surgeons can manipulate surgical instruments within the cavity. In MIS on the other hand, examination of the abdominal cavity is performed through a minor incision of up to 1.5 cm, usually around the belly button area. Through this incision, carbon dioxide can be introduced to the abdomen, inflating the abdominal walls and providing surgeons with a clear view of the patient's anatomical structures using an optical endoscope, as well as room for work. In cases where surgical intervention is necessary, one or more additional incisions of the same size are performed, allowing the introduction of long-shaped surgical instruments into the abdomen. These instruments can be used for performing a number of surgical actions such as cutting, suturing, artery clipping, cauterization etc. Figure 2.2 provides a view of a typical real-life MIS surgery, illustrating the conditions and setup of such an operation. As depicted by this figure, the surgeon manipulates the instruments while looking at a display monitor. The latter illustrates images of the patient's abdomen obtained with the endoscope. Once the operation is finished, the carbon dioxide is expelled from the abdomen and the incisions are closed using a small number of stitches.

Figure 2.2 In MIS, surgeons manipulate long instruments inserted into the patient's abdomen through small incisions, while looking at the patient through a display monitor.

As already mentioned, the great advantage of laparoscopy compared to traditional laparotomy is that overall, a laparoscopic surgery requires fewer (in terms of size) incisions. In cholecystectomy for instance, which is the most widely applied laparoscopic operation, following an open surgery approach requires an incision of approximately eight centimeters in length [10]. Following a minimally invasive approach on the other hand, the same procedure requires three to four incisions, which are however significantly smaller. This ostensibly minor difference between the

two methods, results in multiple benefits of laparoscopy compared to the traditional laparotomy. In purely medical terms, these benefits of laparoscopy are [10–13]:

1. Minimized risks of blood loss, which is a potentially dangerous implication of open surgery.

2. Reduced needs of blood transfusion.

3. Smaller chances of post-operational bleeding implications.

4. Reduced needs of long-term pain relief medication, which is commonly required in open surgery for relieving patients from stitch-healing pain.

5. Exposure of the internal organs to external contaminants is reduced in laparoscopic surgery compared with open surgery, therefore minimizing the risk of post-operative internal infection.

6. Significantly smaller scar after surgery, which also translates in fewer chances of post-operational infection.

Except for the aforementioned reasons of medical importance, laparoscopic surgery is preferred by both patients as well as healthcare providers for a number of secondary, yet important factors. Regarding the patients, faster rehabilitations means that they are allowed to return to their normal everyday lives much more quickly than after an open surgery procedure due to the shorter post-operative recovery time and shorter hospital stay [14]. Also, laparoscopic surgery provides better aesthetic outcome since the small incisions become almost invisible after the healing period. Lastly, shorter hospital stays and fewer post-operative hospital visits reduce the cost of surgical operations. In countries where healthcare is fully privatized, this allows appropriate healthcare access to a wider range of population.

Healthcare providers also benefit in many ways from laparoscopic surgery. First and foremost, the improved patient care arising from the reduced medical risks as discussed earlier, significantly improves the overall efficiency of healthcare delivery which by definition targets on constantly improving its safety records [13]. In addition, the shorter rehabilitation time leads to more surgical procedures per year and consequently to higher revenues. Although the latter might seem insignificant compared to patient safety, it is also a parameter of great importance, since healthcare revenues translate to more investments, leading in long terms to further improvements of the overall healthcare system efficiency.

2.1.3 Training and assessment of surgical skills in laparoscopy

As already mentioned, laparoscopic surgeries are performed using a rigid endoscope, introduced into the patient's abdomen through a small incision close to the belly button. Using the conventional laparoscopic imaging systems, instead of the direct 3D view of open surgery, surgeons obtain only a 2D view of the operating area through a display monitor. This monitor is usually positioned in front of them as illustrated in Fig. 2.2, and consequently surgeons must learn to operate while looking at another direction [15]. Furthermore, the restricted two-dimensional vision combined with the limited field of endoscopic view creates a lack of depth perception. To cope with this, surgeons must learn to depend on other cues in order to enhance their overall sense of depth. These cues are primarily the sense of touch and the interpretation of lights and shadows[15].

Indirect vision is not the only issue in minimally invasive surgical interventions. The standard surgical tools involved in open surgery are also not applicable in laparoscopy. Instead, long-shaped surgical instruments are utilized, inserted into a patient's abdomen through trocars[3] as Fig. 2.2 depicts. The use of such instruments adds to the difficulty of laparoscopic surgery for several reasons, the most important of which is the lack of adequate feedback. Operating in such long tools reduces or in some cases eliminates the sense of tactile feedback. Since surgeons are prevented from directly palpating organs, vessels, and tumors during the intervention, identifying target regions is far more difficult than in open surgery. Kinesthetic feedback, providing indications regarding the forces exerted on tissue, is also significantly reduced in laparoscopy [16], making it more difficult for surgeons to operate on delicate tissues such as vessels, arteries etc.

Except for reduced sense of tactile and kinesthetic feedback, the long shape of laparoscopic instruments also produces a number of additional challenges. Firstly, their long length amplifies tremor and makes them harder to control compared to conventional instruments. Secondly, the use of trocars reduces the degrees of freedom of movement [17]. Thirdly, the poor ergonomic design of laparoscopic instruments' handles makes them difficult to manipulate [18]. Another difficulty of laparoscopic surgery arises from what is called the fulcrum effect, i.e., the abdominal wall acts as a fulcrum where hands movements towards one direction result in opposite instruments movements [18]. Surgeons need to familiarize with this effect and perform actions in a totally different way compared to standard laparotomy. Due to the combination of all the aforementioned factors, highly trained and experienced specialists are required for successfully performing laparoscopic interventions [19, 20], mastering a different set of cognitive, psychomotor and visio-spatial skills [21].

[3] A trocar is a medical device made up of a metallic or plastic obturator, a cannula and a seal. Trocars are used to create and maintain the incisions during a laparoscopic surgery, allowing access to the inner abdomen for the endoscope and surgical instruments.

During the early 1990s though, the initial reports on successful laparoscopic surgical interventions brought enthusiasm to the increasingly competitive healthcare market. Motivated by the profound advantages of MIS, a large number of operators began to attempt laparoscopic surgeries [8, 22]. Along with the numerous benefits of introducing MIS into the standard clinical practice, a number of considerations were also introduced. There were doubts regarding its safety and the qualifications of those performing these procedures [22], since even the most experienced surgeons had no experience in performing or even watching such a surgery [23]. On opposite side, during its early years laparoscopy was performed by individuals trained solely in open surgery, relying on skills that where not even transferable to laparoscopic surgery. Furthermore, traditional methods of acquiring surgical skills, using the apprenticeship model, could not accommodate the new skills required for laparoscopic surgery [24]. Criticizing this lack of established curricula and its unstructured expansion, Cuschieri and Shapiro [25] wrote in 1995 that laparoscopy was "the biggest unaudited free-for-all in the history of surgery" [26].

2.1.3.1 Laparoscopic box-trainers

Based on the aforementioned considerations, in the years that followed the initial expansion of laparoscopy, the surgical community was prompted to reconsider the training strategy in laparoscopic surgery to facilitate the fundamental skills necessary for a safe performance of laparoscopy [8, 27–30]. In open surgery, the traditional training apprenticeship model allowed novices to be trained in a real life environment under the supervision of senior surgeons. It was more than evident that this model, referred to as the Halstedian principle ("see one, do one, teach one"), could not be used for teaching endoscopic surgical interventions [21, 24, 31–33]. In MIS, the supervising surgeon cannot directly guide the trainee's hands, nor intervene to rectify a complication during a real surgical procedure if a problem or difficulty arises[34].

For such practical reasons, the surgical community looked for new teaching methods that would allow surgeons to acquire fundamental skills outside of the operating theater, aiming to a safe introduction of new techniques into surgical practice [34]. Apart from the practical considerations, the ethics of teaching surgical skills on a patient were perhaps the most compelling reason for searching new methodologies, stimulating the development of curricula for teaching fundamental technical skills in a laboratory setting outside the operating room[35]. Societies, regulatory bodies and organizations, with the most important ones being the Society of American Gastrointestinal and Endoscopic Surgeons (SAGES [36]) and the European Association of Endoscopic Surgeons (EAES [37]), defined minimum requirements for those performing laparoscopic surgery, with an emphasis on training outside the operating theater [8].

Training in a laboratory setting was not a new concept in surgery and medicine. Human cadavers and animals where already utilized for teaching and practicing skills in open surgery. Both options however faced important criticism. Although the advantages of training on animal models included realism and opportunities to mimic complications, this method had been criticized a

number of reasons; it was an expensive way of training, the animals anatomy differed from that of humans, and training on animals created ethical considerations [38]. In the same way, training on human cadavers was also an impractical method due to the increased cost of maintaining the appropriate training facilities, and due to their insufficient availability[38–40]. Deterministically, the surgical community searched for alternative solutions, utilizing synthetic training modalities and stepping towards simulation-based training.

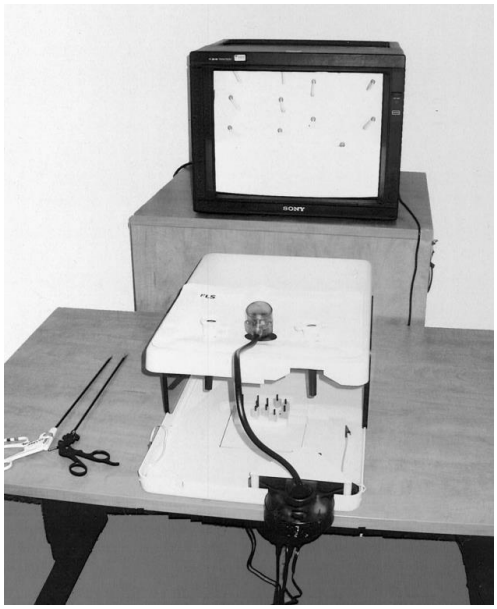


Figure 2.3 First generation of laparoscopic box-trainers consisted of a single box with holes for trocar insertion, simulation the insufflated abdominal cavity [41].

In this context, courses and drills were introduced for teaching basic psychomotor laparoscopic skills. These included the use of inanimate models, where surgeons could practice and enhance their technical skills, while supervisors could provide guidance and assess trainees' performance. On the same time, several manufacturers started producing commercial laparoscopic training platforms, called box-trainers. As illustrated in Fig. 2.3, the first version of these platforms displayed a very basic design that included a simple box with holes for trocar insertion, simulating the insufflated abdominal cavity. In these platforms, hence, surgeons could practice using real laparoscopic instruments and a camera that simulated the endoscope, while manipulating simple objects such as pegs and marbles or even applying sutures on inanimate models of human organs. Adopting a term from the field of psychology, these platforms where described as "part-task" trainers, since they deconstructed complex surgical operations into their component parts, allowing practice of individual laparoscopic surgery skills such as hand-eye coordination, ambidexterity, camera navigation etc.

Along with the development of training modalities, focus was also put in standardizing training with quantitative metrics that would allow objective assessment of skills to substitute the

subjective opinion of experienced surgeons [42]. In 1993, Reznick et al [43] presented a pioneer work, describing a curriculum for evaluating students' performance in a box-trainer platform using objective measures, the Objective Structured Analysis of Technical Skills (OASATS) [44]. Through a standardized set of drills and a specified set of metrics for each drill, the performance of each student and consequently his skills could be assessed following an unambiguous and objective analysis.

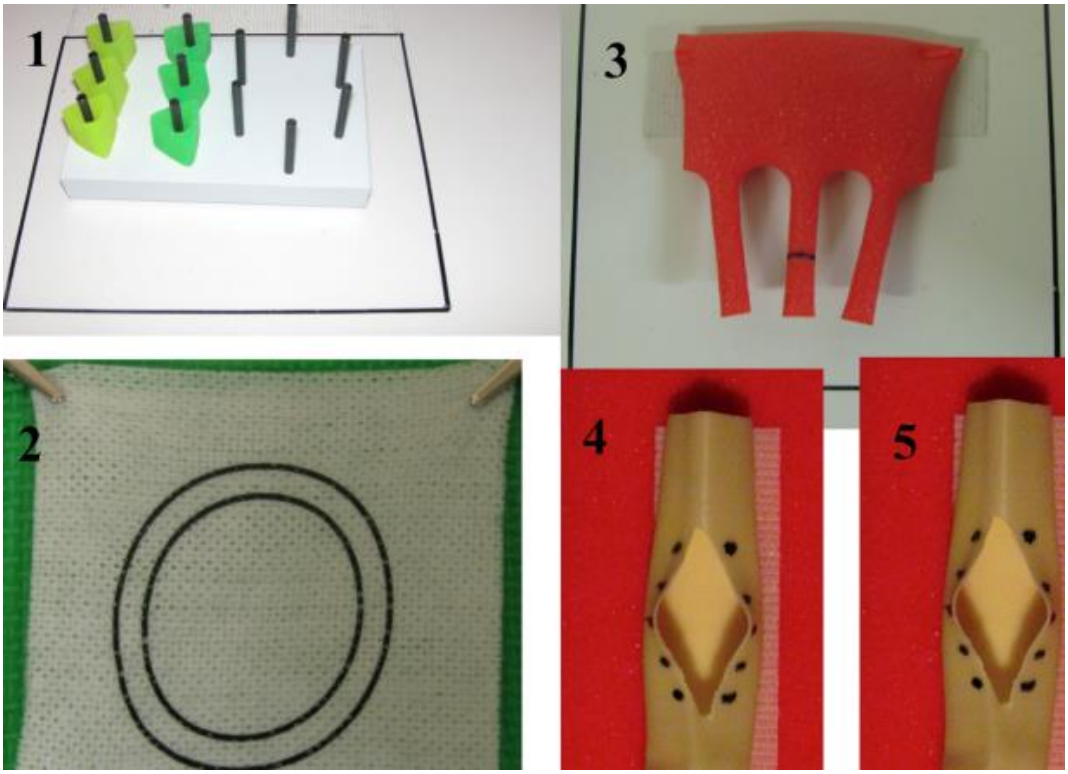


Figure 2.4 The five training tasks of the FLS program, peg transfer (1), pattern cutting (2), ligation loop (3), intracorporeal (4) and extracorporeal (5) knot tying.

A great leap towards the establishment of box-trainers in the standard surgical training curricula was the development of the Fundamentals of Laparoscopic Surgery (FLS) program, a joint educational program between SAGES and the American College of Surgeons (ACS). The FLS, adopting a commercial training system called MISTELS[4], was introduced to systematize training and evaluation of both cognitive and psychomotor skills required to perform MIS [45]. This program included both a cognitive component as well as a manual (psychomotor) component [46].

The cognitive component or curriculum of the FLS program consisted of four chapters; pre-operative considerations, intra-operative considerations, basic laparoscopic procedures, and post-operative considerations. Each of these chapters was designed to provide didactic information and provide trainees with essential cognitive knowledge of laparoscopic surgery [35, 47]. The

[4] McGill Inanimate System for Training and Evaluation of Laparoscopic Skills

psychomotor component of the FLS was a trainer toolbox that allowed testing of five pre-defined tasks: peg transfer, pattern cutting, ligation loop and suturing with intracorporeal as well as extracorporeal knot tying [35, 45, 47], illustrated in Fig. 2.4. Performed in a box-trainer, these tasks were designed for acquiring essential psychomotor skills such as picking, transferring between dominant and non-dominant hands, knot tying, precision cutting and ligating slender structures. In addition to skill training, the FLS committee also developed task-specific metrics for subjective performance assessment and evaluation [45].

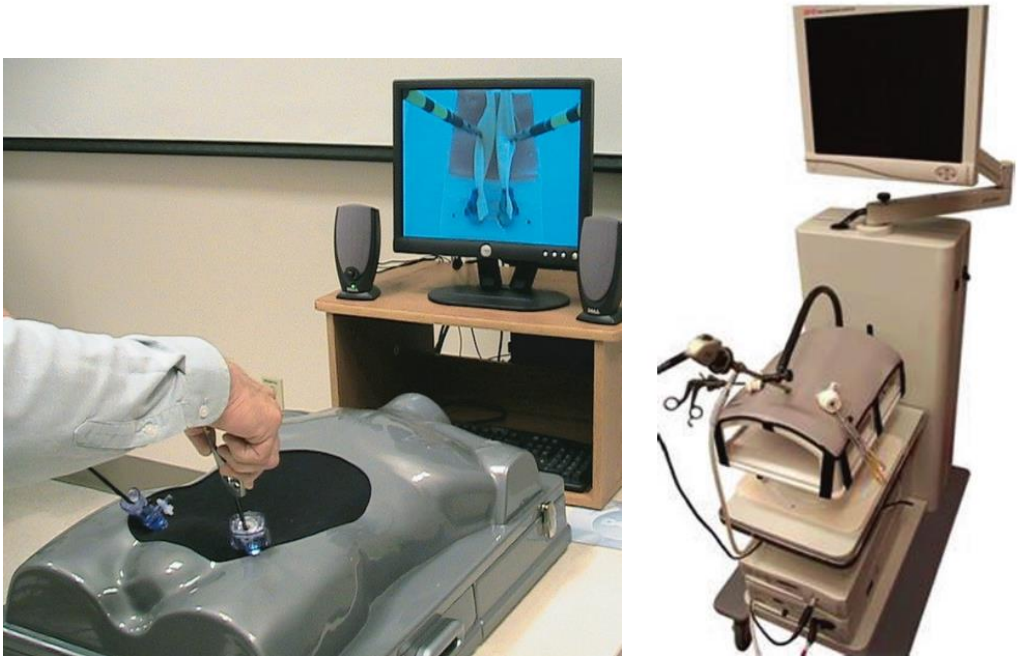


Figure 2.5 Box-trainers evolved into high-end training platforms, providing simulation in a physical reality (PR) environment.

Over the years, the primitive versions of the first generation of box-trainers evolved into sophisticated training platforms that are characterized as Physical Reality (PR) surgical simulators (also known as box trainers), since they require trainees to operate standing up within the confines of simulated anatomy such as the pelvis and upper abdomen (Fig. 2.5) [48]. On their advanced form, the simple boxes were substituted by human torso models while some high-end platforms consisted of the exact hardware setup of an operating room, including a real endoscope with accompanying fiber-optic light source and display (Fig. 2.5). In these platforms, surgeons could practice at on a setup that was very close to that of a real surgical procedure. Coupled with a proper educational curriculum, this type of training modality demonstrated high educational validity and transferability of skills to the real OR, becoming a reliable method for teaching the fundamentals of laparoscopy [35].

Except for the benefits though, there were undoubtedly problems with training in a physical reality environment. First of all, a practical limitation arose from the fact that training models required

often replacement, hence increasing the cost of training. In addition, training individual skills with basic FLS drills did not provide sufficient experience to trainees regarding actual surgical operations, which would yet require practicing on a real operating theater. Finally, physical reality simulators lacked of efficient objective assessment, since the overall process of supervising the training sessions, obtaining the scores and evaluate trainees' performance entailed a considerable amount of labor, time and cost [49, 50]. These considerations, along with the advancement of computer science and technology during the past decades, motivated the pursue for developing of computer-based laparoscopic simulation that would allow automation of training and assessment curricula and would exploit the limitless potentials of Virtual Reality (VR).

2.1.3.2 Virtual Reality Laparoscopic simulation

As a concept, VR surgical simulation existed the beginning of the 1990s. An initial approach towards the utilization of VR technology for surgical simulation was the work of Delp et al. [51] working for NASA, who described an interactive computer system that simulated the effects of surgical intervention in the musculoskeletal system of human lower extremity. The great innovation of this system was that it utilized computer graphics visualizations and the existing virtual reality technology, allowing planning and therefore optimization of operations. A pioneer work however, first discussing a "virtual reality surgical simulator", was presented in 1993 by Satava [52]. Inspired by the aviation paradigm, Satava discussed the implementation of a primitive VR surgical simulator that utilized VR hardware and software, creating a computer-generated 3D abdomen that included models of human organs (pancreas, stomach, liver, biliary tree, gallbladder and colon). Additionally, this simulator included models of surgical scalpel and surgical clamps. Visualization of virtual graphics was achieved using special head-mounted displays (HMD) while movements of surgeons hands where tracked with a device called DataGlove. According to the author, the proposed system pushed current computer technology to its limits and did not produce satisfactory results in terms of realism; however it is fair to state that Satava's work initiated the introduction of VR simulation in the field of surgical training.

In the following years, and especially during the mid-1990s, the field of computer graphics demonstrated tremendous progress. Powered by revenues of the PC-gaming industry, faster graphics accelerators and CPU technology were designed, resulting in constantly increasing levels of visual realism regarding computer-generated graphics. During this period, the innovations in computer graphics units (GPU) introduced by manufacturers such as ATI and NVidia allowed computer scientists and software developers to create algorithms that produced highly realistic virtual 3D words. Utilization of the technological advancements of this era allowed the ambitious idea of VR surgical simulation to become an achievable goal, within only a few years from the first appearance of surgical simulation in literature. The most significant steps in this direction have been taken in the area of laparoscopic surgery. Besides the need for developing devices such as those discussed previously, modeling and simulating a MIS environment presented fewer

obstacles to developers compared to what was required for creating an open surgery simulator [53].

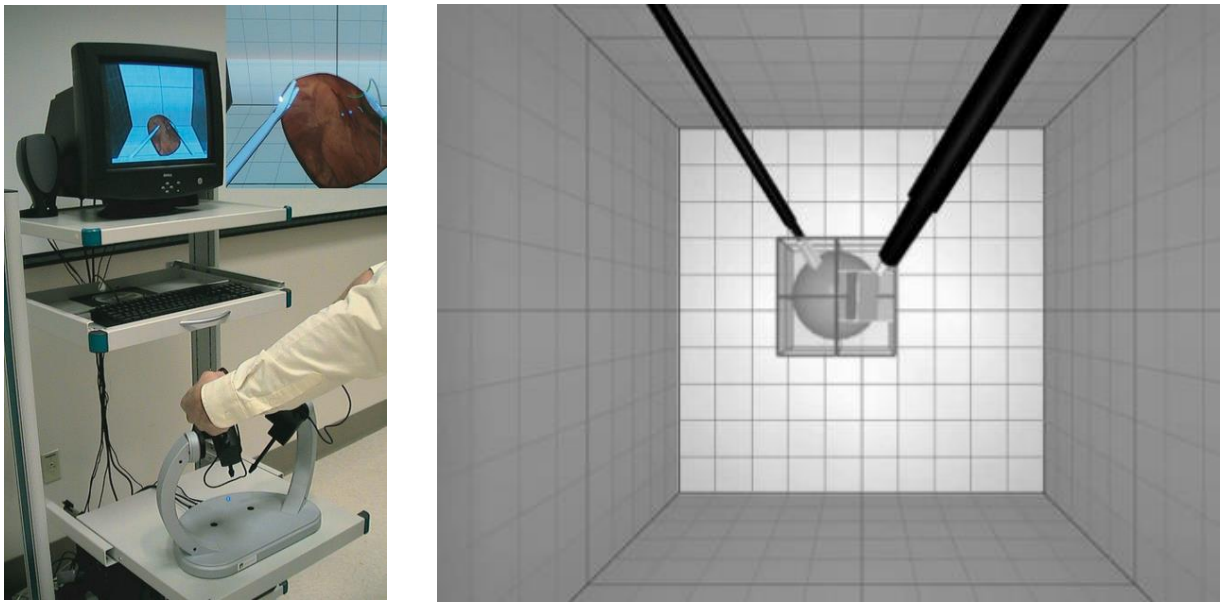


Figure 2.6 The MIST-VR simulator (left), the first commercial VR laparoscopic training platform allowing surgeons to practice in a purely virtual environment (right).

Indeed, in 1997 the first commercially available surgical simulator was presented [54]. It was a VR laparoscopic simulator called MIST-VR (Minimal Invasive Surgery Trainer – Virtual Reality), developed as a joint venture between the Wolfson Center for Minimally Invasive Therapy in Manchester and a company named VR Solutions. Although it was built upon a rather primitive with today's standards PC (32 Mb Ram, 200 MHz Processor), MIST-VR provided a realistic and assessable VR environment where trainees could practice in six purely virtual training tasks, simulating some of the basic maneuvers performed during laparoscopic cholecystectomy [54, 55]. As Fig. 2.6 illustrates, the virtual graphics of MIST-VR were rather simplistic. According to its developers [54], abstracted graphics were an intentional choice, preventing trainees from "being distracted by the appearance of 'virtual organs' which have been shown not to enhance training". As they also stated, the choice of simple graphics allowed their product to run on an affordable computer.

Nowadays however, VR laparoscopic simulators have become high-end platforms that combine cognitive and motor skills training into an integrated VR learning experience, providing unique training opportunities in a highly realistic, purely virtual environment. Except for a PC, the mechanical setup of VR consists of custom laparoscopic instruments equipped with a number of sensors that record instrument's movements, a custom device simulating the laparoscopic endoscope and a display monitor. In addition, some trainers are equipped with diathermy foot pedals. Finally, the most high-end VR trainers are equipped with mechanical feedback devices

connected to the laparoscopic instruments. This is a crucial component of VR platforms since it provides real-time haptic feedback during training, hence enhancing the overall sense of simulation realism.

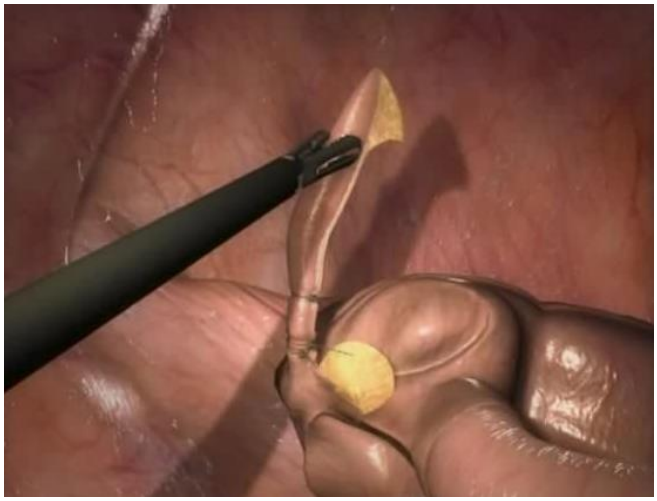


Figure 2.7 A laparoscopic appendectomy, performed in a purely virtual environment of a commercial VR trainer.

In VR trainers, surgeons can be individually guided through a series of training scenarios of progressive difficulty and complexity. This way, novice surgeons are allowed a smooth skill development as well as transition of skills from training to clinical practice. In VR trainers, a large set of different types of basic procedures can be performed, including endoscope navigation, cutting and suturing, needle driving, diathermy and other essential exercises similar to the FLS tasks described earlier. The virtual models utilized for the aforementioned training tasks can vary from simple geometrical shapes such as virtual pegs and cubes to complex ones such as virtual models of human anatomical structures.

The most high-end VR trainers also provide realistic virtual 3D reconstruction of the human abdomen, including virtual models of organs such as stomach, gallbladder, and liver as well as surrounding anatomies. In such environment, except for fundamental psychomotor and cognitive skills, trainees can perform an actual laparoscopic operation such as laparoscopic cholecystectomy, appendectomy, gastric bypass, sigmoidectomy, nephrectomy and almost any type of operation that can be performed laparoscopically. The benefits of training on such devices are profound; surgeons can perform an actual laparoscopic operation in a virtual, and hence safe, environment, before practicing their skills onto real patients in the operating table. Figure 2.7 shows a screenshot of a virtual appendectomy, performed in high-end commercial VR trainer. This figure illustrates the high levels of visual realism achieved in modern state-of-the art VR trainers, such that it is difficult for inexperienced eyes to distinguish between virtual and real.

In addition to skill learning and exercising, VR trainers also provide automated objective assessment of the performance, based on specific metrics recorded during practicing. Such metrics can be the task completion time, the number of errors (specific to each training scenario), or a wide variety of additional factors such as the use of both hands and the total length of instruments movements. In some trainers supporting simulation of actual surgical operations, these metrics might also include medical factors, for instance the total amount of blood loss, etc. Utilizing validated algorithms, VR trainers objectively assess the performance and efficiency of trainees' in each scenario, providing important feedback for their individual psychomotor and cognitive skills, as required for performing a real laparoscopic surgery.

Besides MIST-VR, previously mentioned as the first VR laparoscopic simulator, several manufacturers nowadays develop commercial VR trainers. Amongst the wide range of devices in the current market, the most widely known are:

- LapVR (Immersion Medical, USA), illustrated in Fig. 2.8(a), equipped with force feedback. LapVR offers training in basic tasks focusing on fundamental psychomotor skills; camera navigation, peg transfer, cutting, clipping and needle driving. Additionally, the platform also provides individual skills training in a set of procedural tasks; Adhesiolysis, Bowel obstruction, Bowel suturing and knot tying and Bowel loop ligation. Finally, the LapVR allows trainees the opportunity to practice on various simulated laparoscopic procedures; Cholecystectomy, Appendectomy, Bilateral tubal occlusion, Ectopic pregnancy and Salpingo oophorectomy. LapVR offers objective assessment of performance based on metrics such as time and procedure specific errors, which have been validated in relevant studies [56–58].

- LapMentor (Simbionix, USA), illustrated in Fig. 2.8(b), available in two versions with and without force feedback. Both versions are equipped with training modules that allow practice of fundamental laparoscopic skills; hand-eye coordination, 2-handed maneuvers, camera navigation and object translocation. In addition, training scenarios focusing on procedural skills are supported; clip application, cutting and suturing. Finally, this high-end platform can simulate a wide variety of actual laparoscopic procedures; Cholecystectomy, Ventral hernia repair, Gastric bypass, Nephrectomy, Sigmoidectomy and a variety of laparoscopic gynecological operations. Objective assessment if achieved with metrics such as time, economy of movement, procedure specific errors and a specific checklist relating to cognitive knowledge of the procedure. LapMentor has been assessed regarding both its construct and face validity in various studies [32, 59, 60].

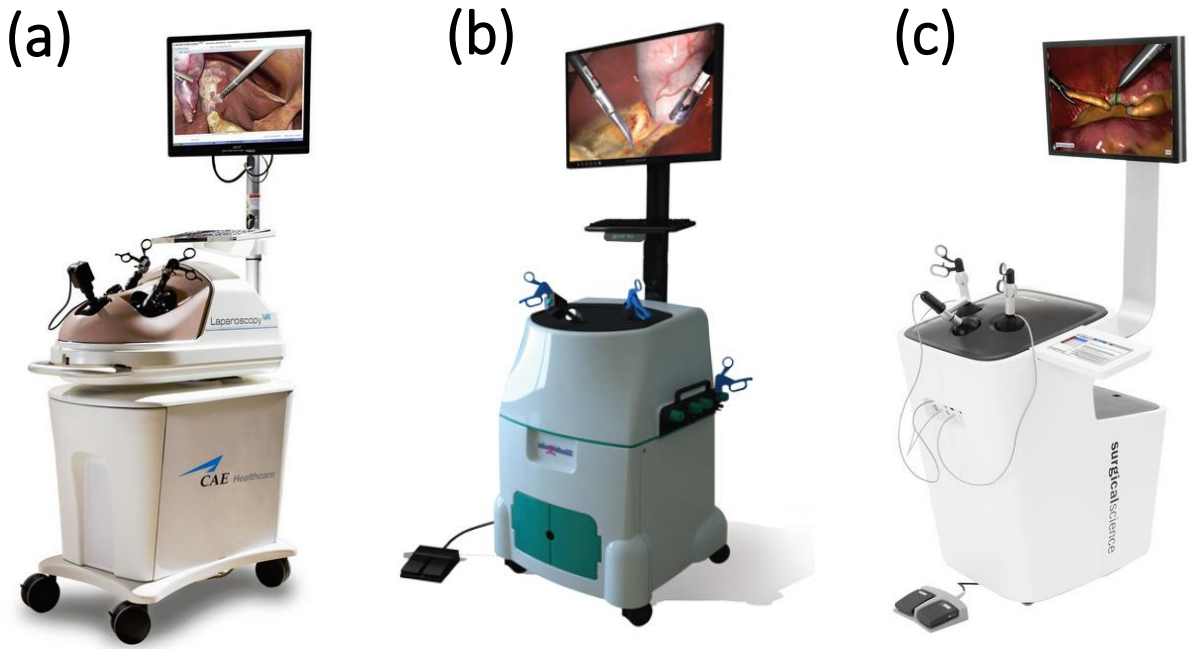


Figure 2.8 Commercially available laparoscopic VR trainers: (a) LapVR (Immersion Medical, USA), (b) LapSim LapMentor (Simbionix, USA), (c) (Surgical Science, SWEDEN)

- LapSim (Surgical Science, SWEDEN), illustrated in Figure 2.8(c), is a high-end simulator also distributed with both haptic and non-haptic hardware. This platform includes a very long list of training scenarios, probably the longest compared to other commercial solutions. From simple tasks focusing on fundamental skills such as camera navigation, instrument navigation and grasping drills, as well as procedural tasks such as clip application, catheter insertion and bowel handling. Finally, it features simulation of multiple surgical procedures. The performance metrics used to assess trainees are specific to the task being performed. These include time, instrument path length and procedure specific errors. LapSim has been evaluated in many studies and construct validity has been established [61, 62].

2.2 Augmented Reality

In the previous sections, we discussed the concept of simulation-based training for laparoscopic surgery, focusing on the two simulation modalities used in MIS training, physical reality (PR) box-trainers and virtual reality (VR) simulators. A new technology bridging the gap between PR and VR is augmented reality (AR), also called Mixed Reality (MR). AR supplements the real world by introducing virtual objects, which appear to coexist in a common environment with objects of the real world [63]. Nowadays AR is gradually evolving into a valuable alternative of VR. Several applications introducing AR as the new alternative have emerged in various fields, the most

important of which being 3D gaming. As we mentioned earlier, this field powered the rapid expansion of VR during the mid-1990s, leading to the creation of software and hardware tools that later allowed the development of VR laparoscopic simulators. Since VR and AR essentially depend on the same building blocks (computer graphics), a transition from one technology to the other would be a natural progress. And indeed, as later sections will present, this is the case for many applications aiming to introduce AR in the medical context. Despite the fact that AR is constantly gaining ground in other medical fields however, its potential utilization in the field of laparoscopic simulation training has not yet been exploited.

In order to introduce AR and allow readers to identify the aforementioned observations, which motivated the fulfillment of the present thesis, the following sections present an introduction to AR, including a brief historical review, the technologies involved and finally a review regarding the use of AR in surgical applications, primarily focusing on MIS and laparoscopy.

2.2.1 History and applications

AR was coined in 1990 from Tom Caudell [64, 65], a researcher in Boeing's Computer Services' Adaptive Neural Systems Research and Development project in Seattle. In a search to find an easier way to help the aviation company's manufacturing and engineering process in assembling aircraft wiring, Caudell invented a complex system that could overlay the positions of where certain cables in the building process were supposed to go, described it as "augmented reality". While however AR firstly appeared as a term during the early 1990's, some applications involving machine generated enhancements of the real world have already been present many decades before. The first such application was a device called Sensorama (Fig 2.9), developed in 1957 by Morton Helig. Sensorama was a machine shaped in similar way to arcade games of the 80s, designed as a cinematic experience that enhanced the senses of viewers. This machine played sounds, blew wind and created vibrations while the viewer was watching at a pre-recorded video of a bicycle ride, projected on three monitors at the sides and in front of his head. Since the projected video illustrated images of the real world enhanced with simulated effects, Sensorama mentioned in the current literature as the first AR application [65].

Figure 2.9 Sensorama, the first Augmented Reality application

An important step towards the realization of AR into a meaningful technology took place in 1966, when Ivan Sutherland and his research group consisting of students from Harvard and Utah universities invented the first head-mounted display (HMD) device [66] allowing the mixture of real-world images with (virtual) visual enhancements [67–70]. His invention (Fig. 2.10), called "The Sword of Damocles" due to its huge size and the fact that it was suspended from the ceiling of a laboratory, could produce wireframe virtual graphics on top of real world images. Whilst primitive with today's standards, Sutherland's invention was an innovation for the computer graphics community, initiating a whole new field of research around wearable/portable display systems and it is considered the first AR display [71].

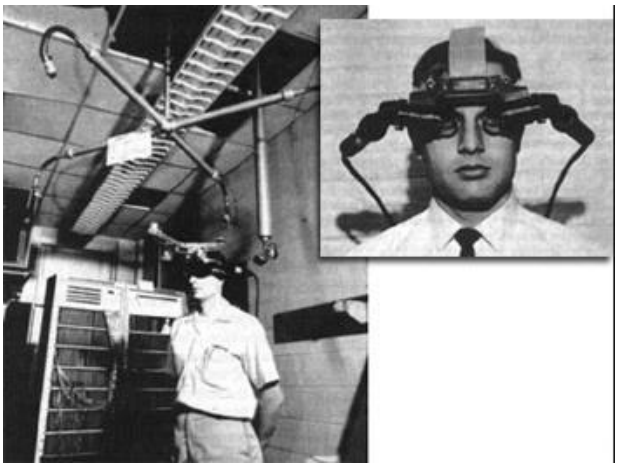


Figure 2.10 Ivan Sutherland illustrating "The sword of Damocles", the first HMD apparatus

As HMD technology advanced through the years from very bulky and impractical apparatus to portable devices that users could wear on their heads, it became a powerful tool on the hands of virtual and augmented reality researchers. Due to the invention of HMDs a wide range of exotic applications emerged, making AR almost a synonym to their use [64, 65, 67, 72].

The true potentials of AR began to appear during the early 1990s. Besides Tom Caudell, mentioned earlier as the inventor of the term “augmented reality” and the developer of the first experimental industrial AR application, two other teams made significant contributions in providing AR with a useful meaning. Louis Rosenberg created what is widely recognized as the first fully functioning AR system for the US Air Force known as Virtual Fixtures (Fig. 2.11), publishing on the same time a study regarding the use of AR for human performance enhancement [73], while a second group of researchers, consisting of Steven Feiner, Blair MacIntyre and Doree, brought out an idea of a prototype system that they called KARMA (Knowledge-based Augmented Reality for Maintenance Assistance)[64]. Their idea was to develop a system that would supply visual instructions for loading and servicing a laser printer, using virtual graphics and HMD glasses instead of a written manual (Fig. 2.11). The concept of AR was brought into public in 1994, when Julie Martin created a TV show called “Dancing in Cyberspace”. The show consisted of dancers who interacted with virtual objects, projected in the same physical space as themselves [73].

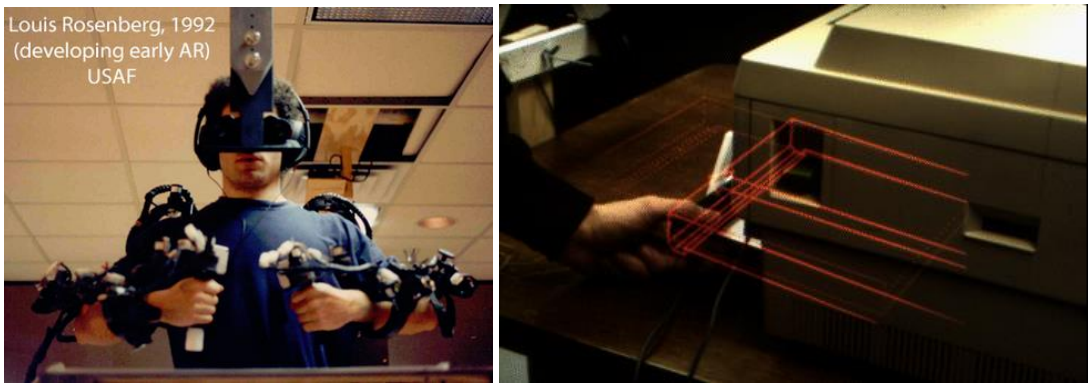


Figure 2.11 Virtual Fixtures (right) and Knowledge-based Augmented Reality for Maintenance Assistance (left), two pioneers in the development of real-life AR applications

Despite the increasing frequency of AR appearance in the literature however, not a commonly accepted description of what the term “augmented reality” stands for existed before 1994. In an effort to derive with a consistent definition, Milgram et al. [74] described AR as "*a form of virtual reality where the participant's head-mounted display is transparent, allowing a clear view of the real world*". This definition received a very large number of references in relative scientific publications, although it suffered from an important restriction; it narrowed AR to the use of HMD devices which were a trend at the time, excluding though a wide range of alternative display devices that were also capable of achieving a mixture of virtual and real elements, such as monitor-

based interfaces, monocular systems, mobile phones and various other combining technologies. To overcome this limitation and avoid relating AR to a specific technology, Azuma et al. [69] contributed a broader definition. Specifically in their work, the authors stated that in order for an application to be characterized as AR, it must fulfill the following requirements [69, 75]:

1. Real and virtual objects must be combined in a common environment.
2. The system must run interactively and in real time
3. Real and virtual objects must be spatially registered

These three rules allowed retaining the essential characteristics of AR without correlating it only with the use of HMDs. Through the years, a compact definition of AR based on the aforementioned rules prevailed in the literature: *Augmented reality is the introduction of superimposed graphics, audio and other sense enhancements over a real-world environment that is interactive in real-time* [64, 76–78].

During the late 1990s, AR became a distinct research field and AR-specific scientific conferences became to appear [65], the most important of which were the International Workshop and Symposium of Augmented Reality, the International Symposium of Mixed Reality and the Designing Augmented Reality Environments Workshop. By 2001, these conferences were united into the International Symposium of Mixed and Augmented Reality (ISMAR) which up to date is the most important scientific event for presenting advances and innovations in the field of AR.

Although the interest around the employment of AR in everyday activities was growing, AR remained mostly a subject of scientific research for many years. The expensive equipment required, as well as the level of software sophistication involved in the development of AR algorithms, posed major obstacles for its conversion from a scientists' toy to a household technology. In 2000 an important change was brought by Hirokazu Kato from the Nara Institute of Science and Technology, who released the ARToolkit library as an open source project to the internet community [79]. The ARToolkit library provided a robust software solution for real-time camera pose tracking, but also yielded a tool for the development of AR applications using a standard web camera instead of the costly display apparatus used at the time.

An additional factor in the rapid expansion of AR during the 2000s was the invention of modern generation mobile phones, called smartphones. Equipped with powerful microprocessors and cameras, smartphones allowed the implementation of AR applications using the phone's camera and monitor, as well as additional sensory inputs such as GPS, accelerometers and gyroscopes. The pioneer company in mobile AR was Mobizily, which developed the first mobile app called Wikitude. This app introduced visual augmentations of important monuments and sightings on top of the camera image, utilizing location data from the device's GPS. The public distribution of ARToolkit, followed by a number of similar software libraries such as ARTag [80], along with the

"invasion" of smartphones in our everyday lives, significantly contributed in AR becoming a widespread technology, earning its position as an alternative to VR.

2.2.2 Augmented vs. Virtual Reality

Virtual reality allows users to entirely immerge into a purely virtual environment. Augmented reality, on the other hand, maintains the real world environment, enhancing it though with the addition of virtual objects, registered at physically correct positions with respect to real objects involved in the scene [69, 75]. During the first years, AR was discussed in the literature as a counterpart of VR. Both technologies however depended on the same computer graphics algorithms and techniques. In their work [74], Milgram et al. aimed on providing a connection between virtual and augmented reality, using a simplistic visual representation (Fig. 2.12). Instead of dealing with these two technologies as being counterparts, the authors described them as lying at different locations of a single continuum, which they referred to as the RV continuum. According to this concept, VR refers to a purely synthetic environment. Reality refers to an environment that only physical objects exist. The whole range between them is covered from what they called mixed reality (MR). The latter encompasses two similar but slightly different modalities, AR and Augmented Virtuality (AV). AR describes real environments enhanced with virtual visualizations, while AV describes virtual environments enhanced with real elements (material textures etc.). Nowadays however, a single name has prevailed and the whole spectrum of mixed reality is commonly referred to as AR [69, 81–83].

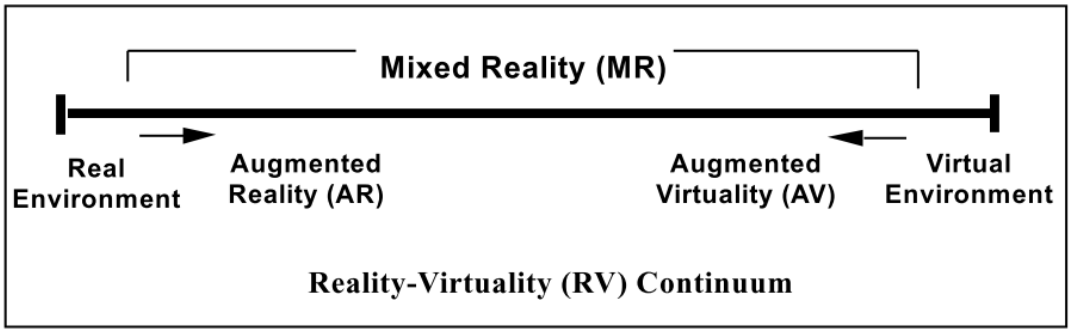


Figure 2.12 The RV continuum of Milgram et al. [74].

2.2.3 Augmented reality displays and applications

As discussed, an AR application enhances images of the real world with computer generated visual augmentations. Hence, a device for displaying such augmentations needs to be placed in the optical path between the viewer and the real world. As mentioned in the previous sections, AR was initially related to the use of a particular display technology (HMD). Through the years though, alternative display solutions emerged and AR received a broader definition that included every potential apparatus, as long as it fulfilled the specific requirements set by Azuma et al. [69].

Nowadays, a very wide variety of display devices has been employed, depending on the specific needs and purposes of each AR application. These devices are divided in three main categories; head-mounted displays, hand-held displays and spatial displays.

2.2.3.1 Head-mounted displays

As the name implies, head-mounted or head-attached displays (HMD) are devices worn on a person's head. These devices are equipped with two monitors, positioned in front of the viewer's eyes. These monitors superimpose virtual graphics on top of real-word images. Two main types of HMDs exist, optical see-through and video-see through (Fig. 2.13). Optical see-through displays allow a direct view of the real world with the use of either transparent LCD monitors or partially transparent half-silvered mirrors. In video see-through displays on the other hand, images of the real world are obtained through cameras, enhanced with virtual graphics and displayed on monitors, positioned in front of the viewer's eyes.

Figure 2.13 Optical see-through displays (left) and Video see-through displays (right)

2.2.3.2 Hand-held displays

The second category of displays includes all kinds of portable devices, as long as these are equipped with the three essential hardware requirements of AR; a processor, a camera and a monitor. The most common hand-held displays are the new generation Smartphones, Tablets and Personal Digital Assistants (PDAs) *[84]*. Being light-weight and small, such devices allow the development of AR applications intended for outdoor use, such as sightseeing and navigation apps *[85]*. Figure 2.14 illustrates an example of such application, providing the user with information regarding available internet hotspots, superimposed on top of an image of the surrounding buildings, using information from the internet and location data.

Figure 2.14 A mobile AR application for outdoor navigation and sigh-seeing

2.2.3.3 Spatial displays

The aforementioned two categories of display apparatus follow a body-attached approach, targeting in person-based AR. AR however can also be utilized in applications involving multiple viewers like museums, exhibitions, television shows etc. Such applications require detaching technology from the user and integrating it into the environment. This is primarily achieved with two types of AR displays; screen-based and projection-based displays. The first is probably the most commonly used setup in existing AR applications, since it only requires integration of a simple camera and a typical TV or PC monitor. As mentioned earlier, screen-based AR received much attention since the introduction of open-source freeware software libraries that allowed easy development of AR applications using a common PC and a webcam. An example of screen-based AR is illustrated in Fig. 2.15 where a user interacts with virtual objects superimposed on top of pattern markers.

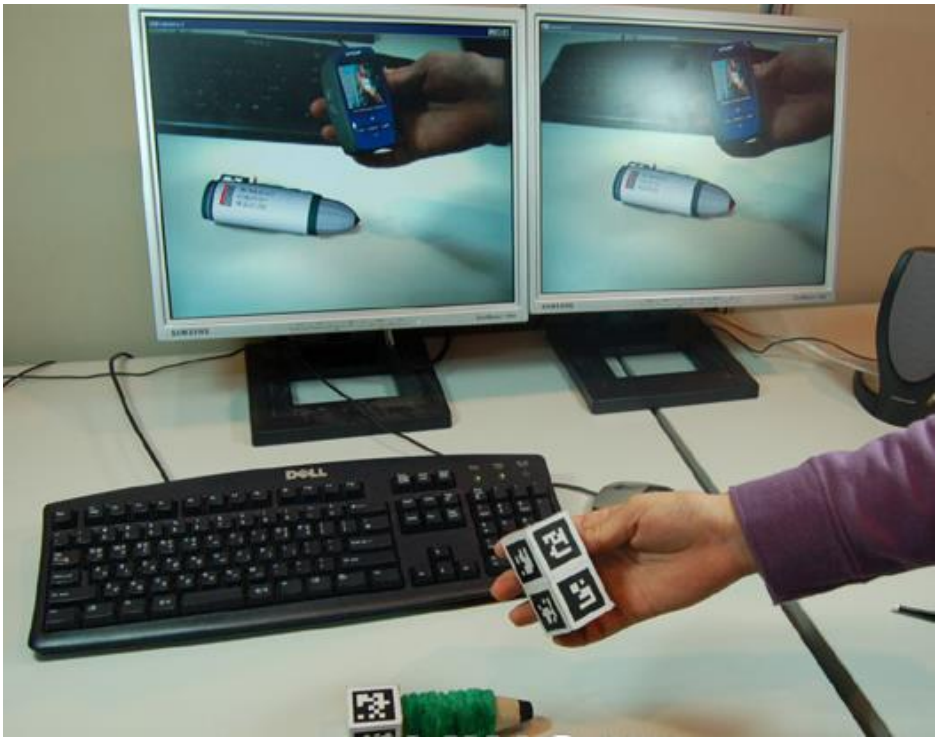


Figure 2.15 Screen-based AR, using a standard camera for image acquisition and pattern markers for 3D tracking/registration.

Projection-based displays *[63]*, the second type of spatial AR displays, directly project virtual graphics on the surfaces of physical objects, creating real-time 3D augmentations. Figure 2.16 illustrates such an example, where the mechanical schematics of a real car are augmented on top of its surfaces using a projector.

Figure 2.16 Projection-based AR

2.2.4 Augmented Reality in Surgery and MIS

In an era that pre-operative medical information became digitized through the use of MRI/CT equipment etc. [86], AR constituted the most suitable way of introducing computer generated information into a surgical environment. The definition of AR [69] as discussed in the previous sections includes three fundamental requirements. Between them, when applied to the surgical context, the most important is registration; any virtual element introduced into a scene must be spatially registered with respect to the real objects involved in this scene. Given the fact that a surgical environment is highly unpredictable, involving both rigid movable objects such as surgical instruments as well as deformable structures of the human anatomy, implementation of real-time AR requires overcoming a series of technical challenges. These challenges have to mainly deal with hardware limitations arising from the special conditions that apply in an operating theater and the high levels of accuracy required in surgical procedures [87]. Especially the latter poses a major obstacle, since sub-millimeter accuracy on the registration of virtual objects is a perquisite. The same accuracy demands also apply on the apparatus responsible for displaying virtual augmentations to the surgeon. Minor inaccuracies that might appear insignificant in other types of applications could produce catastrophic errors in a medical-oriented AR application. Since errors occurring during medical procedures are directly related to patient safety, employment of AR technology in medicine and surgery poses a very demanding field of research.

The first medical application that included spatial registration between real anatomical structures and virtual elements has been introduced in the middle of the 1980s in neurosurgery [87, 88]. Although AR did not exist as a term at that time, the proposed system had clear characteristics of AR since it spatially registered CT images with patient anatomy using fiducial markers attached on a human skull [2]. Another pioneer work bringing the combination of AR and HMD displays into the clinical environment was proposed by Bajura et al. in 1992 [89]. In this paper, the authors described a system for real-time augmentation of ultrasound images on top of a pregnant woman where real-time visualization was achieved using a small video camera mounted in front of a conventional head-mounted display (Fig. 2.17). An electromagnetic tracking system provided the spatial relationship between the ultrasound device and the HMD, allowing the observer to move the probe while ultrasound images appeared stationary with respect to the body [90].

Figure 2.17 Ultrasound images superimposed on top of a pregnant subject, using HMD and electromagnetic tracking [89]

The following years, introduction of AR in the operating room (OR) became a popular subject of research amongst scientists from the field of both medicine as well as computer science and a large number of applications focusing on different subfields of surgery were presented. The main focus among researchers has been primarily put on the use of AR for image guidance therapy. Mischkowski et al. [91] presented an AR tool for maxillary positioning in orthognathic surgery using a portable see-through device for overlaying pre-segmented CT and MRI 3D images on top of the patients skull. The position of the display device with respect to the patient is being tracked using a system of infrared optical cameras positioned on the center of the operating theater (Fig. 2.18). Nicolau et al. [92] described an AR guidance system for liver punctures, helping surgeons to reach tumors and perform Radio-Frequency (RF) treatment via the superimposition of preoperative MRI information. Their system utilizes a single camera display system and radio-opaque pattern markers for real-time tracking of surgical instruments (Fig. 2.19). Magee et al. [93] presented an navigation system for ultrasound guided needle placement training. Their proposal included generation of ultrasound anatomic images on top of an inanimate model of a human torso, using electromagnetic (EM) motion sensors for tracking the ultrasound probe and the needle. This concept has been proposed for several medical applications requiring needle intervention, such as for instance MRI-guided needle placement in spinal biopsies [94].

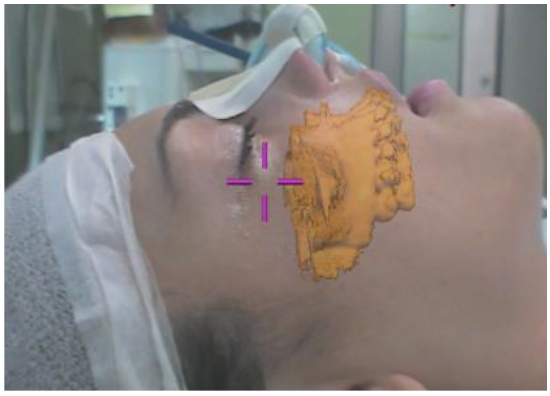


Figure 2.18 An AR tool for maxillary positioning in orthognathic surgery using a portable see-through (Mischkowski et al. [91])

Up to date, ideas employing AR for the visualization of preoperative data on top of the patient during a surgery have been widely proposed in the literature, extensively investigating the potentials of various display and tracking technologies, both in vitro [31–36] and in vivo [101]. Except for surgery though, AR has also been employed in non-surgical medical applications. Talbot et al. [102] for instance, proposed a system for patient positioning and monitoring in radiotherapy. Their system utilizes AR by acquiring live images of the linac[5] and superimposing a virtual representation of the patient body contour onto the correct position on the treatment couch.

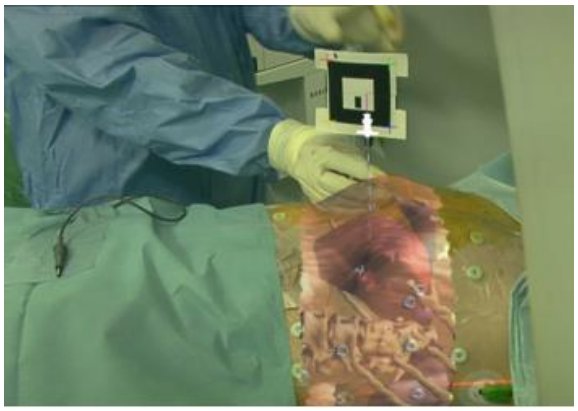


Figure 2.19 An AR guidance system for liver punctures (Nicolau et al. [92])

However, among the various fields of surgery where AR has been employed, MIS is probably the most popular one. The reasons for this are quite straightforward; the great advantage of endoscopic surgery, minimum intervention that translates in reduced trauma and consequently improved patient care in terms of rehabilitation as well as infection risks, aesthetics etc.,. On the other hand, the very nature of MIS involving the use of an optical device introduced into the patient's abdomen and a display monitor guiding surgeons to carry out surgical procedures, yields

[5] A linear accelerator (LINAC) is the device used for external beam radiation treatments for patients with cancer.

an ideal setup for the implementation of AR. Consequently, similar to the system devised by Caudell for assisting electrical engineers in the Aviation industry, AR could be utilized for enhancing the surgeons' field of view and minimize the lack of visual perception by augmenting pre-operative patient data on top of the patient's anatomical structures. A system achieving this in a robust and accurate way would be a tool of major importance in the hands of surgeons, leading to safer operations and minimized implication risks.

The first such proposal however did not appear before the late 1990s when Freysinger et al. developed an image guidance system for endoscopic ENT surgery [103]. Their system achieved real-time superimposition of a 3D path towards predefined targets on top of endoscopic images. On the same period, Fuchs et al. [104] proposed a system that obtained the 3D surface of patient's anatomy using a 3D endoscope and superimposed its virtual reconstruction on top of the real anatomy using HMD glasses instead of the typical monitor involved in laparoscopic surgery. Also targeting in MIS, Traub et al. [105] proposed a system that would help surgeons in achieving optimal port placement and intra-operative navigation in robotically assisted minimally invasive cardiovascular surgery as Fig. 2.20 depicts. Similar studies, focusing on accurate MIS port placement using AR-based guidance systems commonly have be reported in the literature [106].

Figure 2.20 Optimal port placement in cardiovascular surgery using AR visualizations (Traub et al. [105])

Despite however the constantly increasing number of relative scientific publications, the introduction of AR into MIS still faces significant difficulties. Nicolau et al. [86] characterized the possibility of providing augmented reality information in the endoscopic view as the holy grail for surgeons and a great challenge for augmented reality researchers. In their work, Nicolau et al. focused on three critical issues and their potential solutions; display technologies, real-time tracking methods and virtual objects registration techniques. These were, then and now, the main technical challenges for any researcher aiming to introduce AR in MIS, while almost each possible combination of existing solutions has been proposed in the current literature. The solutions

proposed in literature for each of these challenges, summarized in the following paragraphs, require significant improvements in terms of accuracy and robustness, in order to reach a sufficient level for bringing AR systems into the OR [107].

Regarding the display technologies, although as previously described a wide range of AR displays are available, not all of them are applicable in MIS. Mobile devices and portable projectors for instance are obviously excluded since occupying surgeons' hands during an operation is not an option. Alternative solutions such as the use of HMDs [108, 109] worn by surgeons while operating on a patient, or see-through displays [110] where pre-operative data are projected onto a semi-transparent surface laying between the surgeon and the patient have been proposed in the literature. These options however, alter the typical way MIS is performed, introducing additional apparatus in the OR. A display monitor on the other hand is already employed in a typical MIS setup. Hence, the preferable method for providing pre-operative patient information to surgeons is through the video streams displayed on the screen (screen-based AR) [111–113].

The additional technical challenges [86], spatial registration and 3D tracking, are common to surgical applications involving augmentation of virtual information on top of real anatomical structures. In MIS though, both issues are of critical importance since surgeons obtain a close view of the operating area and hence even the smallest inaccuracies heavily distort visual perception. Spatial registration in endoscopic soft tissue surgery poses a special challenge since established optical and electromagnetic tracking devices only deal with rigid structures and are not readily able to track soft tissue deformations [107]. A number of approaches have emerged aiming to overcome these challenges including intraoperative image acquisition with ultrasound, MRI or CT, or approaches using endoscopes as sensors to track artificial or natural features [114, 115].

In addition, utilizing 3D tracking techniques that apply in other surgical procedures is more challenging in MIS. The operating space is limited and consequently, inserting devices such as electromagnetic sensors or infrared-optical sensors into the patients' abdomen is not practical [116]. On the other hand, a typical MIS operation involves a limited number of movable objects (surgical instruments), which perform controlled movements and appear in specific shapes and dimensions with respect to the endoscopic camera. Thereby, an ideal approach would implement vision-based techniques for real-time tracking of the instruments kinematics. Several instrument tracking techniques have been proposed, either targeting on AR laparoscopic applications [117], or other types of applications that require knowledge of the instruments' kinematics [118–123]. Despite the spatial limitations of MIS described earlier, instrument tracking methods involving the use of optical sensors [124] , electromagnetic sensors [125, 126] or pattern markers attached on the instruments' handles [127, 128] have also been proposed.

2.2.5 Related work, augmented reality in laparoscopic simulation training

This thesis investigates the potentials of introducing AR in the context of laparoscopic simulation training and assessment. Although AR has been applied in real operating environments, regarding simulation training it is fair to say that not an actual AR laparoscopic simulation platform exists, either at a research or a commercial level. Although the technology involved in the design of such a system is relatively similar to the methods employed in commercially available VR simulation platforms for laparoscopic training and assessment, there is a clear gap in the current literature for AR laparoscopic simulation platforms. The first appearance of a scientific publication focusing on AR-based laparoscopic simulation, was published in 2005 [129], evaluating the construct validity of the ProMIS laparoscopic simulator by Haptica, later acquired by CAE [130]. ProMIS is a commercial laparoscopic simulation platform that according to its specifications provides an AR training environment, enabling users to interact with virtual and physical models in the same unit while providing accurate, comprehensive feedback on performance. Indeed this platform is the first, and up to date the only commercial simulator to introduce VR elements in a PR training environment. However, virtual augmentations are employed for guidance purposes (e.g. indicating where a tissue cut should be performed), or for achieving enhanced visual effects (e.g. adding bleeding effects at points where a cut has been performed), as Fig. 2.21 depicts, without though implementing any interaction between the instruments and virtual objects whatsoever.

Several studies have been performed to evaluate the construct and face validity of ProMIS in comparison to other available modalities (VR or PR) [131–135], presenting promising findings regarding the advantages of AR compared to other modalities used in laparoscopic simulation training, as illustrated in Table 2.1. In the most cited amongst these studies, Botden et al. [131] compared ProMIS with a state-of-the-art VR laparoscopic simulator, LapSim by Surgical-Sciences [136]. The outcome of this study allowed the authors to conclude that due to better realism and improved haptic feedback, ProMIS AR laparoscopic simulator outperformed LapSim VR platform in terms of didactic value and construct validity. Based on their results, the authors recommended the implementation of ProMIS in the training curricula in laparoscopic skills for surgical residents.

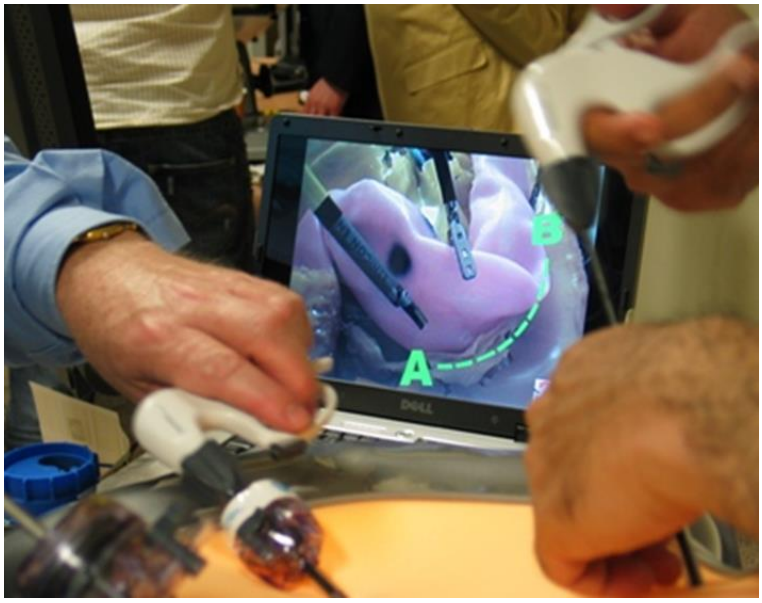


Figure 2.21 ProMIS training platform, employing AR for visual guidance in a PR simulation environment

In another work, Botden et al. [135] presented a review and classification of the existing AR simulators. Aiming to provide an overview of the existing AR laparoscopic simulation market, the authors compared commercially available training platforms that according to their claim, fitted the characteristics of AR: ProMIS, Blue Dragon [137], LTS3-E [138] and CELTS [139]. However, the authors considered as AR any system that enhanced purely physical simulation training with any kind of virtually generated information, such as performance assessment results automatically obtained and processed by a computer. And although these systems indeed target on computer-enhanced laparoscopic simulation, only ProMIS suits the fundamental requirement of AR, which is the combination of virtual and real elements in a common environment [69].

Table 2.1 Advantages and disadvantages of PR, VR and AR in MIS simulation

	PR	VR	AR
Advantages	-Realistic haptic feedback -Cost-effective setup	-Objective assessment -Interactivity	-Realistic haptic feedback -Objective assessment Interactivity
Disadvantages	-Subjective assessment -Lack of interactivity	-Unrealistic haptic feedback -Lack of assessment protocol	-Lack of assessment protocol

Despite the several studies indicating the potential advantages of AR against VR (Table 2.1), little effort has been put on the development of a true counterpart to existing training modalities. At the time the present thesis was written, a search using the keywords “laparoscopic simulation” and “augmented reality” in the largest database for life-sciences publications (http://www.pubmed.org) returned only 35 results, including the works published in the context of this thesis. This obvious gap in the literature, indicating that not a solid work towards the development of a true AR laparoscopic simulator existed, was the main motivation behind the current thesis. To the best of our knowledge, the outcome of this thesis is the first laparoscopic simulation platform allowing real interaction between surgical instruments and virtual objects, converting a PR box-trainer into a fully interactive AR training environment. Towards this goal, we have utilized technologies from computer graphics (CG), VR, physics-based modeling as well as various instrument tracking techniques, in an effort to derive with a fully functional setup. We have designed basic surgical training scenarios and evaluated construct validity of the presented framework in training and assessment of fundamental laparoscopic surgery psychomotor skills such as depth perception and hand-eye coordination.

This page has been intentionally left blank

Chapter 3

A Computer Graphics Primer

Computer graphics is a broad term referring to the mathematical principles, software algorithms and programming techniques focusing on the creation and visualization of virtual content in a display monitor. This Chapter is an introduction to the concept of computer graphics, discussing the fundamental principles and tools utilized to create and visualize VR environments in a digital monitor. It presents the building blocks of 3D computer graphics, and provides a detailed description of the fundamental mathematical theorems and principles behind the construction of 3D virtual worlds. Finally, this chapter includes a step-by-step explanation of the sequence of operations applied in VR rendering, called the *rendering pipeline*.

3.1 Modeling in 3D

A virtual world is composed of 3D models that represent real-world objects. In computer graphics, a 3D model is described as a collection of geometric primitives such as points, lines and polygons (*triangles*, *quadrilaterals* or other simple *convex polygons*) [Mario Gutierrez]. Placing polygons in various spatial arrangements, allows the creation of surfaces that can represent even the most complex shape. The structure that contains information regarding polygons and their spatial arrangement is called a *polygon mesh* and the process of designing a 3D shape with the use of polygons is called *polygon meshing* or *polygonal modeling*[140, 141].

Later chapters of this dissertation will demonstrate how essential components of an AR surgical simulator, from solid models of surgical instruments to deformable models of human anatomical structures, can be realistically visualized using *polygon meshing*. The most famous example however of *polygon mesh* is the one illustrated in Fig. 3.1. This mesh, called the *Utah Teapot*, has become the object of reference throughout the years in the computer graphics community, as it is the most commonly used paradigm for demonstrating the results of rendering algorithms and techniques [142]. The shape of this teapot is defined as a collection of triangles, topologically arranged to form the outer surfaces of a real-world teapot.

Figure 3.1 The Utah Teapot

Polygonal modeling is applied in almost any computer graphics application that deals with 2D or 3D virtual scenes, from simple applications aiming to visualize basic objects and shapes, to sophisticated applications that achieve photorealistic rendering of highly detailed 3D scenes.

3.1.1 Resolution of a polygon mesh

When it comes to real-time computer graphics, where the user interacts with the virtual world such as 3D games, the efficiency of the rendering engine is not evaluated only in terms of rendering quality but also but also on the rendering speed, commonly expressed in values of

frames per second (fps). Based on observations, the minimum frame-rate for an application to be considered real-time is approximately 30 frames per second.

The resolution of a virtual model refers to the number of vertices, which are points in the 3D space used to describe a model's geometry. As later sections will explain, drawing a 2D image of a virtual scene requires a large number of mathematical operations. These calculations are performed in a per-vertex basis, using the CPU (central processing unit) and the GPU (graphics processing unit) of a PC. Since both the CPU and GPU have limitations regarding processing speed, amongst other parameters such as the calculation of lighting and shadows, the frame-rate is heavily affected by the resolution of the rendered models.

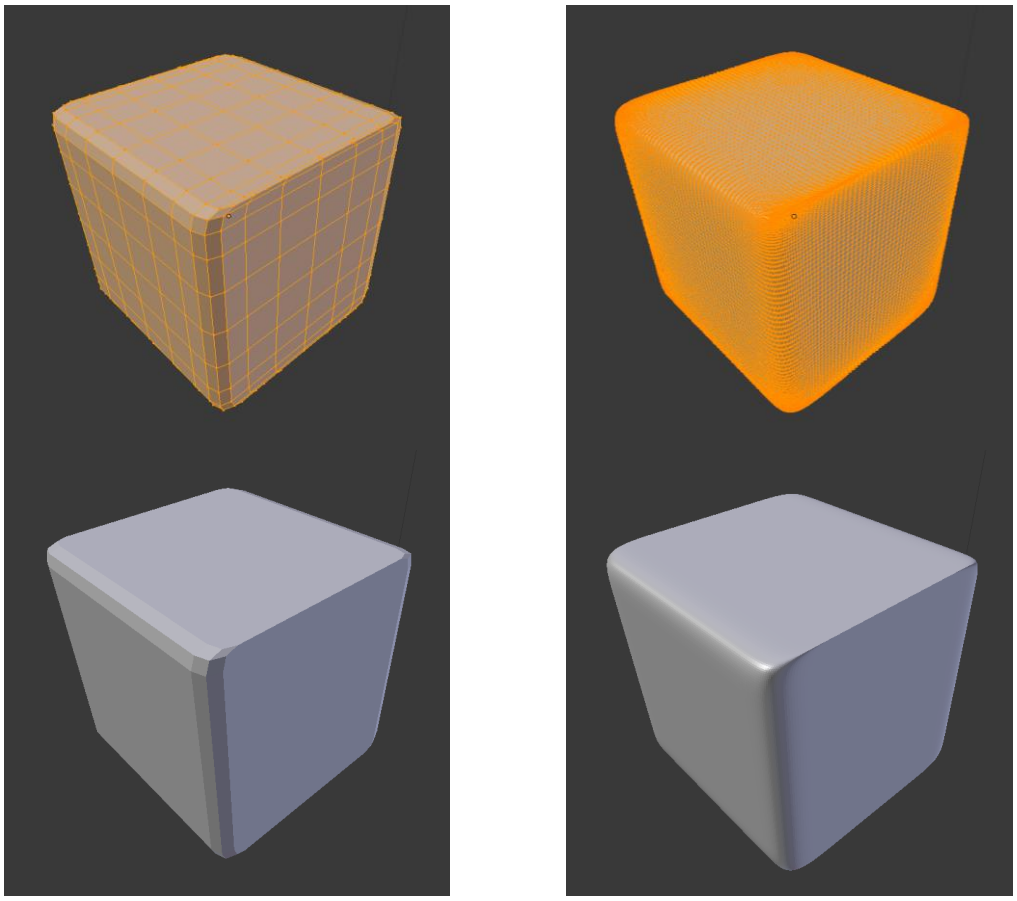


Figure 3.2 A low-resolution mesh (left) vs. a high-resolution mesh (right).

A part of the developer responsibilities during the design of a computer graphics application is to find a compromise between rendering quality and rendering speed. Greater modeling resolution allows the creation of more complex and smooth geometries as demonstrated in Fig. 3.2. However, depending on the specific needs of each application, the best approach is the reduction of the total number of a model's vertices up to the point that the resulting shape still corresponds to the desired geometric characteristics of the virtual model.

3.2 Essential Mathematics of Computer Graphics

Regardless of the quality and complexity of the rendering outcome, the fundamental operation that a computer rendering engine must be capable of performing is to draw polygons on a device monitor. Hence, any computer graphics application needs to be equipped with the mathematics tools that allow the description of polygons in the three-dimensional space and the projection of those on a two-dimensional screen. The mathematical principles behind computer graphics are relatively simple, yet powerful.

3.2.1 The Cartesian Coordinate System

The real world, as humans perceive it, has three dimensions. In mathematics and geometry however, space can be either two-dimensional ($\mathbb{R}^2$) or three-dimensional ($\mathbb{R}^3$), named *Euclidean plane* and *space* respectively, after the ancient Greek mathematician Euclid of Alexandria. Determining the location of a point in the Euclidean space requires a numerical representation that corresponds to the spatial relationship between this point and a known origin. Mathematicians have developed the tools for achieving such representations, in the form of coordinates systems where each point can be uniquely defined by a set of coordinates.

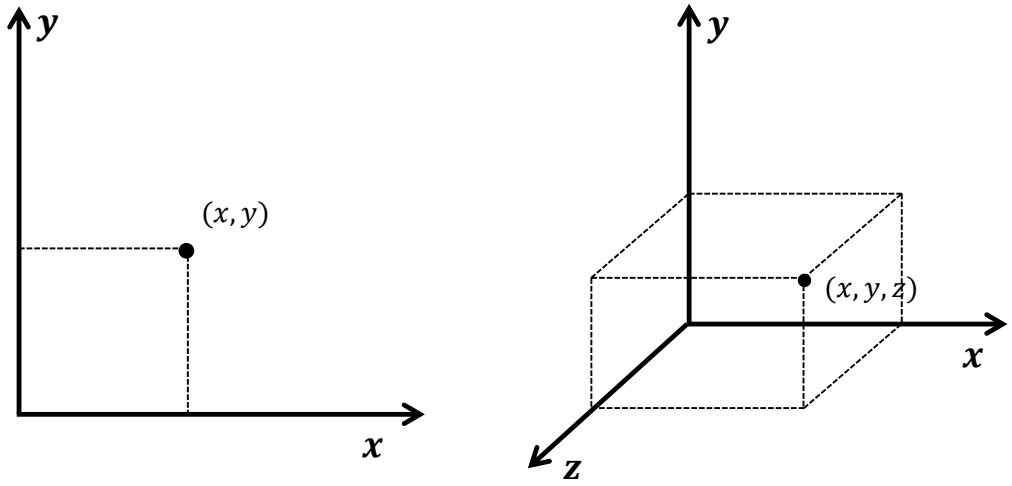


Figure 3.3 A two-dimensional (left) and a three-dimensional (right) Cartesian coordinate system.

As described in the previous section, virtual models are described using points (*vertices*) that form polygons. Hence, in order to design a virtual model and consequently a virtual world, a coordinate system that will allow the definition of vertex locations with respect to a known reference frame is essential. Although an infinite number of coordinate systems exist and any type of coordinate representation can be applicable for drawing virtual scenes, the *Cartesian* is the most frequently used coordinate system in computer graphics[101].The *Cartesian* coordinate system consists of

two or three axes perpendicular to each other, depending on whether it is used to describe the $\mathbb{R}^2$ or the $\mathbb{R}^3$ space. These axes are called the $x-, y$- and z-axis. In the two-dimensional space, a point $\mathbf{p}$ is specified by a set of x and y coordinates, where the usual convention is that the x-axis is horizontal pointing to the right while the y-axis is vertical pointing upwards, as Fig. 3.3 depicts.

In the three-dimensional space, a point $\mathbf{p}$ is defined by a set of x, y and z coordinates where the z-axis can point away from the viewer or towards the viewer as Fig. 3.3 shows. Describing the three-dimensional coordinate space, it is important to mention a property of the *Cartesian* coordinate system called *handedness* [143, 144]. This property is defined by the rule of thumb: which hand, left or right, can align with the coordinate system by pointing the thumb towards the direction of the x-axis, the forefinger towards the y −axis and the middle finger towards the z-axis.

The example of Fig. 3.3 is a right-handed coordinate system. Although handedness is a matter of convention, some basic functions such as the calculation of the cross product between two vectors, depend on the handedness of the coordinate system. Thereby, mostly for compatibility reasons, the majority of the computer graphics algorithms and software libraries use right-handed *Cartesian* coordinates[140].

The first and obvious reason for the popularity of the *Cartesian coordinate system* is that it is easy for humans to understand and visualize. In essence, most people use a representation similar to the Cartesian representation, when dealing with issues of their everyday life as for instance when describing the location of a house or shop within a city grid. In mathematics, the Cartesian system is mostly used because of the simplicity that it provides in fundamental mathematical operations, as for example the calculation of distances between two points or the implementation of geometric transformations.

3.2.2 Algebraic Transformations

As the word implies, transformation is an action that transforms something into something else. In mathematics, transformation is a function that transforms $\mathbf{p}$ into another, unique $\mathrm{A}(\mathbf{p})$, where $\mathbf{p}$ can be a number or a vector of numbers. In the same context, geometric transformations of coordinates are mathematical operations that map the coordinates of a point to another point in space; coordinate transformation of a point in $\mathbb{R}^2$ produces a unique point also in $\mathbb{R}^2$, and transformation of a point in $\mathbb{R}^3$ produces a unique point in $\mathbb{R}^3$. These mathematical operations are divided in two categories, *linear* and *affine transformations*[144].

3.2.2.1 Linear Transformations

A transformation is *linear* if it satisfies the following conditions:

- For every $a \in \mathbb{R}$ and $\mathbf{u} \in \mathbb{R}^n$: $A(a \cdot \mathbf{u}) = a \cdot A(\mathbf{u})$

- For every $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$: $\mathrm{A}(\mathbf{u} + \mathbf{v}) = \mathrm{A}(\mathbf{u}) + \mathrm{A}(\mathbf{v})$

In the 2D or 3D space of Cartesian algebra, *linear* transformations take the form

$$\mathbf{v} = A(\mathbf{u}) \tag{3.1}$$

where the input $\mathbf{u}$ and output $\mathbf{v}$ are vectors corresponding to locations in space. The transformation function $\mathrm{A}()$ can be either a real number or a matrix of real numbers. Expanding Eq. 3.1, a linear transformation of a point in $\mathbf{u} = (u_x, u_y, u_z)$ the $\mathbb{R}^3$ space using a $3x3$ matrix $\mathbf{R}$ produces a point $\mathbf{v} = (v_x, v_y, v_z)$.

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \times \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \tag{3.2}$$

The most common linear transformations are *rotation* and *scaling*. Later sections will demonstrate how a transformation in the form of Eq. 3. 2 can be performed to achieve the rotation of a point around an axis in the 3D space.

3.2.2.2 Affine transformations

There exists a transformation that does not satisfy the two aforementioned conditions in order to be a *linear* transformation. It is called *translation* (commonly denoted by T) and it satisfies the following condition:

- For every $\mathbf{u} \in \mathbb{R}^n$ and a fixed $\mathbf{v} \in \mathbb{R}^n$: $\mathrm{T}(\mathbf{u}) = \mathbf{u} + \mathbf{v}$

The combination of a *linear* transformation and a *translation* produces another type of transformations, called *affine*. An affine transformation is

$$\mathrm{A}(\mathbf{u}) = \mathrm{B}(\mathbf{u}) + \mathbf{v} \tag{3.3}$$

where B is a linear *transformation*. An *affine* transformation in the $\mathbb{R}^3$ space is provided in Eq. 3. 4, where combining the *linear* transformation of Eq. 3.2 with a *translation* by a vector $\mathbf{t} = [t_x, t_y, t_z]$, transforms the point $\mathbf{u} = (u_x, u_y, u_z)$ into another point $\mathbf{v} = (v_x, v_y, v_z)$.

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \quad (3.4)$$

As it will be described in the following sections, the combination of transformations shown in Eq. 3. 4 would describe the rotation of point **u** around an axis, followed by a translation by a vector **t,** resulting in a new location **v** in the 3D space.

3.2.3 Transformation of coordinates

Both *linear* and *affine* transformations are abstractly described as mathematical functions that perform mapping of locations in space to new locations in the same space. According to what was discussed in the introduction of this Chapter, the shape of a virtual 3D model is described as a collection of points in the $\mathbb{R}^3$ space. Consequently, the same mathematical operations that are used the transformation of a single point in space, can be used to transform the collection of points that a virtual model consists of.

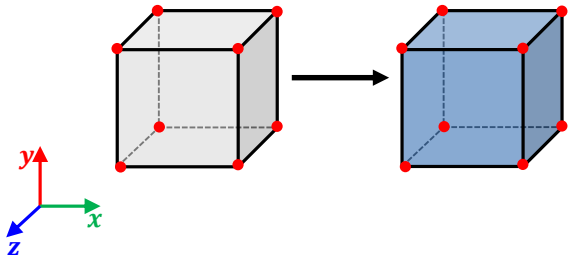
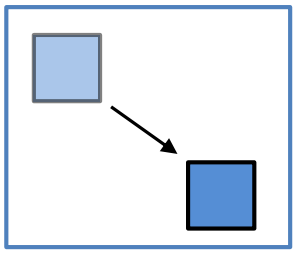


Figure 3.4 Transformation of a virtual object.

Figure 3.4 demonstrates a 3D cube consisting of eight vertices. In this simplified visual illustration, the model is moved into a new location by equally moving each one of the cube's vertices across the same direction with respect to the coordinate space. This example demonstrates the reason why coordinate transformations are an essential tool in computer graphics applications. Transformations are the mathematical tools for the design and spatial manipulation of virtual models, allowing actions such as *translation*, *rotation*, *scaling* or even *animation* (simulating movement of virtual objects). The fundamental geometric transformations used in the field of computer graphics, are *translation*, *scaling* and *rotation*.

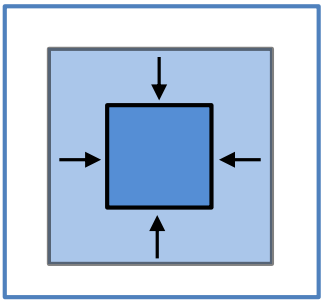
3.2.3.1 Translation



Translation is a non-linear transformation, used for moving a virtual model from one position to another as Fig. 3.4 demonstrated. It has already been mentioned that the shape of any 3D object is fundamentally a list of vertices corresponding to points in space. Consequently, translating a virtual object requires the translation of each of the object vertices by the same translation factor. Regarding a three-dimensional coordinate system, the equation that performs a translation is shown in Eq. 3. 5 were a translation factor of (t_x, t_y, t_z) with respect to each of the Cartesian axes is added to the (x, y, z) coordinates of a vertex to translate it to another location in space.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{3.5}$$

3.2.3.2 Scaling

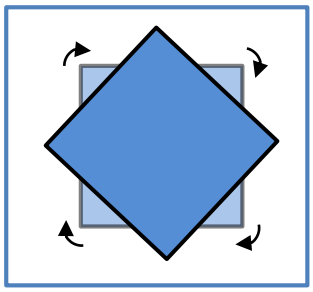


Scaling is a linear transformation that changes the size or the proportions of a virtual model. To scale a 3D model, every coordinate component of its vertices must be multiplied by a given scale factor. For instance, scaling the x coordinate by a factor s_x is achieved by the operation $x' = s_x \cdot x$. The scale factor in each direction of the coordinate space is independent of the other two directions. In a three-dimensional coordinate space of computer graphics, scaling is performed with the multiplication of the vectors corresponding to the coordinates of a model's vertices, by a *3x3* scale matrix, as Eq. 3.6 shows.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.6}$$

where s_x, s_y and s_z are the scale factors for the coordinates x, y and z respectively. Eq. 3. 6 allows scaling to be direction-specific (the scale factor can be different for each direction). Consequently *uniform scaling*, the modification of an objects size (growing or shrinking) without changing its proportions, is achieved using equal scale factors in each of the three coordinate axes.

3.2.3.3 Rotation



Abstractly described, rotation is the process of "turning" something around something else. In Cartesian algebra, rotation is a transformation that "turns" a point or a vector around another point (in $\mathbb{R}^2$) or around an axis (in $\mathbb{R}^3$), by an amount of rotation, called angle of rotation. Euler's rotation theorem, named after the mathematician Leonard Euler, states that in the three-dimensional space, every displacement of a rigid body such as a point of the rigid body remains fixed, is equivalent to a single rotation about an arbitrary axis that passes through the origin of the coordinate system[142]. Rotation is linear transformation, performed using a general $3x3$ matrix as Eq. 3.7 depicts. The multiplication of any 3D vector by this matrix ($\mathbf{R}$), rotates this vector by a given angle around a specific axis by a given angle.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \mathbf{R} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.7}$$

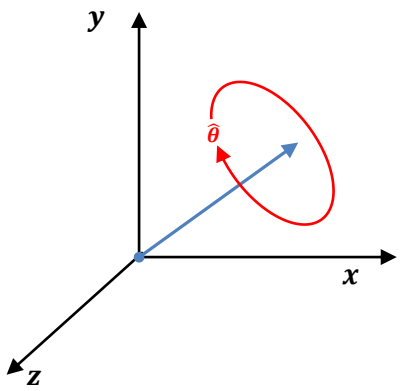


Figure 3.5 Rotation around an arbitrary axis that passes through the Cartesian origin.

In the three-dimensional computer graphics applications, similarly to *translation* and *scaling*, rotating a virtual model with respect to a given reference frame, requires the rotation of each of the model vertices by the same angle of rotation. The axis of rotation does not necessarily have to be one of the cardinal axes, but any axis of known orientation that passes through the origin of the *Cartesian* coordinate system.

A rotation about an arbitrary axis is achieved using the following rotation matrix:

$$\mathbf{R}(n,\theta)=\begin{bmatrix} n_x^{\,2}(1-\cos\theta)+\cos\theta & n_x n_y(1-\cos\theta)+n_z\sin\theta & n_x n_z(1-\cos\theta)-n_y\sin\theta \\ n_x n_y(1-\cos\theta)-n_z\sin\theta & n_y^{\,2}(1-\cos\theta)+\cos\theta & n_y n_z(1-\cos\theta)+n_x\sin\theta \\ n_x n_z(1-\cos\theta)+n_y\sin\theta & n_y n_z(1-\cos\theta)-n_x\sin\theta & n_z^{\,2}(1-\cos\theta)+\cos\theta \end{bmatrix} \quad (3.8)$$

In this equation, θ is the angle of rotation while $[n_x, n_y, n_z]$ is the unit vector that defines the direction of the rotation axis with respect to the *Cartesian* axes. Using the aforementioned rotation matrix, rotations around the principal axes can be deducted. It is important however to note that the *handedness* of the coordinate system affects the way rotations matrices are calculated. A positive rotation in a right-handed coordinate system is a counterclockwise rotation, while in a left-handed coordinate system is a clockwise rotation. For the right-handed system used throughout this dissertation, the rotation matrices for a rotation of ψ degrees around the x-axis, θ degrees around the y-axis and φ degrees around the z-axis are shown in Eq. 3. 9 – Eq. 3. 11 respectively.

$$\mathbf{R}_x(\psi)=\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix} \quad (3.9)$$

$$\mathbf{R}_y(\theta)=\begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \quad (3.10)$$

$$\mathbf{R}_z(\varphi)=\begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.11)$$

Borrowing the terminology from the field of aeronautics and aviation, the angles of rotation around the principal axes are called *roll*, *pitch* and *yaw*.

- *roll* is a counterclockwise rotation of φ around the z-axis
- *pitch* is a counterclockwise rotation of ψ around the x-axis
- *yaw* is a counterclockwise rotation of θ around the y-axis

Roll, pitch and yaw rotations can be multiplied to construct a single rotation matrix (Eq. 3. 7) that, as later sections will discuss, is sufficient in order to achieve any possible orientation of a rigid body in the 3D space.

An important property of rotations is that they are not commutative. The rotation of the order $\mathbf{R}_z(\varphi)\cdot\mathbf{R}_y(\theta)\cdot\mathbf{R}_x(\psi)$ does not produce the same rotation matrix as $\mathbf{R}_x(\psi)\cdot\mathbf{R}_y(\theta)\cdot\mathbf{R}_z(\varphi)$

despite having used the same rotation angles. An order of rotations commonly used in computer graphics is *roll-pitch-yaw*. In this order, a virtual model is first rotated around the z-axis, then around the x-axis and finally around the y-axis.

3.2.3.4 The gimbal lock issue

Although rotations matrices provide a compact way for performing rotations in the 3D space, they suffer from a potential drawback, caused by an effect called *gimbal lock*. In aeronautics, this term is used to describe situations where a gimbal or a gyroscope used to measure the orientation of an aircraft, loses one degree of freedom when rotated to its mechanical end[142]. In mathematics and computer graphics, the gimbal lock effect describes the loss of one degree of freedom that happens when applying rotations at the singularities ($0, \pi/2, \pi$) etc.

To mathematically explain *gimbal lock*, the following example can be given: A sequence of rotations by a round the x-axis, β around the y-axis and γ around the z-axis. Using the aforementioned rotation matrices for $\beta = \pi/2$, this sequence of rotations would be written as:

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos a & \sin a \\ 0 & -\sin a & \cos a \end{bmatrix} \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow$$

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & -1 \\ \sin\alpha & \cos a & 0 \\ \cos\alpha & -\sin\alpha & 0 \end{bmatrix} \begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow$$

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & -1 \\ \sin\alpha\cos\gamma - \cos\alpha\sin\gamma & \sin\alpha\sin\gamma + \cos\alpha\cos\gamma & 0 \\ \cos\alpha\cos\gamma + \sin a\sin\gamma & \cos\alpha\sin\gamma - \sin\alpha\cos\gamma & 0 \end{bmatrix} \Rightarrow$$

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & -1 \\ \sin(\alpha-\gamma) & \cos(\alpha-\gamma) & 0 \\ \cos(\alpha-\gamma) & -\sin(\alpha-\gamma) & 0 \end{bmatrix} \tag{3.12}$$

It can be noticed that the first row and last column of the resulting rotation matrix remains unaffected by the angles of rotation. Consequently, applying such a rotation to a virtual model, would not affect its x coordinates and thus, the model would be locked in rotating around the x-axis, regardless of the values provided for α and γ. This is an effect that should be taken in consideration in computer graphics applications, since it can result in virtual models (or more importantly, the virtual camera) getting locked with respect to a specific axis.

3.2.4 Representing Orientation

The word "orientation" is used to describe the direction of an object, with respect to another, known direction. In everyday life for example, people use the words like "north", "south", "west" or "east" to describe the direction of things with respect to the Earth. In a similar sense, orientation in mathematics describes the direction of a coordinate system with respect to another coordinate system, using three different types of representation: *Fixed* or *Euler* angles, *axis-angle* and *quaternions*.

3.2.4.1 Fixed and Euler Angles

According to Euler's rotation theorem, any orientation can be achieved as a composition of three rotations around the principal axes. Hence for minimal representation, any orientation can be denoted by a vector (α, β, γ) consisting of three angles, each corresponding to the amount of rotation around one of the principal axes. Under this representation however, the orientation of a coordinate system with respect to another coordinate system can be described as a composition of either *intrinsic* or *extrinsic* rotations[145].

Intrinsic rotations: The rotations are performed around the principal axes of the rotating coordinate system, which changes its orientation after each rotation. Consequently the direction of a principal axes depends upon preceding rotations. In this case, the aforementioned α, β and γ angles are called Euler angles. Although there are 12 possible different sequences of rotations that can be performed, a common convention regarding *Euler* angles, is the order Z-Y-X (with the first being the rotation around the z axis).

Extrinsic rotations: The rotations are performed around the principal axes of a fixed coordinate system. In this case, the aforementioned α, β and γ angles are called *fixed-angles*. A common convention regarding fixed-angles, is the order X-Y-Z (with the first being the rotation around the x axis).

The relationship between intrinsic and extrinsic rotations is that any sequence of extrinsic rotations equals to the opposite sequence of intrinsic rotations, and vice versa.

$$\underset{(\text{intrinsic})}{\mathbf{R}_z(\gamma)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha)} \quad \Leftrightarrow \quad \underset{(extrinsic)}{\mathbf{R}_x(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_z(\gamma)} \tag{3.13}$$

Extrinsic X-Y-Z rotation matrix for *fixed-angles* (φ, θ, ψ) :

$$\mathbf{R}(\varphi,\theta,\psi)=\mathbf{R}_z(\varphi)\mathbf{R}_y(\theta)\mathbf{R}_x(\psi)= \\ =\begin{bmatrix}\cos\varphi\cos\theta & \cos\varphi\sin\theta\sin\psi-\sin\varphi\cos\psi & \cos\varphi\sin\theta\cos\psi+\sin\varphi\sin\psi \\ \sin\varphi\sin\theta & \sin\varphi\sin\theta\sin\psi+\cos\varphi\cos\psi & \sin\varphi\sin\theta\sin\psi-\cos\varphi\sin\psi \\ -\sin\theta & \cos\theta\sin\psi & \cos\theta\cos\psi\end{bmatrix} \quad (3.14)$$

Intrinsic Z-Y-X rotation matrix for *Euler* angles (α,β,γ):

$$\mathbf{R}(\alpha,\beta,\gamma)=\mathbf{R}_x(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_z(\gamma)= \\ =\begin{bmatrix}\cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma-\sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma+\sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma+\cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma-\cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma\end{bmatrix} \quad (3.15)$$

Although matrix rotations using fixed or Euler angles are an important tool in computer graphics, they have two major drawbacks. The first is that they suffer from *gimbal lock* as described earlier, which can cause virtual models to "behave" unexpectedly. The second is the fact that matrix rotations do not offer the option of interpolating between two orientations. The latter is an equally significant drawback to *gimbal lock*, since angular interpolations allows smooth transitions in the movement of a virtual model from one orientation to another. This is of special importance in animation applications where the overall orientation transition of a virtual object is the result of stepwise increments/decrements of the individual rotation angles.

3.2.4.2 Axis-angle

An alternative way of describing the orientation of an object in the 3D space is the axis-angle representation. According to Euler's rotation theorem, any sequence of rotations in the three-dimensional Euclidean space is equivalent to a pure rotation around a single axis (called the Euler axis). The axis-angle representation parameterizes orientation using four parameters: the three parameters are the coordinate components of the unit vector providing the direction of the rotation axis, and the fourth parameter describes the angle of rotation. Using this type of representation, the desired orientation is applied on a virtual model using the rotation matrix of Eq. 3.8 that provides coordinate rotation about an arbitrary axis. The axis-angle representation does not suffer from the problem of *gimbal-lock*. However, the same as in matrix rotations, it does not allow interpolation between two orientations.

3.2.4.3 Quaternions

The third type of orientation representation used for computer graphics, involves the use of Quaternions, first discovered and described by Sir William Hamilton in 1844. Prior to discussing the use of quaternions in 3D rotations, a discussion about what quaternions are is essential. The

complete mathematical definition however requires extensive discussion; in short, a quaternion is a 4-tuple that defines an element in $\mathbb{R}^4$ as

$$q_n = w_n + x_n i + y_n j + z_n k \tag{3.16}$$

where w, x, y, z are real numbers or scalars. The product of two quaternions is given by:

$$\begin{aligned}\mathbf{q_0 q_1} = &(w_0 w_1 - x_0 x_1 - y_0 y_1 - z_0 z_1)w + \\ &(w_0 x_1 + x_0 w_1 + y_0 z_1 - z_0 y_1)\mathbf{i} + \\ &(w_0 y_1 + x_0 z_1 + y_0 w_1 - z_0 x_1)\mathbf{j} + \\ &(w_0 z_1 + x_0 y_1 + y_0 x_1 - z_0 w_1)\mathbf{k}\end{aligned} \tag{3.17}$$

while the conjugate of a quaternion is defined as

$$\mathbf{q}^* = w - x\mathbf{i} - y\mathbf{j} - z\mathbf{k} \tag{3.18}$$

3.2.4.4 Using unit quaternions for 3D rotation

Any vector $\vec{v} = (v_x, v_y, v_z)$ in the $\mathbb{R}^3$ space can be expressed as a quaternion of the following form:

$$p = w + v_x i + v_y j + v_z k \tag{3.19}$$

where w is equal to zero, while i, j and k are the unit vectors of the principal $\mathbb{R}^3$ axes. Additionally, a rotation around an arbitrary axis can be encoded in the following quaternion form:

$$q = cos(\theta/2) + (n_x i + n_y j + n_z k) \cdot sin(\theta/2) \tag{3.20}$$

where θ is the angle of rotation, and (n_x, n_y, n_z) is the vector that describes the axis of rotation, similar to the axis angle representation.

It has been proven that a rotation can be calculated as

$$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^* = w + v_x'\mathbf{i} + v_y'\mathbf{j} + v_z'\mathbf{k} \tag{3.21}$$

where $p'$is also a quaternion with $w = 0$. Using the aforementioned equations of quaternion multiplication and the definition of the conjugate quaternion, the elements of p' can be calculated, and converting these back to the $\mathbb{R}^3$ provides a vector $\vec{v}' = (v_x', v_y', v_z')$ which corresponds to the rotated coordinates of $\vec{v}$.

Although at first glance, using quaternions seems to be a complicated way of performing rotations, quaternion-based rotations offer some important advantages when it comes to computer graphics applications. One important advantage is that quaternion based rotations do not suffer from *gimbal lock*. The most important advantage though is that changing the orientation of a virtual

model from one orientation to another can be applied in a smooth way, since quaternion-based rotations allow interpolation between two given orientations[146].

3.2.5 Applying transformations about arbitrary points

In a computer graphics application, virtual models are not always centered with respect to the origin of the coordinate system. In such occasions, a direct implementation of the aforementioned transformation matrices for *scaling* and *rotation* produces undesired results. A visual example of this problem is provided in Fig. 3.6, where the location of a cube with respect to the origin is changed after the cube has been rotated.

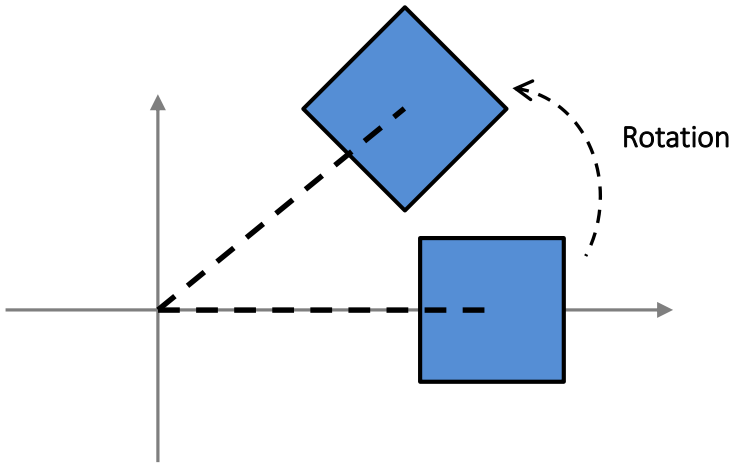


Figure 3.6 Rotation of a virtual object that is not centered at the Cartesian origin.

In order to avoid changing the cube's location, it must be first translated to the origin, then rotated, and then translated back to its initial position, as Fig. 3.7 shows. This sequence of operations would be mathematically expressed as:

$$\boldsymbol{v}' = \mathbf{R}\cdot(\mathbf{v}-\mathbf{t})+\mathbf{t} = \mathbf{R}\cdot\mathbf{v}+(-\mathbf{R}\cdot\mathbf{t}+\mathbf{t}) \tag{3.22}$$

where $\mathbf{v}$ is a vertex of the cube, and $\mathbf{t}$ is the vector corresponding to the initial location of the cube with respect to the Cartesian origin.

Regarding *scaling*, since the coordinate components of each of the vertices belonging to a 3D model are multiplied by a certain scale factor, scaling not only affects the shape of the model but also its location. Consequently, the same sequence of transformations must be applied. First, the model vertices need to be translated to the origin using a translation by $\mathbf{t}$, then scaled by matrix $\mathbf{S}$, and then translated back at the original location with a translation of $-\mathbf{t}$. This sequence of transformations is shown in Eq. 3. 23.

$$\mathbf{v}' = \mathbf{S} \cdot (\mathbf{v} - \mathbf{t}) + \mathbf{t} = \mathbf{S} \cdot \mathbf{v} + (-\mathbf{S} \cdot \mathbf{t} + \mathbf{t}) \tag{3.23}$$

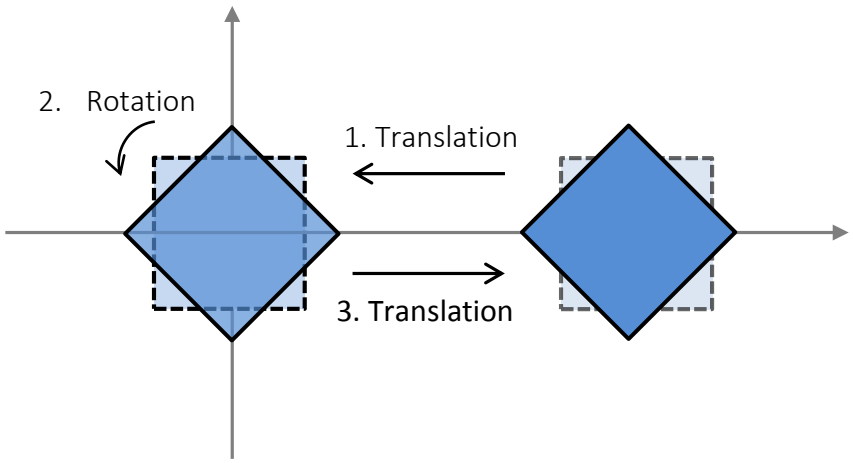


Figure 3.7 The sequence of transformations required to avoid object dislocations.

Both Eq. 3.22 and Eq. 3.23 perform a composite transformation where the translation component includes a rotation or a scaling transformation matrix. In general, the application of multiple linear and affine transformations results in a complex concatenation of calculations. This problem can be avoided via a different coordinate representation, called *homogeneous* coordinates, which are presented in the following section.

3.2.3 Homogeneous Coordinates

Up to this section, the *Cartesian* coordinate system and the functions of geometric transformations in the $\mathbb{R}^3$ space were presented. However Roberts et al. [147] proposed a different kind of coordinate representation in the field of computer graphics, called *homogeneous coordinates,* first introduced by the mathematician August Ferdinand Möbius in 1827. In *homogeneous coordinates,* also called projective coordinates, an extra dimension is added at each point $\mathbf{p}(\mathrm{x}, \mathrm{y}, \mathrm{z})$ in the $\mathbb{R}^3$ space, converting it to a four-element column vector in in the $\mathbb{R}^4$space with the following conversion.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix} \tag{3.24}$$

Cartesian *Homogeneous*

where $= 1$, and vice versa

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow \begin{bmatrix} x/w \\ y/w \\ z/w \end{bmatrix} \tag{3.25}$$

Homogeneous *Cartesian*

where $w \neq 0$.

As following sections of this chapter will demonstrate, homogeneous coordinates have a natural use in computer graphics because they provide a more practical way for calculating 3D to 2D projections compared to the standard *Cartesian* coordinates. However, additional advantages arise from the way geometric transformations are calculated using the homogeneous coordinate representation. Although the *Cartesian* system provides a solid framework for any potential calculation that a computer graphics application might require, Cartesian coordinates have a significant drawback regarding their direct use in a computer graphics application; applying multiple transformations at once such as translation, rotation and scaling of objects results in an awkward combination of mathematical operations, as discussed in section 3.2.5. For instance, introducing a 3D model in a virtual reality world requires the following sequence of operations:

1st: Scale the model to the desired size or proportions

2nd: Rotate the model to the desired orientation

3rd: Translate the model to the desired position

According to the equations presented in the previous sections, the function that would apply this sequence of transformations for a single vertex $v(x, y, z)$ belonging to a virtual model would be:

$$\boldsymbol{v}' = \mathbf{R} \cdot \mathbf{S} \cdot \boldsymbol{v}(x, y, z) + \mathbf{T} \tag{3.26}$$

In *homogeneous* coordinates, the aforementioned linear transformation matrices (*rotation* and *scaling*) can be extended to work with homogeneous coordinates, as Eq. 3.27 and Eq. 3.28 depict.

$$\mathbf{S} = \begin{bmatrix} s_X & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.27}$$

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.28}$$

In addition, the introduction of an extra dimension allows *translation* to be also implemented as a $4\text{x}4$ matrix multiplication, using the following matrix.

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.29}$$

Since these three fundamental transformations are all performed via matrix multiplication, every combination of the linear and affine transformations presented earlier, can be achieved with multiplication of the four-element column vector corresponding to a point in the three-dimensional space with a $4x4$ transformation matrix. This is a great advantage of *homogeneous* against *Cartesian* representation, since it results in the unification of the mathematical operations taking place during the rendering process [148, 149].

For instance, using Eq. 3.27 - 29 for *homogeneous* coordinates, the concatenated transformations of Eq. 3.17 would instead be performed as:

$$\boldsymbol{v}' = \mathbf{T} \cdot \mathbf{R} \cdot \mathbf{S} \cdot \boldsymbol{v}(x, y, z, 1) \tag{3.30}$$

In this equation, the typical order of operations would require multiplying the coordinates of the vertex v by $\mathbf{S}$, then the product of this operation by $\mathbf{R}$ and finally by $\mathbf{T}$. In linear algebra though $\mathbf{T} \cdot \mathbf{R} \cdot \mathbf{S} \cdot v = \mathbf{M} \cdot v$ where $\mathbf{M} = \mathbf{T} \cdot \mathbf{R} \cdot \mathbf{S}$ and hence, an alternative way of deriving with $\mathbf{v}'$ would be to first find $\mathbf{M}$ and then multiply with the vector that corresponds to the coordinates of $\mathbf{v}$.

Mathematically wise, both Eq. 3.26 and Eq. 3.30 require the same number of total calculations in order to derive with $\mathbf{v}'$ for a single vertex $\mathbf{v}$. In computer graphics however where virtual models consist of thousands or even millions of vertices, the pre-computation of a single $4x4$ transformation matrix that includes a combination of multiple transformations, significantly simplifies the way calculations are performed in a computer processor, improving the efficiency of a computer graphics application in terms of rendering speed [150, 151].

3.2.3.1 Pose of a rigid body

The term *pose* or *3D pose* will be mentioned in various sections of this dissertation. It refers to the position and orientation of rigid body or a coordinate system relative to another coordinate

system. The *pose matrix* is a $4x4$ *homogeneous* transformation matrix, consisting of a $3x3$ rotation part and a translation vector, as depicted in Eq. 3.31.

$$\mathbf{M} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.31)$$

The *pose matrix* an essential tool in computer graphics, since it allows defining relative positions between the several coordinate frames involved in a virtual reality environment. The pose matrix is used to perform transformations that change the position and orientation of objects without affecting the object's shape (*rigid body transformations*). An important property of a pose matrix (as any homogeneous transformation matrix) is the fact that it is invertible. The inverse of a pose matrix is also a pose matrix, as the inverse of any homogeneous transformation matrix is also a homogeneous transformation matrix. This is another great advantage of the homogeneous representation of coordinates, since this property allows easily performing inverse transformations, and also calculating inverse relationship between coordinate spaces. For example, if $M_{i\to j}$is the transformation matrix that provides the pose (position and orientation) of a coordinate system i with respect to a coordinate system j, then $M_{j\to i} = {M_{j\to i}}^{-1}$ provides the pose of j with respect to i.

3.3 Coordinate Spaces in Computer Graphics

Although a single coordinate system can be infinitely extended to include a virtual environment of any size, in a computer graphics applications multiple coordinate frames are employed. The reason is that each of these frames is essential in order to describe different pieces of information. For instance, in a very simple game where the player drives a car in a circular racing track using the arrow buttons of a keyboard. The first piece of information that the game designer needs to define, is the shape of the virtual car model, including components such as wheels, bumpers, doors and any other element needed in order to achieve a realistic virtual model of a real-world car. Since this model will consist of vertices, as already described in the previous Section, a coordinate system needs to be defined for the modeling process. In computer graphics, this coordinate system is called the *model space*.

Since however the game would allow the player to drive the car across a racing track, the relative position between the car and the track also needs to be defined. For this purpose, a second coordinate system is employed. In computer graphics, this coordinate system is called the *world space*.

Finally, a racing game would require the player to predict and perform actions such as accelerating, breaking and turning through keyboard interactions. Consequently the virtual scene should be projected on the 2D monitor from a certain point of view that would allow the player to see the car and the racing track from a certain perspective. This would require the introduction of an additional coordinate system that would define the position of the camera with respect to the virtual scene. In computer graphics, this coordinate system is called the *camera space*.

3.3.1 Object Space

The *object space* is the local reference frame of a virtual object. It is an independent coordinate system that moves along with a virtual model, following any rigid or non-rigid transformation applied to this model. The *object space* is also called *model space* because it is the coordinate system used to design and/or describe the geometry of a virtual model. Usually but not restrictively, the object space is considered at the geometric center of a virtual model.

Figure 3.8 illustrates probably the simplest example of a virtual model, a 3D cube. In this figure, the Cartesian axes of the cube's object space can be seen at the center of the cube. Assuming that the length of the cube sides equals to 2 units, the corresponding local coordinates of the eight vertices forming the cube with respect to the *object space* would be in the range [-1, 1] in each direction, as Fig. 3.8b illustrates.

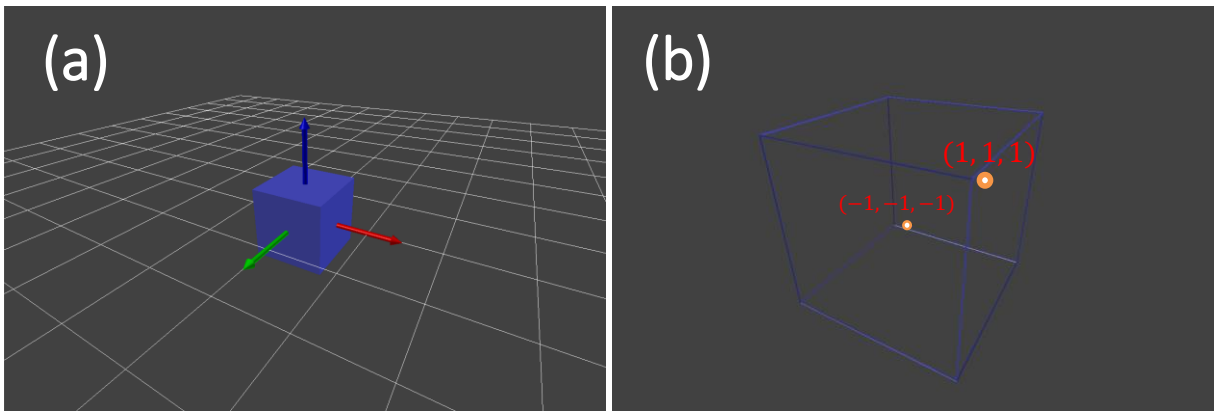


Figure 3.8 The object space of a virtual cube, assumed at the center of the cube's geometry. The vertex coordinates, expressed with respect to the object space.

3.3.2 World Space

In most of the cases, a virtual reality application requires rendering of multiple virtual models within the same scene. Since each of these objects is designed on the *object space*, rendering a scene of multiple objects based on their local vertex coordinates would result shapes overlapping[142]. Hence, the position and orientation (pose) of virtual objects needs to be defined

using another reference frame instead of the *object space*. In computer graphics, the *world space* or *global space* plays the role of the main reference frame in a virtual scene.

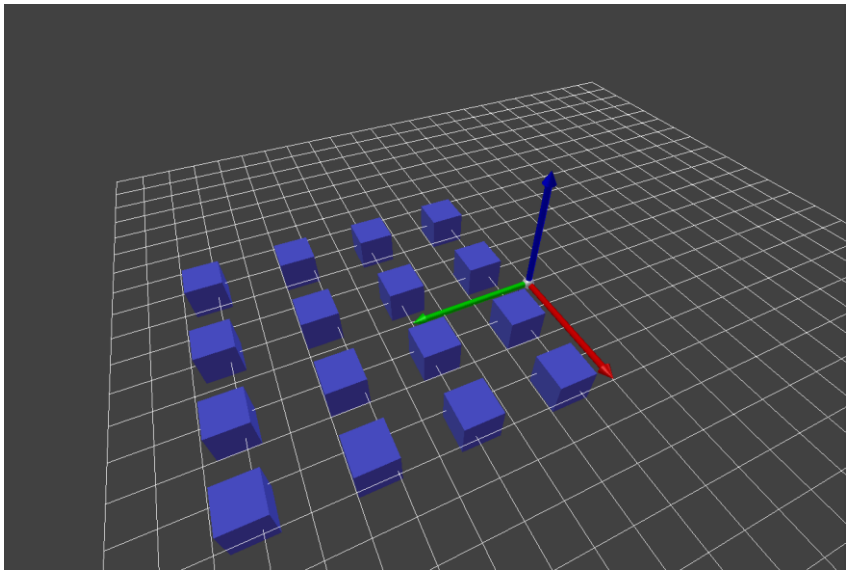


Figure 3.9 Introduction of multiple virtual objects within the world space of a VR scene.

Figure 3.9 illustrates a several virtual cubes introduced into virtual world scene. It can be noticed that the main frame of reference in this scene is the *world space*, at the center of the figure. This figure demonstrates the importance of the *world space*. It allows the introduction of multiple objects within the same scene, the pose of each is defined with respect to a central reference frame. The *pose* of a virtual object with respect to the *world space* is called *model transformation*, and the matrix providing this transformation is called the *model matrix*.

3.3.3 Camera Space

Computer graphics applications implement the notion of a virtual camera, which corresponds to the viewer's eye in the virtual world. As later sections will describe, one important step in the process of rendering, is the projection of three-dimensional coordinates onto a 2D image plane. The equations of projection are fundamentally coordinate transformations that depend on several parameters regarding the characteristics of a camera. However, the first piece of information that must be defined is the position of the viewer and the direction that the viewer looks at. In mathematical terms, a virtual camera is in essence a reference frame corresponding to the viewer's eye. This reference frame is called the *camera space*. The camera space is a coordinate system used to encapsulate information regarding the position and direction of the viewer with respect to the virtual world.

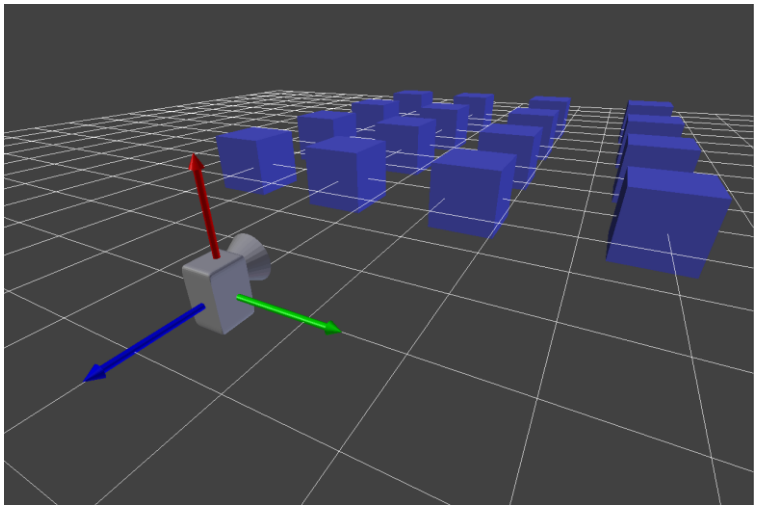


Figure 3.10 A virtual camera, added at the VR scene of Fig. 3.9. The camera space is at the center of the camera, with the z-axis pointing away (right-handed camera space).

As any three-dimensional reference frame, the *camera space* is also a Cartesian coordinate system formed by three principal axes. The x and y axes of the *camera space* are aligned with the horizontal and vertical dimensions of the view plane, while the z-axis points towards the screen or away from the screen, depending on the *handedness* of the *camera space*. In a right-handed coordinate system, the camera looks towards the negative direction of the z-axis, while in a left-handed system, the camera looks towards the positive direction of z-axis.

3.4 The rendering pipeline

The process of creating a pixel image of a 3D virtual world is called *rendering*. Previous sections explained the building blocks of a virtual world, and provided the fundamental mathematical background for shaping and transforming 3D objects within a virtual world. The ultimate goal of a computer graphics application however, is the production of images of the 3D virtual world, and draw these images on a two-dimensional device monitor. The term *rendering pipeline* or *graphics pipeline* refers to the complete sequence of processing stages that a rendering engine performs in order to produce two-dimensional visualizations of virtual worlds. Although the specifics of each rendering step might vary depending on the software and hardware tools responsible for performing the actual rendering, a general description of the rendering process is shown in Fig. 3.11.

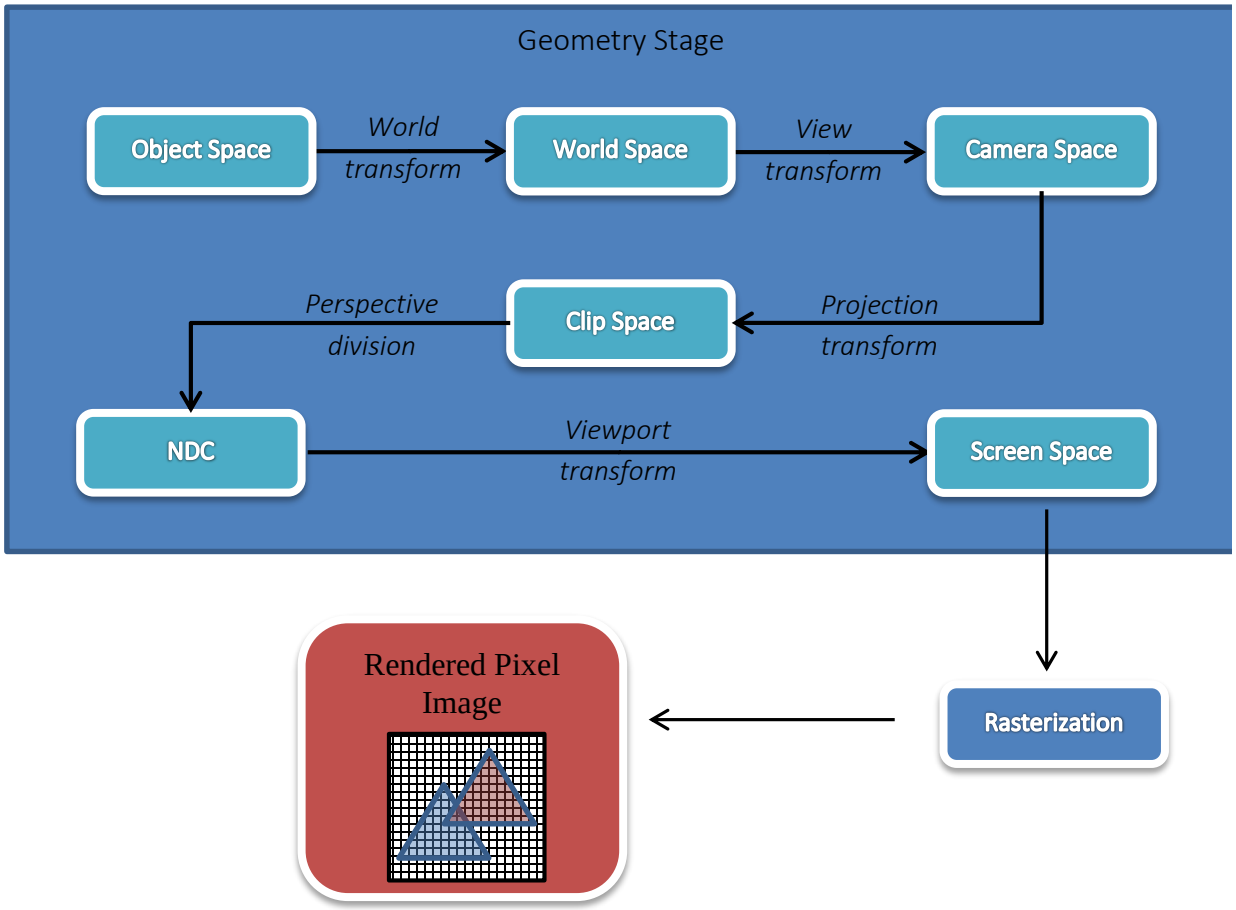


Figure 3.11 The rendering pipeline.

In terms of mathematics, the *rendering pipeline* is the calculation of a sequence of concatenated transformations that performs a mapping of vertex coordinates expressed with respect to the object space, into pixels expressed with respect to the coordinates of the device screen. The rendering pipeline is divided into two main stages. The first stage operates on the vertices of the geometric primitives that virtual objects consist of. The second stage performs calculations on the projected coordinates of these vertices, utilizing information regarding the polygons each vertex belongs to, as well as color information regarding vertices.

3.4.1 Step 1: World Transform

The first step in the rendering process is the introduction of virtual models within the virtual world at desired locations and orientations. Prior to this stage, a model is essentially a collection of floating point triplets, corresponding to model's vertex coordinates with respect to the object space. The purpose of this rendering step is to transform these vertices into global coordinates, using the mathematical tools described in previous sections.

In order to convert local into world coordinates, a homogeneous transformation matrix is constructed for each object based on its desired pose. This matrix is called the *model matrix*, commonly symbolized as $M_{model \to world}$ or M_{world}. Using the *model matrix*, each vertex $v(x_m, y_m, z_m)$ belonging to the *object space*, is transformed into a vertex $v'(x_w, y_w, z_w)$ in the *world space*.

3.4.2 Step 2: View Transform

As section 3.3.3 described, a virtual camera corresponds to the eye of the viewer within the virtual scene. Properly rendering the virtual world with respect to the viewer, requires the expression of world coordinates into camera coordinates. At the second step of the *rendering pipeline*, the world coordinates of step 1 are transformed into coordinates of the *camera space*. This process is called the *view transform*, and it applies a transformation from world to camera space in each vertex. This transformation is once again performed using an affine transformation matrix called the *view matrix*, commonly symbolized as $M_{world \to view}$ or M_{view}. Using the *view matrix*, each vertex $v(x_w, y_w, z_w)$ belonging to the *world space*, is transformed into a vertex $v'(x_e, y_e, z_e)$ in the *camera space*.

3.4.2.1 Construction of the View Matrix

Since the view matrix corresponds to the pose of the virtual world with respect to the virtual camera, it would be expected to build this matrix using information regarding the position and orientation of the *world space* with respect to the *camera space*. Instead, it is much more practical to construct the inverse transformation matrix, which provides the pose of the camera with respect to the virtual world ($M_{view \to world}$). This matrix, describing the location $EYE(e_x e_y, e_z)$ as well as the orientation of the camera with respect to the *world space*, is given by

$$\mathbf{M}_{view->world} = \begin{bmatrix} x_{side} & x_{up} & -x_{dir} & e_x \\ y_{side} & y_{up} & -y_{dir} & e_y \\ z_{side} & z_{up} & -z_{dir} & e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.32}$$

where, as illustrated in Fig. 3.12, $V_{dir}(x_{dir}, y_{dir}, z_{dir})$ is an orthonormal direction vector that coincides to the $-z$ axis of the camera coordinate system (*right-handed*), $V_{up}(x_{up}, y_{up}, z_{up})$ is an orthonormal direction vector that coincides to the y axis of the camera coordinate system and $V_{side}(x_{side}, y_{side}, z_{side})$ is an orthonormal direction vector that coincides to the x axis of the camera coordinate system.

Calculating these three orthonormal vectors requires knowledge of two parameters regarding the virtual camera. The first is the location of the world space that the camera is aiming at, and the second is the direction that will roughly be the upwards direction of the camera with respect to

the world space, shown as $\overrightarrow{UP}$ in Fig. 3.12. The direction vectors corresponding to the camera axes are:

$$\vec{V}_{dir} = \frac{\overrightarrow{EYE} - \overrightarrow{AT}}{\|\overrightarrow{EYE} - \overrightarrow{AT}\|} \tag{3.33}$$

$$\vec{V}_{side} = \frac{\overrightarrow{UP} \times \vec{V}_{dir}}{\|\overrightarrow{UP} \times \vec{V}_{dir}\|} \tag{3.34}$$

$$\vec{V}_{up} = \vec{V}_{dir} \times \vec{V}_{side} \tag{3.35}$$

Using the fundamental property of affine transformations, obtaining the transformation of world space coordinates into camera coordinates is just a matter of inverting $M_{view \to world}$:

$$M_{world \to view} = {M_{view \to world}}^{-1} \tag{3.36}$$

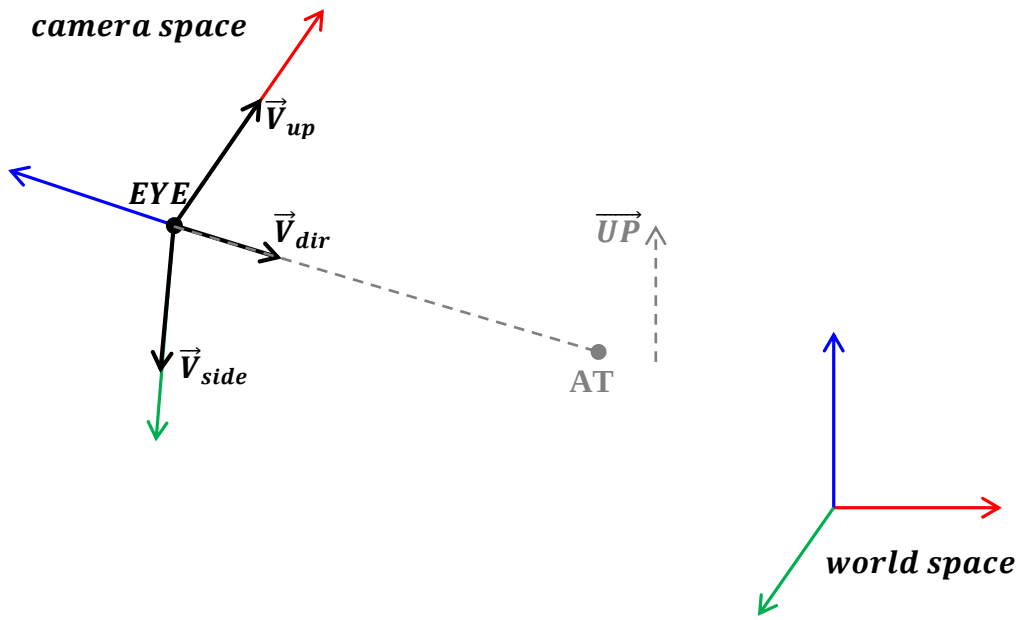


Figure 3.12 Construction of the camera view matrix.

3.4.3 Step 3: Projection Transform

The concatenated sequence of operations taking place in third step of the *rendering pipeline* is called *projection* or *viewing transformation*. *Projection* is a general term that describes any dimension-reduction process, and the field of mathematics that involves with geometric projections is called *projective geometry*. Etymologically, the word "projection" means estimation. Abstractly defined, a geometric projection is an estimation of how a geometric shape would

appear when viewed from a certain perspective, and while *Euclidean geometry* describes objects as they are, *projective geometry* describes objects as the "would appear". Despite the name however, this step of the rendering pipeline is not designed to actually produce 2D coordinates corresponding to the projections of 3D vertices. Instead, it prepares vertex information for projection in 2D, which happens on the next step of the pipeline called *perspective division*.

In addition projection preparation, in this rendering step the vertex coordinates of virtual objects are normalized within a box- shaped volume called the *clip space*. As the name implies, the *clip space* is used to clip vertices that lie outside the camera field of view. The procedure that determines which vertices should be rendered and which should be ignored is called *clipping* or *clip testing* and it is highly significant since it increases the rendering performance[152]. To optimize the rendering process, *projection* and *clipping* are combined into a single mathematical operation; however it is essential to explain the theory behind the two main projection types before discussing how these are actually implemented in a rendering engine.

3.4.3.1 Orthographic Projection

In computer graphics two types of projections are used, *orthographic* and *perspective*. Both involve a viewing volume that defines the area of a virtual scene visible from the position of the camera. In orthographic projection, also known as *parallel projection*, all points belonging to the 3D space are projected in a parallel way onto the projection plane. The viewing volume of orthographic projection is a cubic area formed by the far, near, left, right, top and bottom *clipping planes (*Fig. 3. 3.13*)*. In orthographic projection, the size of objects is not affected by their distance to the projection plane.

As previous sections showed, computer graphics applications define a coordinate space for the camera, and projections of the virtual world are performed with respect to this coordinate space. Most commonly, the camera space is defined such as the projection plane is parallel to the $x - y$ plane, while the z-axis is either directed towards or away from the projection plane. In such a setup, where the projection plane is parallel to a cardinal plane as in the example of Fig. 3.13 (either $x - y$, $x - z$ or $y - z$), the orthographic projection of an object is performed by scaling with a scale factor equal to zero for axis perpendicular to the projection plane. This is a transformation that creates 2D points from 3D points simply by discarding the third coordinate component [141, 143].

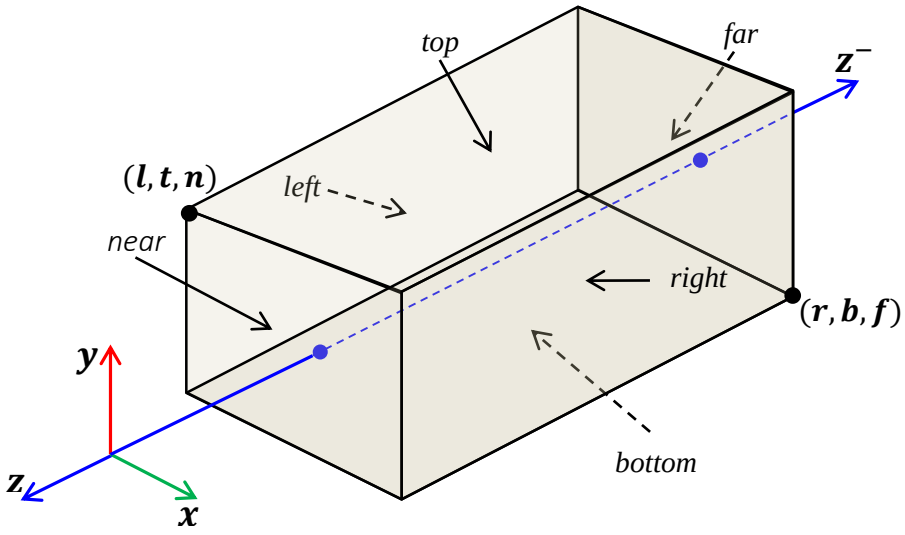


Figure 3.13 The viewing volume of orthographic projection.

Since projection is also a transformation of coordinates, it is important for a computer graphics application that this transformation is also provided in a $4x4$ matrix form, in order to maintain the unification of calculations at any stage of the rendering process. The projection matrix that can be applied for achieving this effect for a right-handed system is:

$$\mathbf{P}_{ortho} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.37}$$

In some cases however, virtual objects need to be projected onto arbitrary planes. The transformation matrix that provides a projection of coordinates onto a plane perpendicular to a vector $\hat{\boldsymbol{n}}$ is given by:

$$\mathbf{P}_{ortho}(\hat{n}) = \begin{bmatrix} 1-n_x^2 & -n_x n_y & -n_x n_z & 0 \\ -n_x n_y & 1-n_y^2 & -n_y n_z & 0 \\ -n_x n_z & -n_y n_z & 1-n_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.38}$$

The fact that objects maintain their size and proportions in orthographic projection, regardless of their distance to the viewer, produces unrealistic results, since it differs from the way people are used to see the world. However, an important property of this projection type is that parallel lines remain parallel. This is a property that allows viewers to accurately perform distance measurements in a projected image. For this reason, orthographic projection is a type of

projection used computer graphics applications involving with Computer Aided Design, Architecture and Engineering.

3.4.3.2 Perspective Projection

Perspective projection, a more complicated type of projection compared to *orthographic*, is most preferable in virtual reality applications since it mimics the way human eye perceives the world and hence produces more realistic results. Using perspective projection, the rendering outcome gives the viewer the sense of actually being within a virtual scene[140]. This type of projection implements a model called the *pinhole camera*[152]. The *pinhole camera* model describes the camera as a box with a small hole in the center of one of its sides. As rays of light enter the box through this hole, images are projected onto the opposite side of the box, showing objects upside down. The *pinhole camera* is in essence the way real-world cameras worked before the era of digital imaging[153]. Some important properties of perspective projection are:

1. Lines are projected into lines
2. Parallel lines do not remain parallel. Instead, they intersect at the vanishing points.
3. Ratio and hence proportions are not preserved, since the size of a projected object varies depending on its distance from the center of projection.

In perspective projection, the viewing volume, also called *frustum*, has the shape of a truncated pyramid, defined the same clipping planes as in orthographic projection. The size of the pyramid is a result of four parameters: the distance between near and far clipping planes, the *focal length* (f) which is the distance of the center of projection to the projection plane, as well as the horizontal (φ_{fov}) and vertical (θ_{fov}) angles of the field of view. The term *frustum width* denotes the distance between the *left* and *right* clip planes, the term *frustum height* denotes the distance between the *top* and *bottom* clip planes, and the term *aspect ratio* denotes the ratio between height and width. In perspective projection, as illustrated in Fig. 3. 3.14, the projection plane is considered parallel to the near plane, at an opposite direction with respect to the origin across the z-axis.

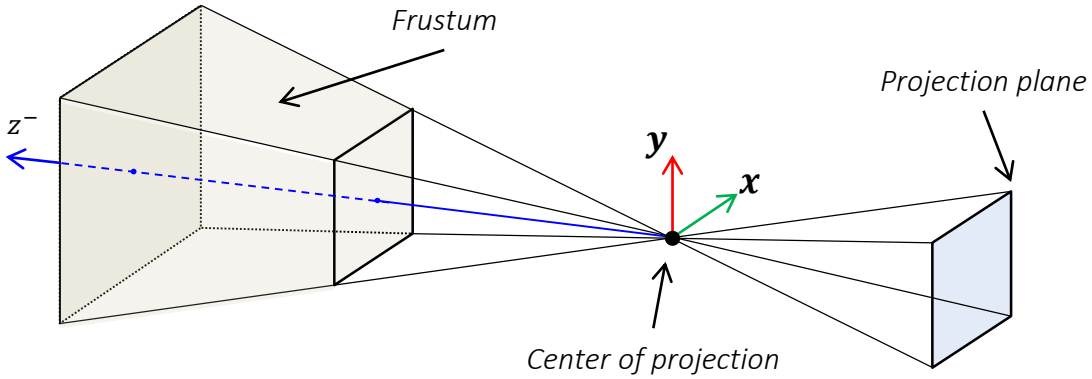


Figure 3.14 The frustum of perspective projection.

Figure 3.15 shows the top and side cross-sections of the perspective projection frustum. From these cross-sections, using the rule regarding the radio of similar triangles, it can be deducted that

$$\frac{x_p}{x_e} = \frac{-f}{z_e} \Rightarrow x_p = \frac{-fx_e}{z_e} \tag{3.39}$$

$$\frac{y_p}{y_e} = \frac{-f}{z_e} \Rightarrow y_p = \frac{-fy_e}{z_e} \tag{3.40}$$

and

$$z_p = -f \tag{3.41}$$

In order however to avoid the complexity of equations caused by the negative sign that x and y coordinates of projected points obtain, the sign can be neglected and the projection plane can be considered to be aligned with the near plane of the frustum [149]. Consequently, with respect to the *Cartesian* coordinate system of a camera the coordinates of the projection $\mathbf{p}'$ for a point $\mathbf{p}(x_e, y_e, z_e)$ are:

$$p' = \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = \begin{bmatrix} f \cdot x_e / z_e \\ f \cdot y_e / z_e \\ f \end{bmatrix} \tag{3.42}$$

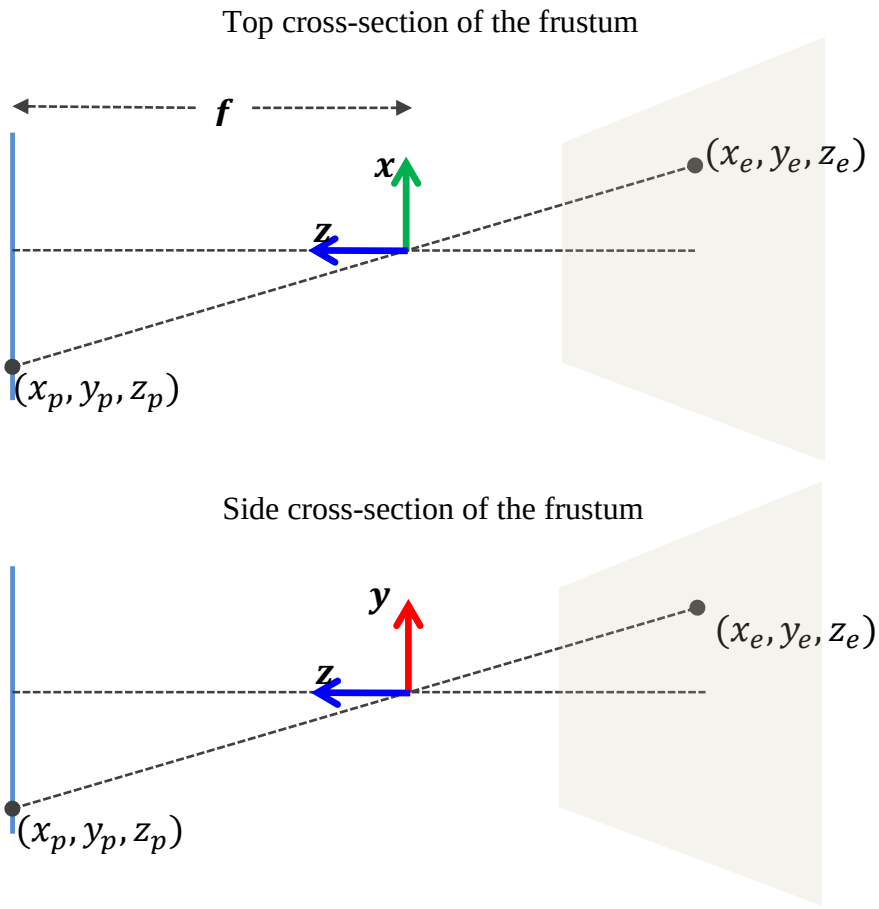


Figure 3.15 Cross-sections of the view volume, showing the projection of a point onto a plane at a distance f from the origin of the camera.

In section 3.2.3, the homogeneous coordinates where described as having natural use in projection calculations. The reason is that *homogeneous* coordinates provide a more elegant way of performing projections. According to Eq. 3.24 and 3.25, the conversion from *homogeneous* to *Cartesian* space is basically a division of coordinates by a factor w, since a homogeneous point (x, y, z, w) is the equivalent of a Cartesian point $(x/w, y/w, z/w)$. Up to this point though, conversions between Cartesian and homogeneous space where provided using a value of w equal to 1. In projective geometry however, w plays a critical role in building the equations of projection. For instance, using a value of w equal to z, the *Cartesian* coordinates of Eq. 3.42 could be rewritten in the equivalent *homogeneous* coordinates [144]:

$$\mathbf{p}' = \begin{bmatrix} fx & fy & fz & z \end{bmatrix} \tag{3.43}$$

This conversion would allow perspective projection to be achieved using a homogeneous transformation matrix in the form of Eq. 3.44, that converts a point $\mathbf{p} = [x, y, z, 1]^{\boldsymbol{T}}$ to the point $\mathbf{p}' = [fx, fy, fz, z]^{\boldsymbol{T}}$.

$$\mathbf{P}_{persp} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.44}$$

3.4.3.3 The Clip Space

When rendering a virtual scene, it is important to derive with a fast and efficient way of determining which points of the virtual scene should be rendered. Although *perspective* and *orthographic projections* are the basis for the creation of 2D images corresponding to certain views of the 3D virtual world, none of these methods takes into account the relationship between a vertex and the clip planes. On the contrary, a piece of information is actually lost, since after applying the aforementioned projection matrices the distance of a vertex with respect to the camera is discarded. Consequently, a direct implementation of these projection equations would not provide useful information for determining which vertices should be clipped.

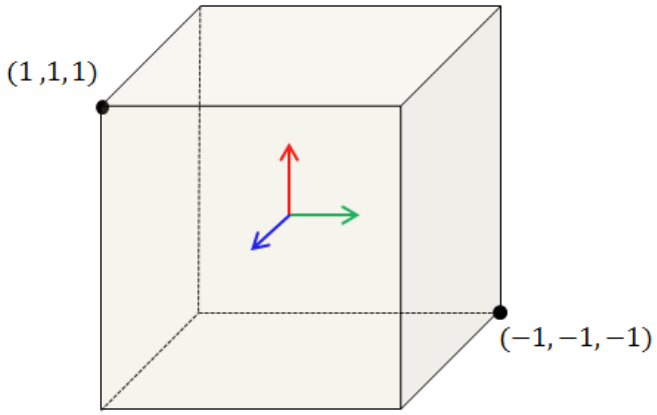


Figure 3.16 Canonical view volume, bounded within the [-1, 1] range across the principal axes.

In the rendering pipeline, the *projection* and *clipping* process are combined with the introduction of an additional coordinate space, the *clip space* or *canonical view volume*. The clip space is a cube ranging from -1 to 1 across each direction, centered on the origin of the camera space. The general idea behind the introduction of the clip space is that if a mapping function allowed to scale and translate the viewing volume so that it matches to the clip space, determining what parts of the

virtual scene lie within this space would be just a simple operation that would discard any vertex having coordinate components ranging outside [-1, 1].

Orthographic to Clip Space Conversion

As demonstrated in Fig. 3.13, the viewing volume of orthographic projection is already a box-shaped area consisting of six clipping planes with the following fixed coordinates:

Clipping plane	*Fixed coordinate*
left	$x = l$
right	$x = r$
top	$y = t$
bottom	$y = b$
near	$z = z_n = f$
far	$z = z_{nf}$

Hence, transforming the viewing volume of orthographic projection into the canonical view volume requires to transformations:

1st: A translation, so that the viewing volume becomes centered with respect to the camera space. This is achieved by subtracting the coordinates of the viewing volume center. Since is already centered with respect to the x and y axis of the camera space, the view volume center coordinates are:

$$(x, y, z) = \left(0,0,\frac{z_f + z_n}{2}\right) \tag{3.45}$$

2nd: A scaling across the three principal axes, that normalizes the viewing volume within the range [-1, 1]. Such scaling is achieved with the following scale factors:

$$Scale\ factor\ across\ x = \frac{2}{r - l} = 2/width$$

$$Scale\ factor\ across\ y = \frac{2}{t - b} = 2/height$$

$$Scale\ factor\ across\ z = \frac{2}{z_f - z_n}$$

Combining translation and scaling into a single homogeneous matrix provides the transformation that will convert the orthographic view volume into the canonical view volume:

$$\mathbf{M}_{ortho} = \begin{bmatrix} 2/width & 0 & 0 & 0 \\ 0 & 2/height & 0 & 0 \\ 0 & 0 & 2/(z_f - z_n) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -(z_f + z_n)/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow$$

$$\mathbf{M}_{ortho} = \begin{bmatrix} \frac{2}{width} & 0 & 0 & 0 \\ 0 & \frac{2}{height} & 0 & 0 \\ 0 & 0 & \frac{2}{z_f - z_n} & -\frac{z_f + z_n}{z_f - z_n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.46}$$

Using the resulting projection matrix, the clip space coordinates for orthographic projection are calculated as:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \mathbf{M}_{ortho} \cdot \begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ 1 \end{bmatrix} \tag{3.47}$$

Perspective Frustum to Clip Space Conversion

In perspective projection, the viewing volume or frustum is a truncated pyramid, so an additional step is required in order to transform this pyramid into the clip space, which is the conversion of the frustum into a box-shaped volume. Although this might seem as a complicated procedure, the required mathematical operations for this conversion have been provided in the previous section, as the equations of perspective projection. Figure 3.17 illustrates how mapping the x and y coordinate components of any point belonging to the frustum into their corresponding projected coordinates, is in essence the equivalent of a distortion that converts the frustum from a truncated pyramid to a box-shaped area. For instance, the point (x_1, y_1, z_1) is mapped to (x_p, y_p, z_p), hence closer to the centeriline of the frustum but at the same distance from the camera as the original point. Based on this observation, it can be deducted that the following mapping could be used to convert the perspective viewing volume into an orthographic viewing volume:

$$(x, y, z) \mapsto (f\frac{x}{z} \quad , \quad f\frac{y}{z} \quad , \quad z)$$

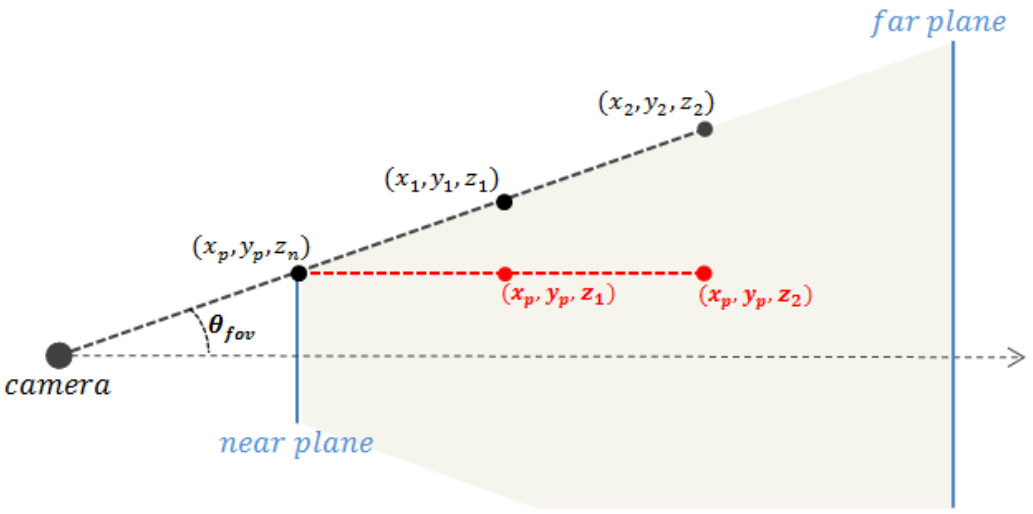


Figure 3.17 Cross-section of the perspective projection frustum, demonstrates the distortion of the frustum to a box-shaped volume.

However, leaving the z component unaffected by the mapping process produces a problem; straight lines are not mapped into straight lines [140, 142, 154]. To overcome this issue, a mapping called the pseudo-depth mapping is applied for z:

$$(x, y, z) \mapsto (f\frac{x}{z} \quad , \quad f\frac{y}{z} \quad , \quad \mathrm{A} + \mathrm{B}/z) \text{ , where } \mathrm{A} = z_n + z_f \text{ and } \mathrm{B} = -z_n z_f$$

The properties of pseudo-depth mapping are:

1. Lines are mapped to lines.

2. Relative depths are maintained.

3. Points on near and far planes are mapped accordingly.

The aforementioned mapping can be rewritten in homogeneous coordinates as:

$$(x, y, z, 1) \mapsto \left(f\frac{x}{z} \quad , \quad f\frac{y}{z} \quad , \quad \mathrm{A} + \frac{\mathrm{B}}{z}, 1\right)$$

Taking into account role of w in homogeneous coordinates, the above relationship can be equivalently written as:

$$(x, y, z, 1) \mapsto (fx\,, \quad fy\,, \quad \mathrm{A}z + \mathrm{B}\,, z)$$

This final form of the mapping operation can be also written as an affine transformation matrix, called the *perspective matrix*:

$$\mathbf{M}_p = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & z_n + z_f & -z_n z_f \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.48}$$

Applying this transformation matrix to any point belonging to the frustum, essentially converts the frustum to a box-shaped volume. This process is visually illustrated in Fig. 3.18. The conversion of frustum volume to the normalized clip space is achieved using the same transformation matrix that applies in orthographic projection. Concatenating these two transformations into a single matrix produces the final perspective projection matrix used in the *rendering pipeline*:

$$\mathbf{M}_{persp} = \mathbf{M}_{ortho} \cdot \mathbf{M}_p = \begin{bmatrix} \frac{2f}{width} & 0 & 0 & 0 \\ 0 & \frac{2f}{height} & 0 & 0 \\ 0 & 0 & \frac{z_f + z_n}{z_f - z_n} & -\frac{2z_n z_f}{z_f - z_n} \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.49}$$

After applying the perspective projection transformation, the resulting clip space coordinates are:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \mathbf{M}_{persp} \cdot \begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ 1 \end{bmatrix} \tag{3.50}$$

However, it can be noticed that due to the form of the final matrix, the homogeneous clip space coordinates are not normalized in the range [-1, 1] as in orthographic projection, but in the range [$-w_c$,w_c]. For this reason x_c, y_c and z_c are tested against w_c. If any coordinate is less than $-w_c$, or greater than w_c, then the corresponding vertex will be discarded.

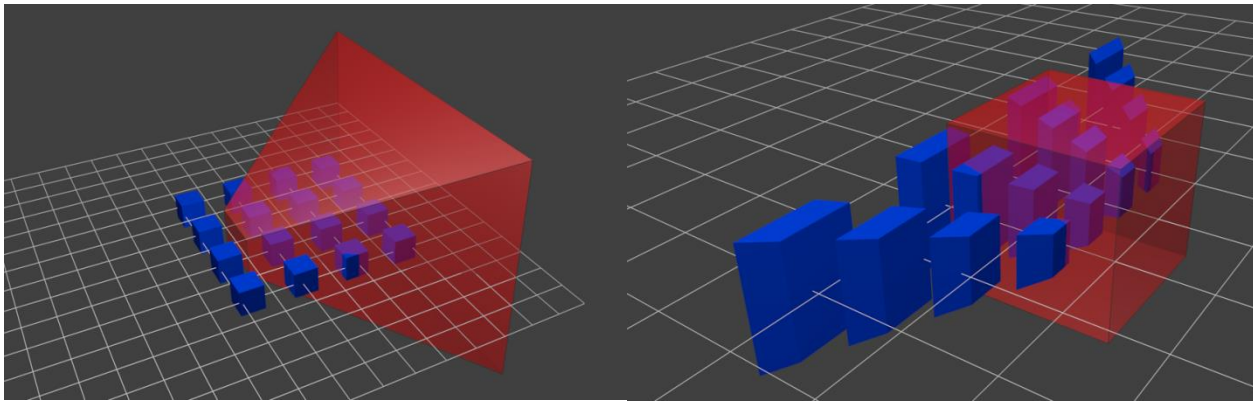


Figure 3.18 Distortion of the view frustum during the transformation of *view space* to *clip space*.

3.4.4 Step 4: Perspective Division

At this stage of the *rendering pipeline*, all vertices have passed through the *projection transform* step and have been normalized within the *clip space*. The output of the projection transform is a collection of non-clipped vertices, expressed in the homogeneous form $(x_c, y_c, z_c, w_c)^T$. Since the viewing volume was normalized in the range [-1, 1], the view plane has been also normalized into a square, bounded in the same range across the x and y dimensions. This plane is called the *normalized device coordinates frame* or *NDC space*. Hence, the remaining step in the projection process is the actual projection from three-dimensional clip space coordinates into two-dimensional coordinates of the *NDC space*. This is achieved with a division by w_c, called *perspective division,* which is directly performed by a simple conversion of homogeneous clip space coordinates into Cartesian coordinates:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} \rightarrow \begin{bmatrix} x_c / w_c \\ y_c / w_c \\ z_c / w_c \end{bmatrix} = \begin{bmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{bmatrix} \tag{3.51}$$

The result of this conversion is a set of (x_{ndc}, y_{ndc}) *normalized device coordinates (NDC)* that correspond to the projection of vertices in the NDC plane. While z_{ndc} could be discarded, since the NDC space is two-dimensional, its value is important because it maintains information regarding the pseudo-depth of each vertex.

3.4.5 Step 5: Viewport Transform

The fifth step of the rendering process is the mapping of NDC coordinates to screen space coordinates that correspond to actual pixels of the display device. Usually, a computer graphics application is not designed in a device-specific logic. Up to this stage of the rendering pipeline, no information regarding the display medium was taken into account. However, a computer graphics

application should be compatible with multiple display device resolutions. Additionally, most applications usually occupy smaller portions of the screen rather than the entire screen space. The portion of the screen that displays the rendering result is called *viewport*.

Figure 3.19(b) illustrates the parameters of the display device that are required in order to map coordinates of the NDC space (Fig. 3. 19a) in pixel coordinates of the display window. These parameters are:

- (vp_{xpos}, vp_{ypos}) : The screen coordinates of the top-left corner of the viewport in pixels
- vp_w and vp_h: The width and height of the viewport in pixels
- S_w and S_h: The width and height of the screen in pixels

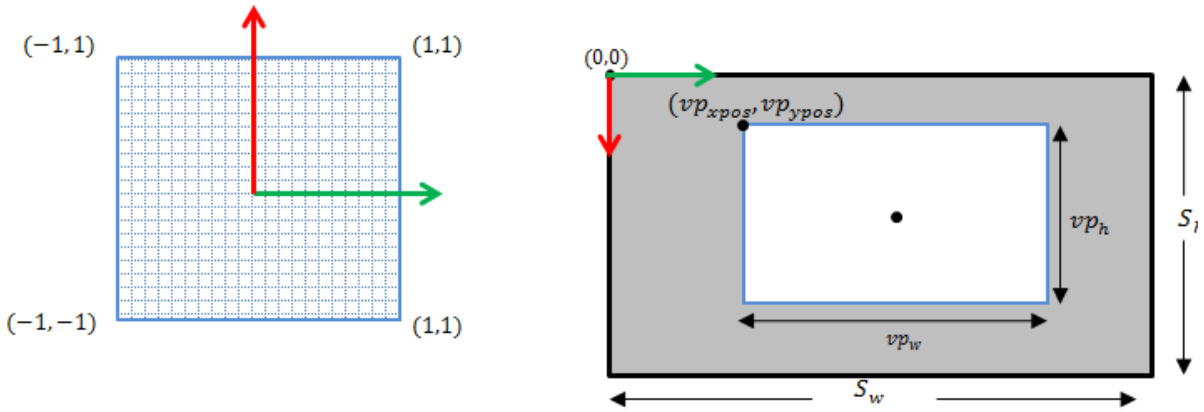


Figure 3.19 Conversion from the NDC space (left) to Viewport coordinates (right).

Using the aforementioned parameters, mapping of NDC to screen coordinates is achieved as a combination of scaling and translation:

$$x_{screen} = \frac{vp_w}{2} x_{ndc} + \frac{vp_w}{2} + vp_{xpos} \tag{3.52}$$

$$y_{screen} = -\frac{vp_h}{2} y_{ndc} + \frac{vp_h}{2} + vp_{ypos} \tag{3.53}$$

The minus sign in the second equation is due to the fact that, as illustrated in Fig. 3.19, the y-axis of the viewport has an inverse direction to the y-axis of the *NDC space*.

3.4.6 The Vertex Structure

The first stages of the *rendering pipeline*, discussed in previous sections, manipulate spatial information regarding virtual objects, and produce a projection of vertices onto the two-dimensional screen space. Other than its geometric shape however, describing a real-world object

requires the description of several other characteristics such as its color, texture, weight etc. In the same way, rendering a virtual model is not just a process of converting the model's vertex coordinates into pixels at a display monitor. Up to this, vertices where dealt as pure locations in the 3D object space, and the presented calculations only focused on the geometric transformation of these vertices from the 3D object space to 2D screen coordinates.

In computer graphics applications though, a *vertex* is a structure that contains several pieces of information, as the following table illustrates.

Vertex structure	
Position coordinates	$\mathbf{p}_v = (\mathrm{v}_x, \mathrm{v}_y, \mathrm{v}_z)$
Vertex normal	$\mathbf{n}_v = (\mathrm{v}_{nx}, \mathrm{v}_{ny}, \mathrm{v}_{nz})$
Color	$\mathbf{c}_v = (r, g, b)$
Texture coordinates	$\mathbf{t}_v = (u, v)$

Primarily, the *vertex structure* contains the coordinates of the vertex with respect to the reference frame of a virtual model. In addition, this structure contains information regarding its color, expressed by three values in RGB format or four values in RGBA format, depending on implementation. The color of each vertex is assigned at the content creation stage (3D modeling) of a virtual object and does not change on a frame-to-frame basis. Finally, the vertex structure contains a unit vector called vertex normal, and a set of 2D coordinates, called *texture coordinates* or *uv coordinates*.

3.4.6.1 Face and Vertex normal

The normal vector of a 3D triangle is a three-element unit vector that defines the direction that the triangle is facing at. The face normal of a triangle consisting of the points $\mathbf{p}_a$, $\mathbf{p}_b$ and $\mathbf{p}_c$, as the one shown in Fig. 3.20a, equals the cross product of the vectors $\mathbf{v}_{ab}$ and $\mathbf{v}_{ac}$.

$$\textbf{Face normal}: \mathbf{v}_n = \mathbf{v}_{ab} \times \mathbf{v}_{ac} \tag{3.54}$$

In a 3D model consisting of multiple triangles (faces), a single vertex can be used for the definition of several neighboring triangles. The vertex normal is calculated as the normalized average of the surface normals of the faces that contain that vertex.

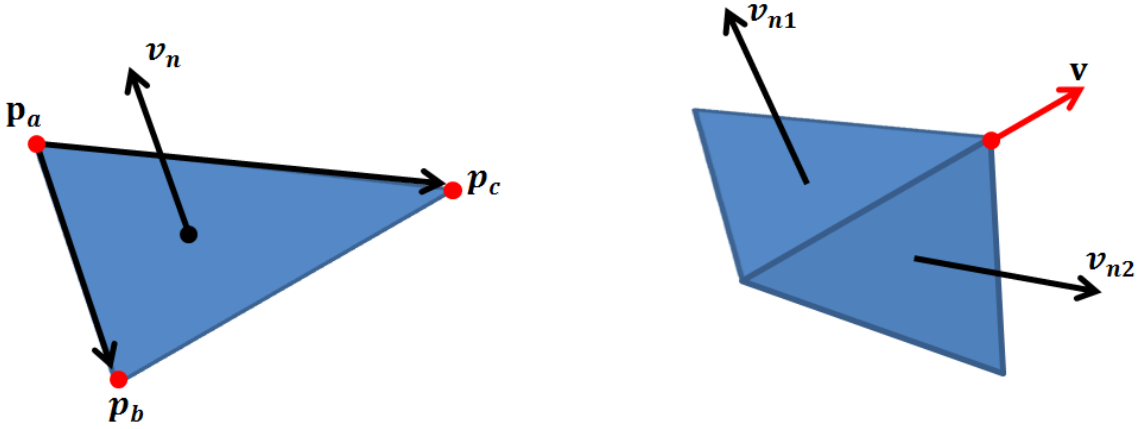


Figure 3.20 Illustration of face normal (left). The vertex normal (right) is calculated as the normalized average of neighboring polygons.

3.4.6.2 Texture coordinates

A virtual model, besides color, can be assigned with a texture. Textures are two-dimensional images that are projected on the outer surfaces of virtual models. To properly project a texture onto a 3D object, a correlation between the 2D coordinates of the texture image and the 3D vertex coordinates of the object is essential. This is achieved by a process called *unwrapping*, which unfolds the outer surfaces of the 3D model and scales them to match the dimensions of the texture image, as Fig. 3.21 illustrates. *Unwrapping* produces a unique correlation between the 3D model vertices and 2D points on the image space, normalized in the range [0, 1]. The (u, v) coordinates of these points are called *texture* or *uv coordinates.*

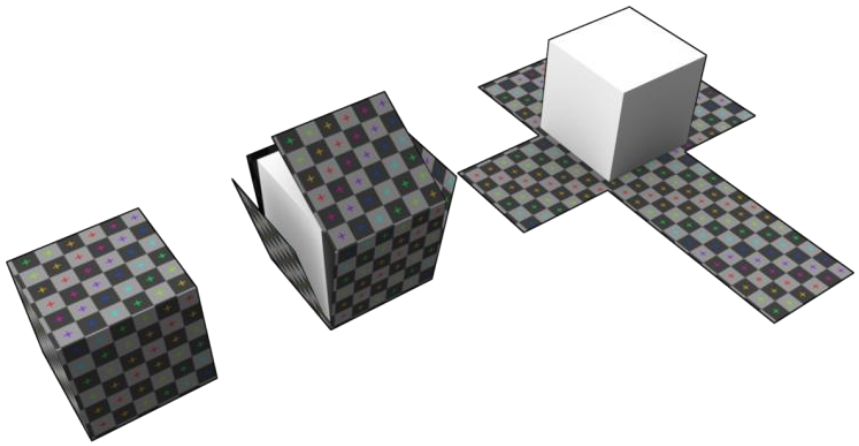


Figure 3.21 Surface unwrapping of a 3D cube provides the texture coordinates of each vertex.

3.4.7 Step 6: Rasterization

Having passed through the previous stages of the *rendering pipeline,* all the visible vertices are now transformed into 2D coordinates of the screen space. These vertices, according to what was discussed at the beginning of this chapter, where used for the definition of geometric primitives (e.g. triangles). Consequently, the output of previous rendering steps is fundamentally a list of visible primitives, projected onto the screen space. The remaining step in the production of the rendering outcome is the utilization of spatial and color information regarding these primitives, into a pixel image that will be drawn on the display device. This is performed during the final step of the rendering pipeline, called *rasterization*.

Rasterization is a procedure that determines the individual pixels covered by the projected primitives, and assigns color as well as depth values to these pixels. In modern computer graphics engines, the rasterization stage is implemented as a sequence of complicated hardware operations, performed solely in the GPU. The specific order of these operations differs depending on the architecture/manufacturer of the GPU and the algorithms utilized. Providing a detailed description of the rasterization mathematics, algorithms and techniques, is not within the scope of this dissertation. However, a general description of the main rasterization steps is essential for the understanding of Augmented Reality issues that will be presented in later chapters, such as *occlusion handling*.

Rasterization is divided into three main operations, *raster-scan*, *color interpolation* and *depth interpolation*.

1. *Raster-Scan*: This process determines which pixels are enclosed within a primitive. The output of the raster-scan operation is a list of pixels, called *fragments*, which will be assigned a color and a depth value in the next steps of the rasterization process (Fig. 3. 22a).

2. *Color interpolation*: The covered fragments, as provided by the raster-scan operation, are assigned a color value. This value is calculated by an interpolation of the vertices color values (Fig. 3. 22b).

3. *Depth interpolation*: The fragments provided in the raster-scan operation, are assigned with a depth value, calculated as an interpolation of the vertices color values (Fig. 3. 22c).

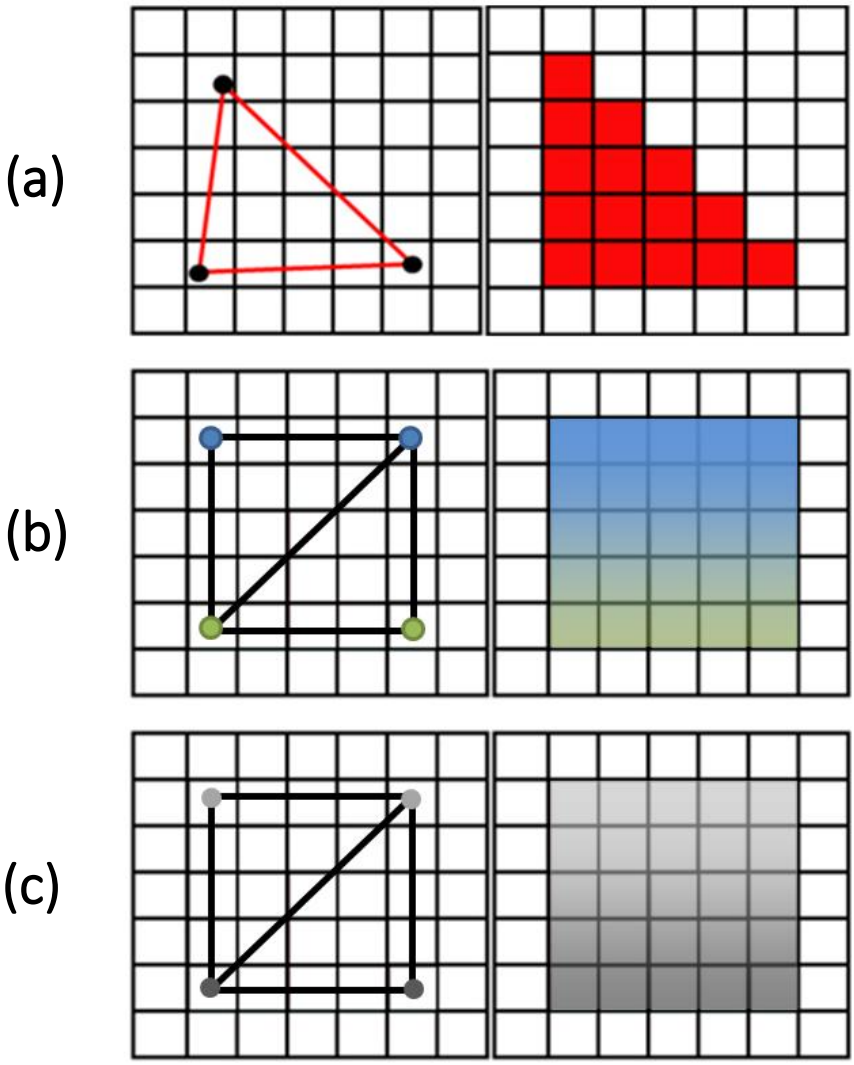


Figure 3.22 The three rasterization steps. Raster-scan (a) determines which pixels (fragments) are covered by a triangle. Color interpolation (b) assigns color values to fragments, by interpolating the color information of the vertices that form each primitive. Depth interpolation assigns depth values to each fragment by interpolating the depth information of the same vertices.

The end-result of the aforementioned sequence of operations is a collection of fragments, each characterized by a color and depth value, as well as a set of coordinates corresponding to a certain pixel of the display monitor. Since however depth has not yet been utilized, the same screen pixel can be assigned to multiple fragments. Consequently, producing the final pixel image requires an additional operation, which will compare the depth information of overlapping fragments, and discard those hidden behind other fragments. Although many depth-sorting algorithms have been proposed in the past, the most common method used by modern computer-graphics libraries and GPU manufactures is the depthbuffer method, also called the *z-buffer* algorithm.

This algorithm utilizes two memory buffers of equal sizes. The first, called the *colorbuffer*, is used to store color per pixel information. The second, called the *depthbuffer* or the *z-buffer*, is used to

store depth per pixel information. For each frame to be rendered, the depthbuffer is initialized with a value equal to the depth of the far clipping plane. Then, the complete collection of fragments is tested processed in a sequential order, as Table 3.1 illustrates. If the depth of a fragment is smaller than the corresponding (based on its pixel coordinates) depthbuffer value, both buffers are updated to the new fragment color and depth values.

Table 3.1 The *z-buffer* algorithm

```
For Each primitive
    For Each fragment of the primitive with pixel coordinates (x,y), color c and depth d
        If d < Depthbuffer at (x,y) then
                Set Colorbuffer at (x,y) = c
                Set Depthbuffer at (x,y) = d
        End
    End
End
```

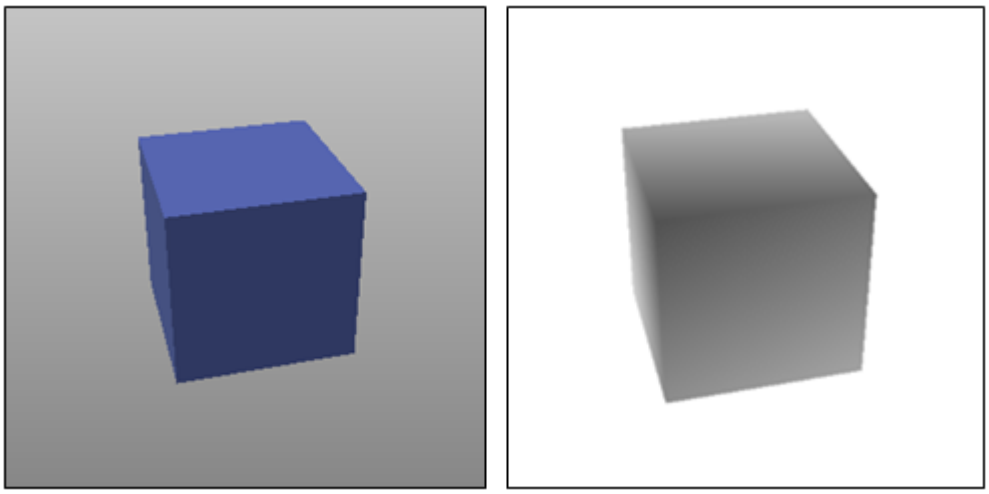


Figure 3.23 The colorbuffer (left) and depthbuffer (right).

The final outcome of the z-buffer algorithm is a colorbuffer containing the actual color pixel image that will be drawn on the display monitor. This color is essentially derived as a result of the color content of the virtual object affected by various different types of light sources (ambient, diffuse and specular light). Later chapters will discuss the importance of depth-buffering for solving the *occlusion problem*, a very significant issue in Augmented Reality.

Chapter 4

An Augmented Reality Framework for MIS Simulation

Modern computer-based laparoscopic simulators are highly sophisticated virtual reality devices, equipped with state-of-the-art hardware and a software components. Combined into a solid simulation framework, these components provide the tools for design and implementation of configurable surgical training scenarios in a controlled environment. The ultimate goal of this thesis was the implementation of an Augmented Reality equivalent to the existing Virtual Reality laparoscopic simulation frameworks, introducing the increased visual realism of AR in the field of computer-based laparoscopic simulation. This Chapter provides an overview of a prototype Augmented Reality surgical simulation framework, describing how the typical architecture of VR-based frameworks was redesigned and enhanced to suit the specific needs of AR-based laparoscopic simulation.

4.1 Research & Development Goals

As stated in Chapter 1, the main objective of the present thesis was the investigation, development and integration of methods for the development of techniques for AR-based training and assessment in MIS simulation. According to what was discussed in the introductory Chapters, the commercially available surgical simulators are either box-trainers utilizing a combination of real instruments and inanimate models or VR devices utilizing a combination of custom instrument-like apparatus and virtual models. Our primary goal was to merge these categories of simulators using a mixed-reality approach, creating a surgical simulation framework that would utilize real laparoscopic instruments and a standard box-trainer environment enhanced with AR graphics. Since AR is by definition a mixture of reality and virtual reality, the initial motivation was the development of a simulation framework that would provide the means for introducing virtual elements within the box-trainer scene and allowing interaction between these elements and real surgical tools.

A high-end surgical simulator must fulfill a diverse set of requirements, covering a wide range of research fields from mechanical/electronic engineering to state-of-the-art software design. Developing such a framework from scratch within the limits of a PhD thesis is an almost impossible task. On the other hand, the minimum demands for the creation of even a very basic surgical simulation framework still requires understanding and involvement with multiple topics of research. Based on these facts, the present project was therefore decided to focus solely on the implementation of vital surgical simulation components, excluding some important but non-vital parts such as haptic-feedback, integration of sounds etc. After a thorough background research followed by a preliminary brainstorming session, we concluded on three prerequisites that would provide the building blocks for the creation of a prototype AR framework:

1. Integration of AR graphics into a standard box-trainer environment.
2. Integration of real-time rigid/soft body dynamics based on the specific needs and limitations of AR.
3. Implementation and/or utilization of robust and accurate techniques for laparoscopic instrument pose tracking.

Addressing potential issues and deriving with robust solutions regarding the aforementioned prerequisites would be a great step towards the development of a functional framework and hence a basis for the design and implementation of AR-based scenarios for training and assessment of fundamental surgical skills such as depth perception, hand-eye coordination and bimanual operation in an AR environment. As a final goal of this thesis, we aimed to assess the construct validity of the framework itself, but also derive important findings regarding the potentiality of using AR technology in the field of laparoscopic simulation.

4.2 A General Overview of Surgical Simulation Frameworks

A standard computer-based surgical simulator is fundamentally a multisensory mechanical interface accompanied by a software engine, allowing surgeons to practice and enhance fundamental surgical skills in a gaming-style virtual reality environment. To some extent, VR laparoscopic simulation is the medical equivalent of video-gaming, where joysticks are substituted by surgical instruments, and gaming levels are substituted by training tasks.

Contrary to simple gaming joysticks though, the mechanical interface of a surgical simulator consists of precisely engineered mechanical parts equipped with highly accurate electronic sensors. Primarily, this interface includes custom-made devices that replicate real surgical instruments. These devices are equipped with tracking sensors for extracting real-time information regarding the instruments' pose with respect to the simulation environment, as well as additional measurements regarding the opening-closing angle of the instruments handles and the rotation of the instruments' shaft. In addition, the mechanical setup includes a dummy camera, also equipped with pose tracking sensors, which replicates a real surgical endoscope. Finally, some setups utilize haptic-feedback apparatus for simulating the forces applied on the instruments when interacting with either rigid or deformable objects during training. Actions and movements performed by the trainee on the mechanical interface of the simulator are reproduced as actions and movements of virtual their virtual counterparts (instruments and endoscope) within the VR training environment.

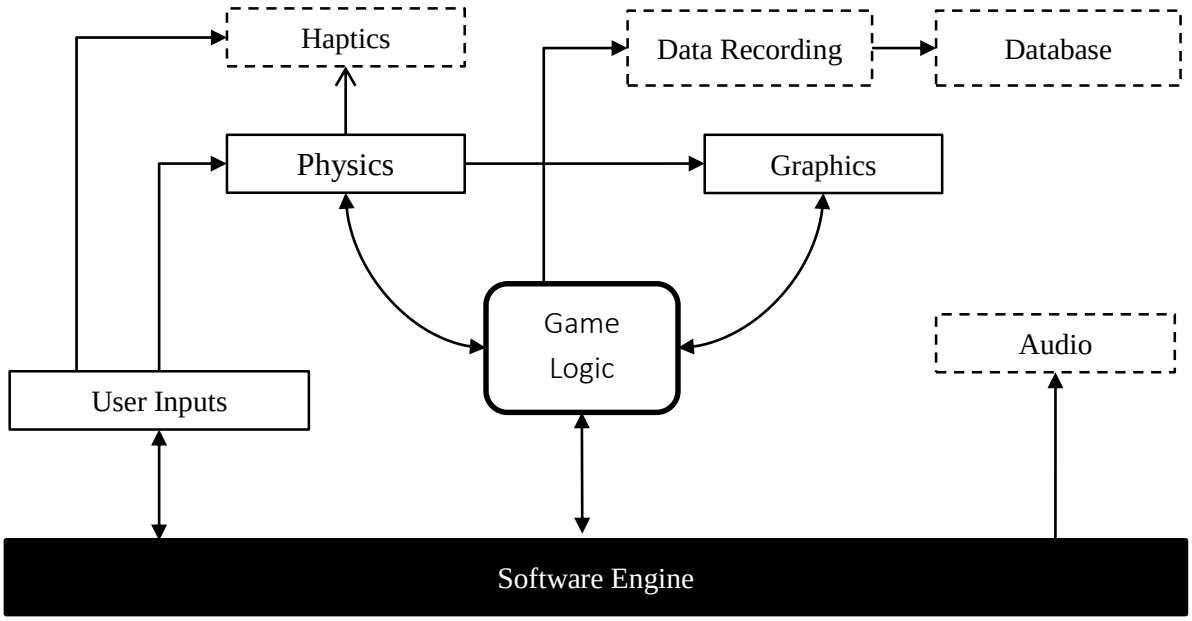


Figure 4.1 Generalized schematic diagram of a game engine

The software interface of a surgical simulator on the other hand, has an almost identical structure to that of a video game platform. In video games, the software interface responsible for functionalities such as handling user inputs, rendering and updating virtual graphics, reproducing sounds etc. is called the *game engine*. Describing a game engine as a single entity is a generalization. Practically, game engines are built upon a wide collection of sub-modules operating in parallel, each responsible for performing different types of functionalities and computations. Similarly in a computer-based surgical simulator, the software interface is a collection of software engines, each specifically designed and parameterized to fulfill certain computations and operations such as reading sensory data, computing the physical behavior of virtual objects based on the movements and actions of instruments, playing sounds, controlling the haptic feedback devices (if employed) and rendering images of the virtual training scene on a display monitor. In addition, the software interface is responsible for executing the game logic behind surgical training scenarios. The latter includes recording and assessing performance metrics, detecting errors, identifying when certain training goals have been achieved and monitoring several other scenario-specific parameters. Finally, the software interface of a surgical simulator provides software tools for storing user information and performance results in databases. Figure 4.1 schematically illustrates the general architecture of a surgical simulation software engine, highlighting the vital elements with solid outlines.

4.3 Overview of the Proposed Augmented Reality Framework

As stated in section 4.1, the goal of the present thesis was the development of an AR surgical simulation framework that would utilize real laparoscopic instruments and a standard box-trainer instead of the highly sophisticated equipment utilized in commercially available VR simulators. Based on this concept, the architecture of our framework was designed following a minimum-hardware approach, utilizing only essential hardware components of a surgical simulation platform. A component-based schematic of the proposed framework architecture is illustrated in Fig. 4.2. As this figure depicts, the framework architecture consists of three levels. The lower level includes the hardware components of the simulator. The middle level, the “heart” of the proposed framework, consists of a collection of software modules integrated into a common software framework. Finally, the top level is the front-end of the simulator, providing a graphical user interface (GUI) that allows users to register and select training scenarios as well as recordings of the metrics extracted during user training, which can be further processed for assessment purposes.

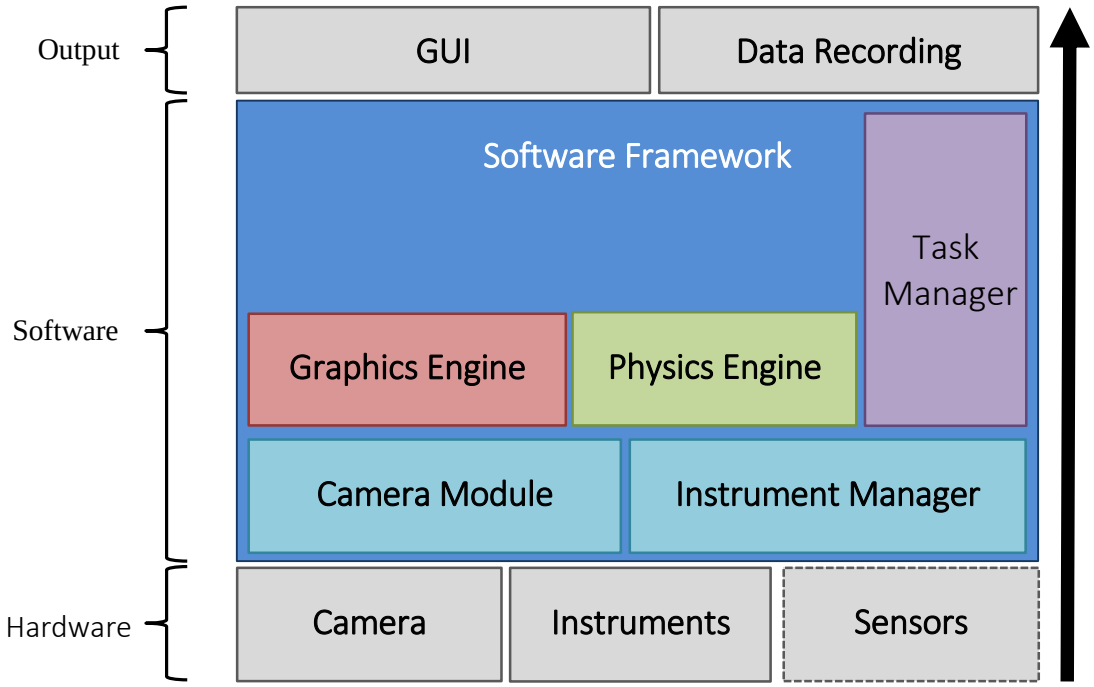


Figure 4.2 Component-based schematic of the proposed framework architecture.

4.3.1 Hardware components

The hardware components of the presented framework include a standard PC with a monitor (Intel^{R} Core^{TM} 2 Duo 3.1 GHz), a standard laparoscopic box-trainer, an endoscopic camera which can either be a USB (Logitech C905 Webcam) or a Firewire camera (PtGray Flea®2) with appropriate wide-angle lenses, as well as a set of real laparoscopic instruments. In addition, the hardware components include sensory devices utilized for instrument pose tracking. Specifically, the framework has been designed using trakStar^{TM} (Ascension Tech Corp., Burlington, VT) electromagnetic (EM) position-orientation sensors for real-time pose tracking of laparoscopic instruments, utilizing a novel sensor calibration technique produced and published during this thesis [155]. As later Chapters will describe however, we have also experimented with image-based instrument tracking solutions [156, 157]. Although utilization of sensors allows the implementation of more advanced training scenarios, during this thesis we have created some basic AR training tasks that did not require integration of any sensory device whatsoever. Consequently, the proposed simulation framework can be designed in two different versions of the same hardware setup, one utilizing sensory devices for laparoscopic instrument tracking and one utilizing vision-based tracking techniques. The advantages and disadvantages of version are thoroughly discussed in the relevant publications, as presented in the result sections of the current dissertation.

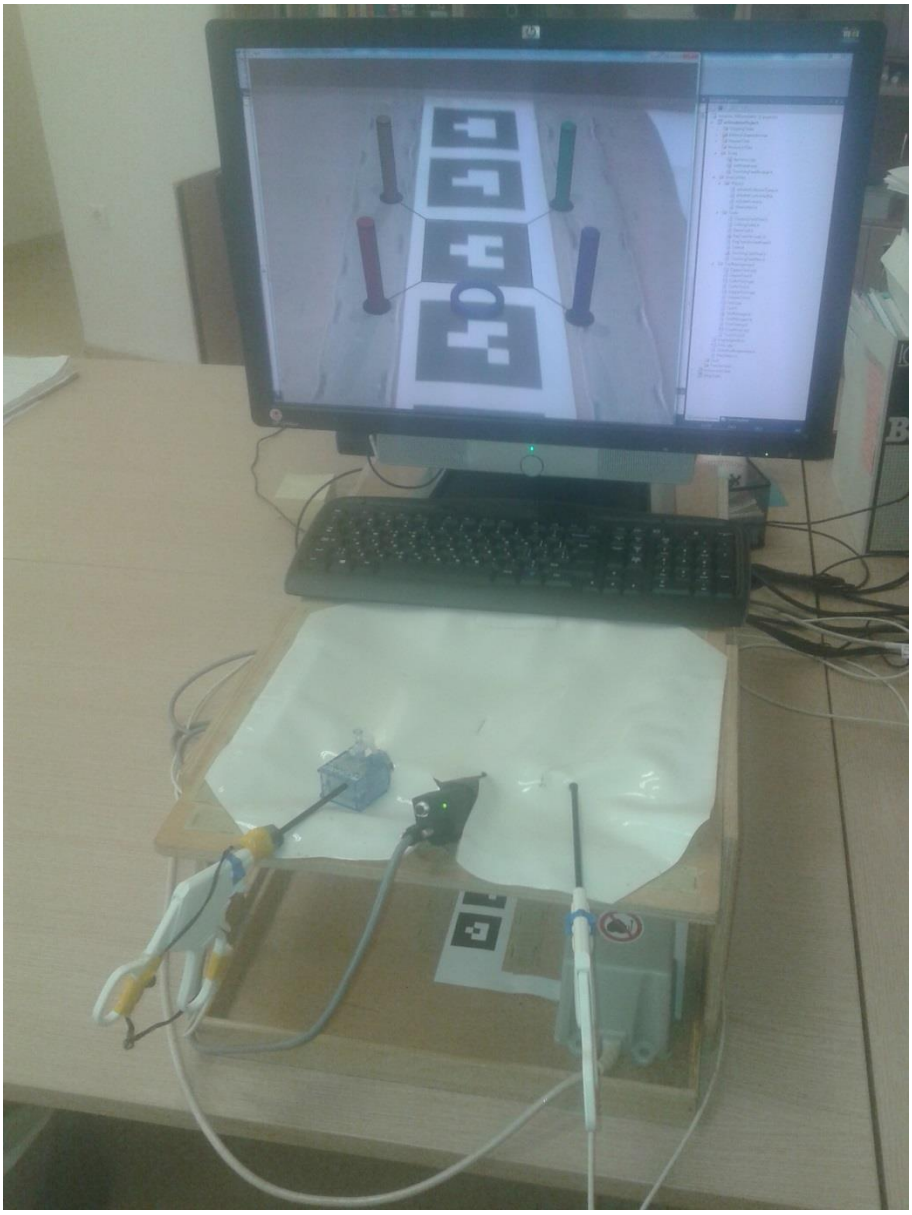


Figure 4.3 Hardware setup of the presented framework

4.3.2 Software components

The second level of the proposed framework is a software engine, consisting of a collection of software subsystems. The structure of this software engine was designed in advance, following the general architecture of game engines. To fulfill the high real-time requirements of laparoscopic simulation, allow direct interaction with hardware devices such as EM sensory devices etc., and provide a flexible object oriented design, the software framework of the proposed simulation platform is implemented in C++. Every sub-module is designed as C++ abstract class responsible for performing specific operations. The software engine of the simulator consists of the following sub-modules:

Camera module: A C++ class that provides connectivity and control functions of the camera hardware. This module encapsulates methods that allow utilization of both standard USB cameras and IEEE 1394 Firewire cameras. The camera module is designed to provide the software functions for real-time acquisition of camera frames.

Instrument manager: This module is a high level wrapper class that encapsulates all the essential methods for instrument tracking. This module acquires inputs from a camera or EM sensors (depending on the instrument tracking method employed) and returns real-time information regarding the instruments' pose and state. The instrument manager is a configurable module, supporting different instrument types depending on the needs of each training scenario. In this thesis we have developed training tasks involving several types of instruments such as laparoscopic graspers, laparoscopic clipper and laparoscopic scissors.

Graphics engine: This engine is responsible for producing AR visualizations on top of a box-trainer scene. During this thesis we have developed two variations of graphics modules, a primitive version based on OpenGL[158] and an improved version utilizing Ogre3D [159]. A detailed description of the graphics engine architecture is provided in the following Chapter.

Physics engine: This engine is responsible for real-time rigid/soft body dynamics simulation and collision detection. Operating in parallel with the graphics engine, this module is responsible for simulating the behavior of virtual objects introduced at the training environment. The physics engine of the proposed framework is designed using the BulletPhysics [160] library as a basis. A detailed description of the physics engine architecture is provided in the following Chapter.

The presented software engine is designed in modular manner where each subsystem operates as a black box, receiving certain inputs and exporting the corresponding outputs. This design pattern allows great flexibility in experimentation of various solutions regarding each sub-module, without requiring significant modification of other sub-modules. For instance, the instrument manager can be redesigned to implement alternative instrument tracking techniques without any additional modification of the other software modules, as long as the output maintains a predefined format.

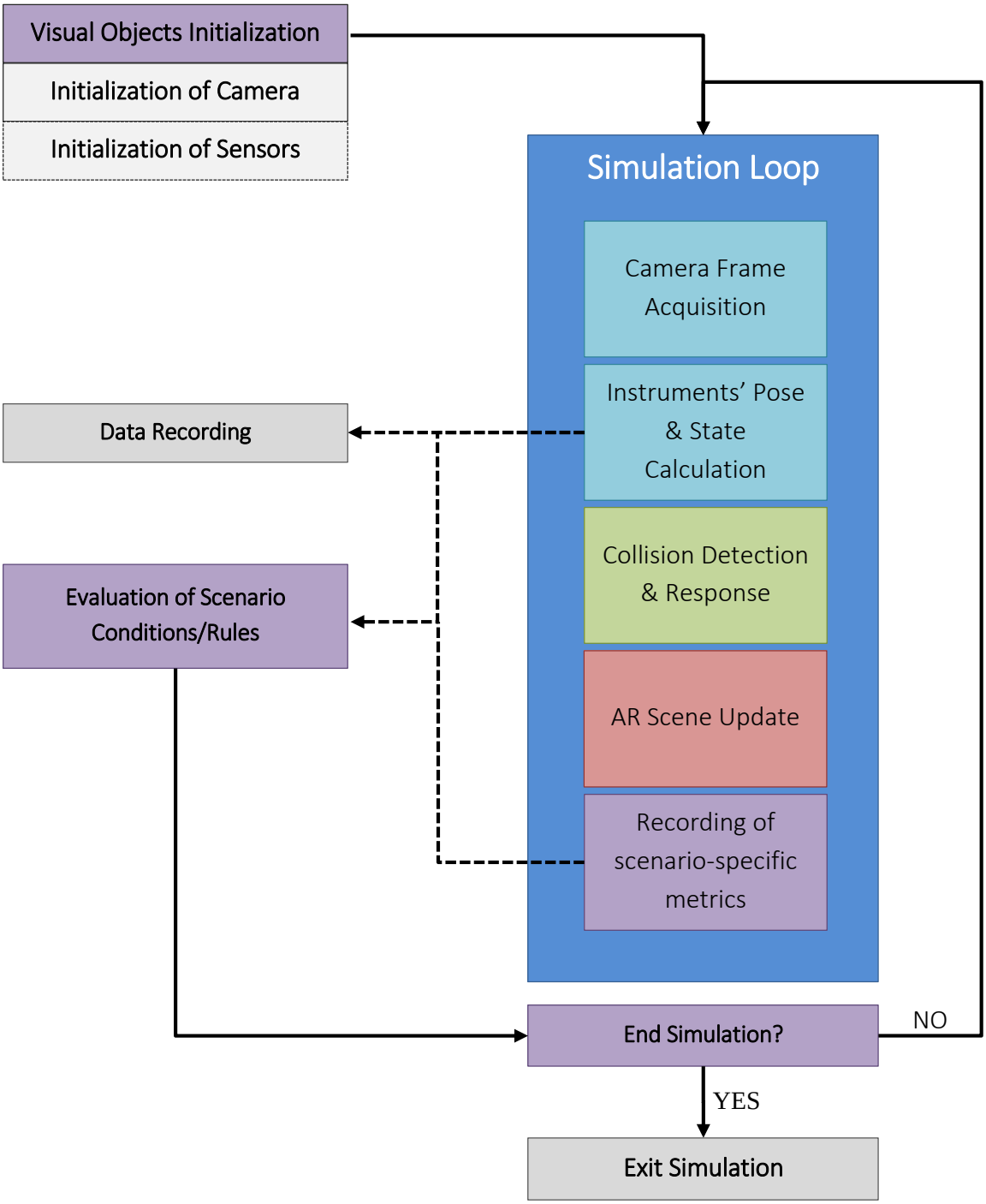


Figure 4.4 Flowchart of the procedures performed during execution of a training task. Each procedure is denoted by a color, indicating the corresponding module as illustrated in Fig.2.

The aforementioned software modules seamlessly interact and exchange data during the execution of a training session, following the specific rules and directions of a training scenario.

This is achieved in a sequential loop that performs scenario-specific operations in a video-gaming loop manner, denoted in Fig. 4.4 as the *Simulation Loop*. Responsible for initializing a training scenario and executing the simulation loop is a top-level class called the *Task manager*. This manager provides the definition of a training scenario, the instruments and virtual objects involved in an exercise as well as the scenario directions and rules.

4.3.3 The Simulation Loop

Before stepping into the technical details and principles of operation regarding the core modules of the presented framework, it is important to describe the sequence of operations performed within the software engine of the framework during the execution of a training session. As depicted in Fig. 4.4, the initialization of a training session requires initialization of all the essential components such as camera and sensors (if applicable) as well as the virtual elements involved in the exercise. As it will be described in the following Chapter, the latter includes both visual and behavioral models of virtual objects.

Following the initialization stage, execution of a training session takes place within a simulation loop. Similar to the game loops in PC games, the simulation loop is a pipeline of operations infinitely performed in a sequential order until certain criteria/conditions have been met. For instance, in a laparoscopic clipping exercise where users are requested to clip a virtual artery at a predefined location, the simulation loop ends once the trainee successfully clips the artery. Depending on each training scenario however, such conditions can also be depended of other parameters such as a time limit, a catastrophic error etc.

Although the exact procedures taking place within the simulation loop are scenario-depended, the general sequence of operations for the implementation of an AR-based training session can be described as follows:

Step 1: Acquisition of a camera frame. (*Camera module*)

Step 2: Calculation of the instruments state, including 3D pose and additional instrument-specific parameters such as the opening-closing angle of the instruments' handles. (*Instruments manager*)

Step 3: Simulation of the virtual objects physical behavior based on instruments' state. This step includes detection of collisions between objects, and computation of collision responses and deformations in scenarios that involve deformable bodies. (*Physics engine*)

Step 4: The resulting poses or states, as obtained from the previous step, are utilized for refreshing the AR training scene. (*Graphics engine*)

Step 5: Calculation and evaluation of task-specific metrics. Assessment of simulation status (continues or stops the simulation loop). (*Task manager*)

4.3.4 Graphical User Interface and Data Recording

At the front-end of the presented framework, a graphical user interface (GUI) accompanies the hardware and software setup, allowing trainees to create a user accounts and practice on the available training scenarios. Although development of surgical simulator at a commercial level is out of scope regarding the current thesis goals, we have equipped the proposed framework with some basic data recording functionalities. During each training session, performance metrics and scenario results can be stored in external data files, thus allowing further processing for performance and construct validity assessment purposes. Figure 4.5 illustrates the GUI of the proposed simulation framework.

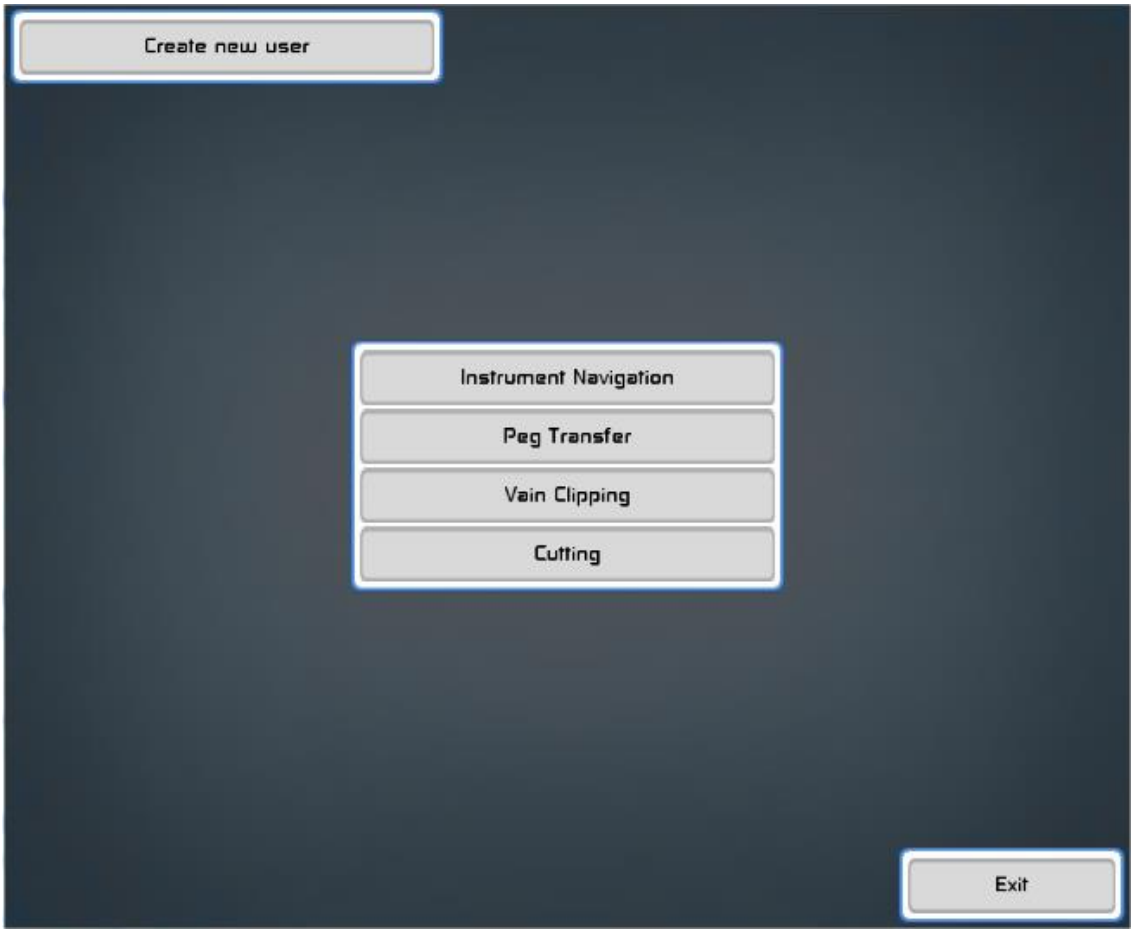


Figure 4.5 The Graphical User Interface (GUI) of the proposed surgical simulation framework.

Augmented Reality Graphics Engine

A core component of the proposed simulation framework as highlighted in Chapter 4 is the software engine responsible for the production of real-time AR graphics. Chapter 3 provided a detailed description regarding the sequence of operations for rendering 3D virtual worlds on a 2D display monitor, known as the *rendering pipeline*. As already stated in the introduction of this thesis, AR is fundamentally the superimposition of virtual objects on top of a real camera image. The AR graphics engine of the proposed simulation framework, built upon two open-source computer graphics libraries, OpenGL and Ogre3D, is responsible for the creation of a mixed-reality box-trainer environment, thus providing a basis for the development of AR training scenarios. This Chapter presents the operating principles of an AR graphics engine built for the purposes of this dissertation, describing the important AR rendering steps and how these where practically implemented with the integration of ARToolkit, OpenGL and Ogre3D.

5.1 Introduction

Although this mixture of virtual and real world sounds somehow exotic, it is essentially achieved with only minor, yet significant enhancements of the *rendering pipeline* operations. During this thesis we have experimented with two approaches regarding the creation of AR graphics, one based on OpenGL and one based on Ogre3D. Although each approach required a different implementation, the general principles of AR rendering were common in both versions. The following sections present the main rendering steps for achieving real-time AR graphics and their practical implementation, as utilized during the development of the proposed surgical simulation framework.

5.1.1 Coordinate Spaces and Transformations

As discussed in section 3.3 of Chapter 3, VR scenes are built upon a central reference frame called the *world space*, used to define the pose of any virtual object introduced in a VR scene. The same logic implies in AR scenes as well. A global reference frame is defined, acting as a link between the real and virtual world. This global frame usually corresponds to a physical place of the real world such as the floor of a room. The pose of every virtual object introduced within the AR scene is expressed with respect to this central reference frame. Thus, having provided the link between real and virtual world in the form of a common reference frame, introduction of virtual elements within the real world is achieved as a sequence of transformations that provide a mapping between real and virtual world coordinates, as Fig. 5.1 illustrates [83].

In addition, as VR scenes utilize virtual cameras for obtaining images of the virtual world, in an AR environment a camera is also used to obtain images of the real world. Consequently, similarly to VR, rendering virtual objects on top of real images also requires the definition of a *view matrix*. Contrary to VR however where the view matrix is manually defined, in AR applications this matrix must correspond to the actual pose of the real camera with respect to the real world scene.

Equivalently to the essential transformation matrices involved in the *rendering pipeline* as described in Chapter 3, the essential coordinate transformations involved in an AR application are:

$\mathbf{M}_{\boldsymbol{o}}$ (Object to World Space): A $4x4$ homogeneous transformation matrix providing the pose of a virtual model with respect to the *world space* (described as the *model transform* in Chapter 3)

$\mathbf{M}_{\boldsymbol{c}}$ (World Space to Camera Space): A $4x4$ homogeneous matrix providing the transformation from *world space* to *camera space* (described as the *view transform* in Chapter 3)

$\mathbf{M}_{\boldsymbol{p}}$ (Camera Space to Image Plane): A $3x4$ projection matrix providing the perspective projection transformation from *camera space* to *pixel space* (described as *viewport transform* in Chapter 3).

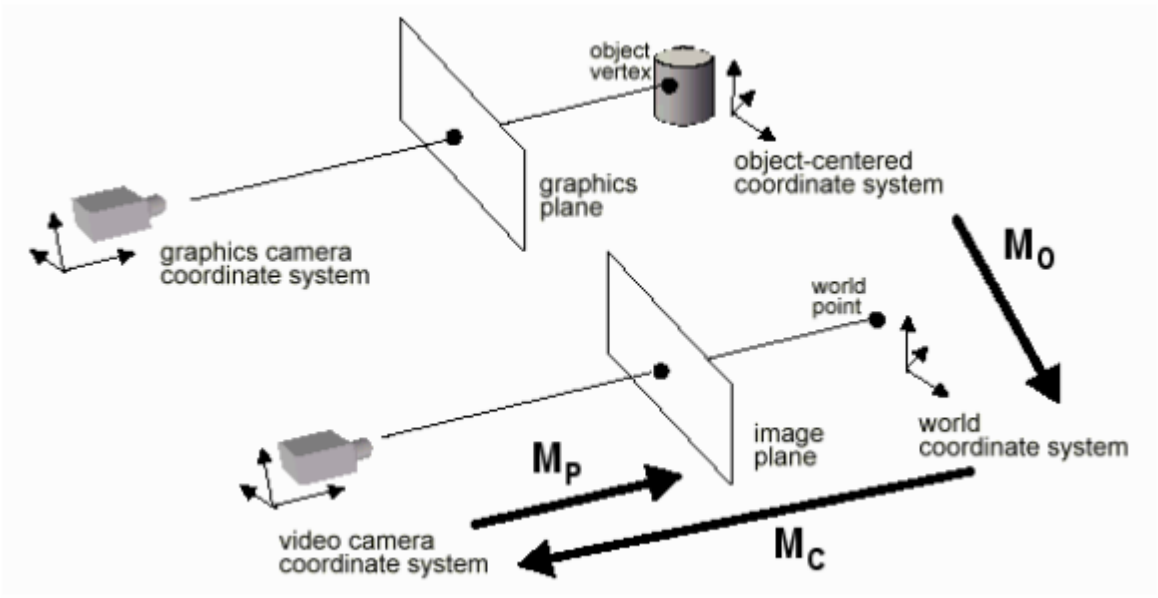


Figure 5.1 Coordinate Systems in an Augmented Reality Environment (image taken from [83])

Application of this sequence of homogeneous coordinate transformations provides the mapping from coordinates of a virtual model's *object space* to pixel coordinates is achieved as:

$$\begin{bmatrix} u \\ v \\ h \end{bmatrix} = \mathbf{M}_p \cdot \mathbf{M}_c \cdot \mathbf{M}_o \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \tag{5.1}$$

The same approach is followed in the proposed framework. A global reference frame is assumed, located at the center of the box-trainer bottom plane with the z-axis pointing upwards. This reference frame provides a global coordinate system that can be applied to express both real and virtual coordinates. Hence, the pose of any virtual model introduced to the training scene as well as the pose of the camera with respect to the training environment is expressed with respect to this reference frame. In addition, as later Chapters will show, the poses of the laparoscopic instruments are also expressed with respect to this reference frame, thus allowing simulation of interactions between real instruments and virtual objects.

5.1.2 Camera Parameters

In AR applications, rendering of virtual models is achieved using standard computer graphics techniques. Hence as in VR, the position and orientation of virtual models with respect to the world ($\mathbf{M}_o$) depend on the desired scene setup. Unlike pure VR however where a virtual camera is utilized, the camera *view matrix* ($\mathbf{M}_c$) and *projection matrix* ($\mathbf{M}_p$) in AR applications depend on the pose and design characteristics of the real camera used to obtain images of the real world

respectively. In the field of computer vision, these are known as intrinsic and extrinsic camera parameters.

5.1.2.1 Intrinsic Parameters

The intrinsic camera parameters describe the physical properties of a real camera, including optical, geometric and digital characteristics/specifications. Specifically, the intrinsic camera parameters are concerned with the following camera characteristics:

1. Focal length (f)
2. Principal point offset (o_x, o_y)
3. Pixel dimensions in the horizontal and vertical direction (s_x, s_y) in millimeters
4. Optics distortions due to camera lenses shape

Creating the projection matrix ($\mathbf{M_p}$) requires knowledge of the first three parameters. Similar to the virtual cameras described in Chapter 3, the focal length of a real camera is the distance between the image plane and the focal center of the camera. Since real cameras utilize optical lenses, the focal length is depended of the lens shape and curvature. In virtual cameras of computer graphics applications, the principal point is located at the center of the image plane. This is not the case though for real cameras since due to minor assembly inaccuracies, the image plane is not at a perfect right angle with respect to the principal axis [161]. In addition, the pixels of real cameras are usually non-square, having thus different dimensions across each direction (x and y). According to what was described in Chapter 3 regarding the mathematics of projection, the aforementioned parameters are essential for achieving a mapping of *camera space* coordinates to pixel coordinates in the *image space,* calculated as:

$$x = -(x_{im} - o_x) \cdot s_x \,,\; y = -(y_{im} - o_y) \cdot s_y \tag{5.4}$$

Based on this mapping, the AR projection matrix can be defined as follows:

$$\mathbf{M}_p = \begin{bmatrix} -f/s_x & 0 & o_x & 0 \\ 0 & -f/s_y & o_x & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.2}$$

Real world cameras also demonstrate radial distortions, usually on the periphery of the image, which can be corrected with the following radial displacement [162]:

$$[x, y] = [x_d(1 + k_1 r^2 + k_2 r^4) \,,\; y_d(1 + k_1 r^2 + k_2 r^4)] \tag{5.3}$$

where k_1 and k_2 are called the radial distortion coefficients, x_d and y_d refer to the distorted coordinates of a point in the *camera space* while $r^2 = {x_d}^2 + {y_d}^2$. Radial distortion parameters are not taken into account for the construction of the projection matrix; however they are

essential for correcting distortions in the transformation of image coordinates to world coordinates. This transformation is required in computer vision algorithms for 3D tracking of pattern markers that will be described in later section of this chapter [79].

5.1.2.2 Extrinsic Parameters

As opposed to the intrinsic parameters that describe internal parameters of the camera, the extrinsic parameters refer to external parameters of the camera and more specifically, to its position with respect to the real scene. The same way that the *view matrix* ($\mathbf{M_c}$) in pure VR applications provides the spatial relationship between a virtual camera and the virtual world origin, the external camera parameters provide the pose of the camera relative to a known reference frame belonging to an AR scene. This relationship is expressed as a $4x4$ homogeneous transformation matrix, consisting of a $3x3$ rotation matrix and a translation vector. Deriving the pose of a camera with respect to the real-world environment is a pre-requisite for realistic registration of virtual objects within the real world image. Several techniques for real-time camera pose estimation have been proposed in the current literature, divided into two categories; sensor-based techniques[63] and vision-based techniques [70, 163]. In addition to these categories, hybrid techniques have been proposed, utilizing a combination of the aforementioned categories [63].

In the present thesis, we aimed to design training scenarios where the user would manipulate a real camera in order to obtain the desired view of the training environment. Taking into account the specific needs of computer-based surgical simulation, we concluded that the proposed framework should involve a camera pose estimation technique that would fulfill the following requirements:

- Sub-millimeter and sub-degree accuracy in pose tracking
- High tracking stability (small jitter)
- High tracking rates (> 30Hz that is the lowest framerate of real-time applications)
- 6 DOF in camera movements, allowing the implementation of non-static camera training scenarios.

At the time the presented framework was developed, the most widely used technique for real-time camera pose estimation fulfilling the aforementioned requirements, was 2D marker tracking.

5.1.3 Integration of ARToolkit

In a wide range of practical applications, planar (2D) markers are used for carrying information. These applications range from commercial, industrial and shipping data systems where planar barcode markers carry product identification information, to robotic applications where binary markers carry information that allows real-time robot localization[164]. Marker tracking is the most widely accepted technique for camera pose tracking in AR applications [79, 80, 165, 166].

The planar markers used in AR applications consist of a heavy black square outline into which, a unique pattern is printed. This pattern can either be a specific shape or a binary code, as illustrated in Fig. 5.2. Using computer vision algorithms, adaptive thresholding of a camera image allows segmentation of black and white areas within the camera frame. Then, line detection and region extraction allows localization of squares within this image. Finally, template matching yields the identity of these squares, determining if they correspond to pattern markers or not. Deriving the pose of each detected marker with respect to the camera coordinate system (denoted as $\mathbf{M}_c$ in Fig. 5.1) is achieved as a function of its corner pixel coordinates.

Our framework integrated the ARToolkit [167] marker tracking library. ARToolKit is a C/C++ software library developed for AR applications, which provides algorithms for real-time pattern marker tracking. In addition, ARToolkit is also equipped with camera calibration algorithms, allowing accurate calculation of intrinsic parameters for USB and FireWire cameras. In addition, the library features video grabbing modules supporting a wide variety of compatible cameras. The basic version of ARToolkit is distributed free for non-commercial or research applications. It has been initially developed by Dr. Hirokazu Kato of Osaka University, Japan, and is supported by the HITLab at the University of Washington and the HITLabNZ at the Canterbury University in New Zealand.

Figure 5.2 A matrix-code marker (left) and a pattern marker (right) for camera pose tracking in AR applications.

At the initial stages of this thesis we experimented with the open-source free version of ARToolkit (Version 2.2). However, the instrumental role of pattern-marker tracking in our AR surgical simulation platform led us to the integration of a commercial version of the same library that is called ARToolkitPro (Professional Version 4.4), distributed by ARToolworks [168]. Both ARToolkit and ARToolkitPro provide high-level routines for pattern marker tracking. The latter however utilizes more advanced computer vision algorithms, achieving higher tracking accuracy and improved stability.

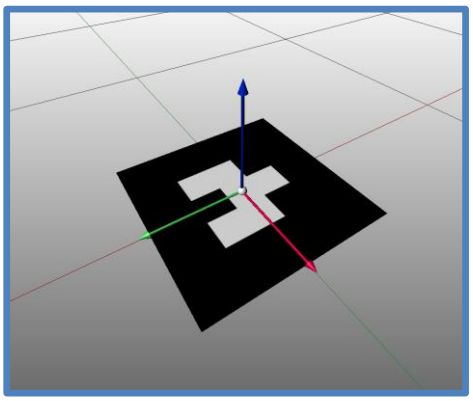


Figure 5.3 The right-handed coordinate system of an AR marker used to define a global coordinate space in the presented framework. The x (green) and y (red) axes are co-aligned with the planar surface of the pattern marker, while the z axis (blue) is pointing upwards.

As depicted in Fig. 5.3, a pattern marker is considered a reference frame whose $x - y$ plane is aligned with the surface of the marker and its z axis is pointing upwards. The marker tracking technique described earlier provides the pose of the camera with respect to this coordinate system and hence, a pattern marker can be used to define the *world space* of an AR scene.

In the proposed framework, a 2D pattern marker is attached at the bottom center of the box-trainer. Real-time tracking of this marker provides the spatial relationship between camera and training environment, thus allowing real-time augmentation of virtual objects within the box-trainer scene.

5.1.3.1 Acquiring Camera Parameters with ARToolkit

The ARToolkit library is distributed with an optimized camera calibration technique that allows extraction of intrinsic camera parameters with an automated calibration procedure. This techniques works by acquiring images of an A4 sized chessboard-like calibration pattern consisting of 6x8 black and white squares. In order to ensure accurate calibration results, the pattern must be printed without any scaling so that every square of the chessboard pattern will be exactly equal to 30mm, and affixed in a perfectly flat surface such as the wooden surface of a table etc.

The ARToolkit calibration method is based on edge and corner detection. The straight lines of the chessboard appear curved due to camera lens distortion and hence, measuring the distance between corners and comparing with the expected results the method is able to calculate a lens distortion map. Based on this map, the distortion correction parameters are calculated and then exported/stored in a camera parameter file. To increase accuracy, multiple images of the chessboard must be acquired, each from different angles and distances. In addition, the intrinsic camera parameters depend on the focus setting of a camera, and hence it is a good practice to perform the calibration protocol using the focus setting that will be mostly used in an application.

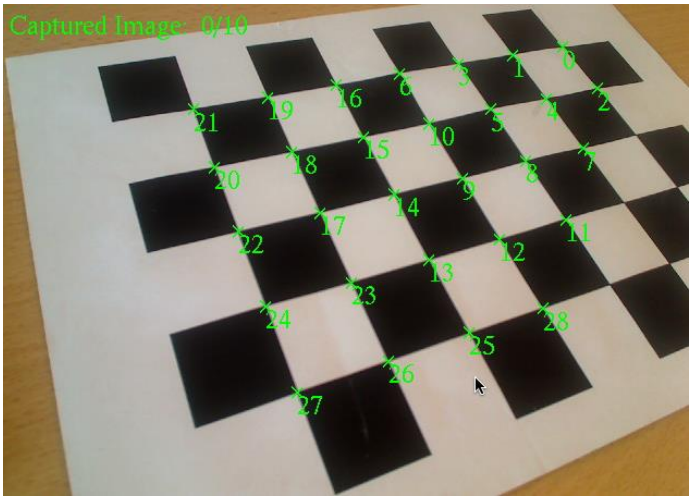


Figure 5.4 A snapshot of camera calibration procedure provided by ARToolkit, showing the calibration chessboard pattern. Accurate detection of the chessboard corners is essential for accurate calculation of the intrinsic camera parameters.

As stated in Chapter 4, in our framework we utilized both a Logitech C905 USB camera and a PtGray Flea®2 Firewire camera. Consequently, we obtained a separate parameter file for each of the two cameras that we used. Regarding focus, we selected a setting that provided a clean focus within the box-trainer area and performed the calibration protocol using this focus setting for each of the two cameras. It is important to note that Logitech C905, as is the case for most of the commercial USB Webcams, operates on an autofocus mode. To achieve the best possible calibration accuracy, autofocus was disabled using the camera driver utility provided by the manufacturer. On the other hand, Flea®2 has a manual focus control, which can be mechanically fixed at the desired focus setting.

5.1.4 Creating an AR scene in a standard box-trainer

As later sections will show, in the proposed framework we integrated two different setups regarding the computer graphics libraries utilized for rendering. Although implementation differs between setups, the required steps for achieving AR rendering are the same; the sequence of operations for the creation of an AR scene is similar to the rendering steps described in Chapter 3, with three simple but significant steps added to the rendering process:

5.1.4.1 AR rendering step 1 - Projection matrix construction

According to Chapter 3, the viewing volume (frustum) of a virtual camera employed in a typical VR application is a truncated pyramid consisting of six clipping planes. The frustum and consequently the clipping planes are normally application-defined, depending on the desired characteristics of the virtual camera. Using these characteristics, a camera projection matrix is constructed, allowing the transformation of *camera space* coordinates to *clip space* coordinates. In AR applications,

rendering of virtual models is performed using the same rendering techniques as in VR and hence a definition of a virtual camera is also required. However, since virtual models are combined with images of a real scene, the virtual camera frustum and consequently the camera projection matrix needs to reflect the characteristics of the real camera. Section 5.1.2.1 provided the mathematical formulation for the creation of the camera projection matrix and section 5.1.3.1 described how intrinsic camera parameters are pre-calculated using ARToolkit. Using the pre-calculated intrinsic camera parameters and Eq. 5.2, a projection matrix corresponding to the actual frustum of the camera utilized in our setup is constructed at the initialization stage of the application. Since this matrix only depends on the physical properties of the camera, its calculation is performed only once during the initialization stage of a simulation session.

5.1.4.2 AR rendering step 2 - Camera background projection

The first step towards the creation of an AR scene is the introduction of a planar surface that will serve as a canvas for drawing the camera frame. In our experimental setup, the camera resolution is set at 920x765 pixels. Hence, images acquired by the camera are 4x920x768 arrays of data, containing color pixel information (RGBA format). In order to draw these images on top of any 2D plane, a texture of equal size is created. When a new camera frame is acquired, the pixel values of this texture are updated with the new pixel color data. Assigning the resulting texture to a 2D virtual plane allows real-time rendering of the camera frames. In order to avoid distortions like stretching of the camera image, this plane must have equal proportions (height/width ratio) to the camera frame. This process is a standard practice in computer graphics applications that involve video rendering within virtual environments. To ensure that any virtual models introduced in the scene will be rendered in front of the camera image, the textured planar surface must be located at the furthest visible location of the viewing volume. Taking into account the definition of the viewing frustum and the camera-clip space conversion presented in Chapter 3, the ideal candidate for this planar surface is the far clipping plane. As described, the frustum is defined using the real camera parameters and hence, the proportions of the far clipping plane are equal to the proportions of the original camera image. In addition, any virtual object inside the camera viewing volume will be, by definition, in front of this plane and consequently, superimposed on the camera image.

Figure 5.5 illustrates the concept of camera background projection implemented in our simulation setup. The wooden surface appearing on the left image is the inner bottom plane of the box-trainer, obtained using a standard USB of Firewire camera. The right image illustrates how this image is assigned as a texture on the far clipping plane of the camera frustum.

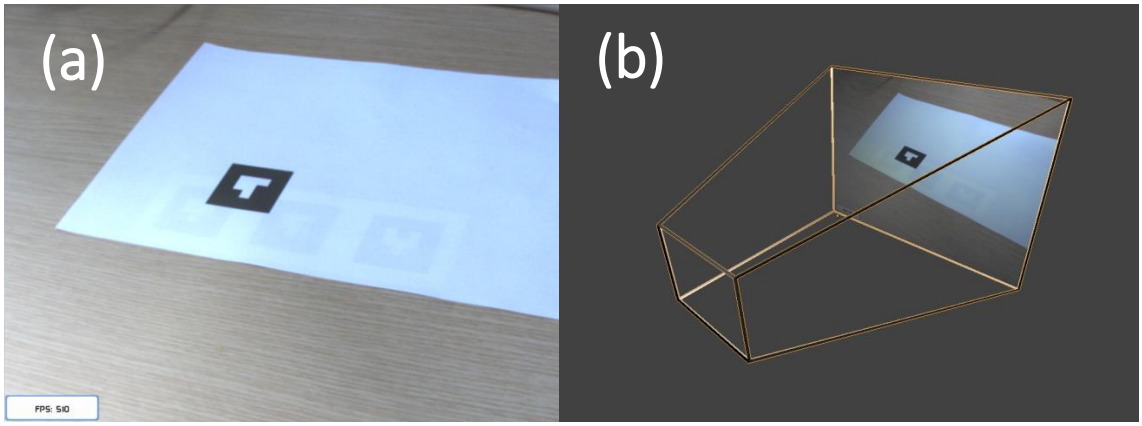


Figure 5.5 Camera background projection. A still frame (a) obtained from the camera is applied as a texture on the far clipping plane of the camera frustum (b).

5.1.4.3 AR rendering step 3 - View matrix utilization

The remaining step for the creation of an AR scene is superimposition of virtual models in visually realistic locations. As it was stated in Section 5.1.3, a pattern marker is employed to define the global coordinate system of the AR environment, located at the center of the box-trainer bottom plane. This marker is visible in the left image of Fig. 5.5. Introducing virtual objects at desired locations within the box-trainer environment is achieved using the coordinate transformations described in Chapter 3. Finally, the remaining step for AR rendering is to define a camera view matrix that corresponds to the pose of the real camera with respect to this coordinate system. (See definition of the *view matrix*, Chapter 3, section 3.4.2). This last piece of information is obtained with ARToolkit pattern marker tracking. The arGetTransMat() function of ARToolkit returns the pose of a pattern marker with respect to the camera coordinate system in the form of a 4x4 homogeneous transformation matrix and hence, it can be directly applied as a camera view matrix, resulting in visually realistic rendering of the mixed-reality scene. This rendering step is performed at a per-frame basis, since the relationship between marker and camera constantly changes during a training session. The end result of this process is illustrated in Fig. 5.6 where a virtual cube is rendered in front of the camera image. The visual outcome creates the impression of a virtual cube positioned on top of the pattern marker at the bottom plane of the box-trainer.

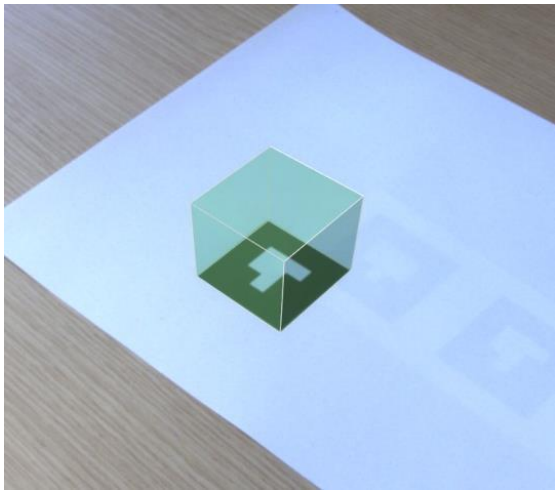


Figure 5.6 A virtual cube rendered on top of a camera frame, using a single pattern marker to define the AR world space.

5.2 ARToolkit, OpenGL and GLUT Rendering

The first step towards the development of the proposed framework was to obtain and assess some preliminary findings regarding the potential use of AR in laparoscopic simulation training. Hence, our first goal was to create some primitive AR training scenarios that did not require implementation of highly realistic graphics or animations, but simple AR visualizations such as 3D spheres superimposed on top of the box-trainer scene in predefined locations. Our initial approach regarding the graphics module that would produce these visualizations was to integrate a graphics library already provided with the ARToolkit distribution, called OpenGL[158]. OpenGL is a low-level C++ computer graphics API (Application Program Interface) developed and maintained by SGI, Silicon Graphics Inc. 1992, providing a fast and portable interface for 2D/3D computer graphics rendering. OpenGL is a popular choice among graphics programmers because it is highly optimized and platform independent. Being a low-level rendering API, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. In order to draw a virtual model in OpenGL, its visual shape must be defined at runtime, using specific function calls that describe: (a) the 3D position of vertices, (b) the color, (c) the normal of each vertex, (d) how these vertices are connected into 3D polygons and (e) the (u, v) texture coordinates of each vertex, as described in Chapter 3. Furthermore, the OpenGL API is not equipped with routines for the creation of a render window or reading inputs (events) from the keyboard and mouse. These functionalities however are applicable using an open-source library called GLUT, which is also included in the ARToolkit distribution. GLUT is equipped with some basic 3D drawing commands, allowing introduction of simple geometric primitives such as spheres, cubes and cylinders into a virtual scene, as well as essential functions for the creation of render windows and the communication with PC peripherals (mouse & keyboard).

Combining ARToolkit and OpenGL/GLUT, the creation of an AR scene is an almost straightforward procedure achieved with a direct implementation of available routines, following the flow of operations shown in Fig. 5.7 and the three AR rendering steps described in Section 5.1.4.

Initialization Stage: A connection to the camera is established using built-in ARToolkit methods. The necessary parameters for initializing this connection are a camera ID and a path to the data file containing intrinsic camera parameters as described in section 5.1.3.1. Additionally at this stage, the ARToolkit tracking functionalities are initialized, providing information regarding the type of 2D markers that the tracker should detect (either pattern markers or binary code markers). In parallel, an OpenGL render window is created. The dimensions of this window must be equal to the dimensions of the camera frame since the latter must completely cover the virtual camera field of view. Having performed the initialization stage, the three AR rendering steps described in Section 5.1.4 are performed in a sequential loop.

Step 1, Projection matrix construction: In OpenGL, defining the viewing frustum of a virtual camera is achieved by passing specific parameters to the glFrustum() function. According to documentation, ARToolKit uses a calibrated camera perspective that typically results in an off axis projection matrix for OpenGL. However, rather than decomposing ARToolKit's projection into separate parameters to pass to glFrustum, the projection matrix can be directly loaded using the command glMatrixMode(GL_PROJECTION_MATRIX) and then use glLoadMatrix(matrix) to feed OpenGL with the appropriate value for the projection matrix.

Step 2, Obtaining a new camera frame: A new camera frame is obtained by calling the arVideoGetImage() function of ARToolkit. This function returns a pointer (e.g. *image) to the memory location that stores pixel color data of the new camera image. These data are important for the next two steps of the AR rendering process.

Step 3, Drawing a camera background: Drawing a camera image on the background of the virtual scene as described in Section 5.1.4, is also achieved using a built-in routine of ARToolkit. First, a call of glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) clears both the color and depth buffers of OpenGL. Then, passing the new image data pointer to arglDispImage(image) creates and fills the background texture of the view frustum with data of the new camera frame.

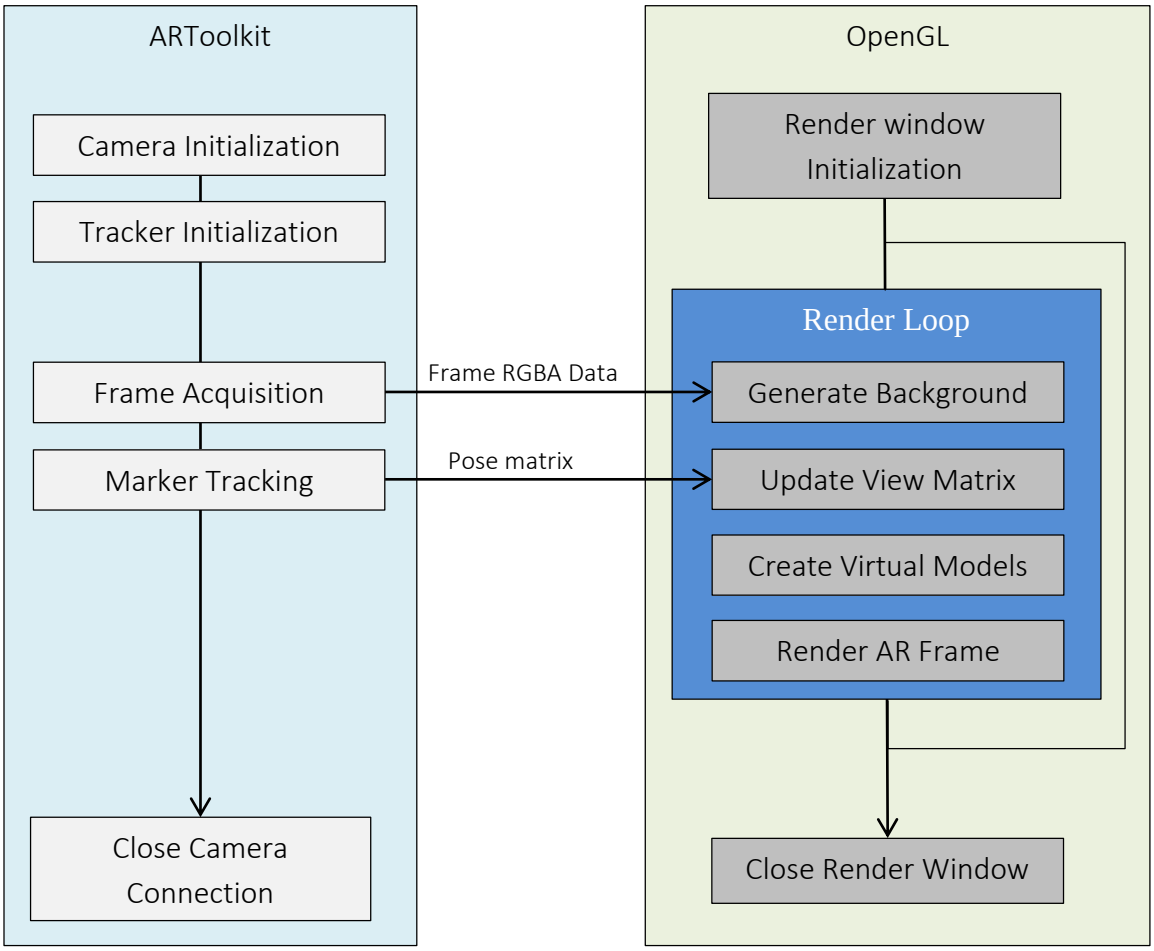


Figure 5.7 Flowchart of operations for AR scene rendering, as performed in the early, OpenGL-based version of the proposed AR graphics module.

Step 4, Setting the view matrix: Pose tracking of pattern markers is achieved by passing the new image data pointer to arDetectMarker() function. This function returns an array of all visible markers, defined by their ID, and their poses with respect to the camera in the form of 4x4 homogeneous transformation matrices. The pose matrix of the marker with ID equal to the pattern marker attached at the bottom of the box-trainer is used as a view matrix in OpenGL. This is achieved by calling glMatrixMode(GL_MODELVIEW) which notifies OpenGL for the loading of a new view matrix, followed by glLoadMatrix(value) to feed OpenGL with the appropriate value for the view matrix. The ID of each marker is taken into account at this stage, since multiple markers might exist in a single camera frame.

Figure 5.7 illustrates the flowchart of operations performed for rendering in the ARToolkit/OpenGL integration. As shown in this figure, steps 1 through 4 are repeated continuously until the application quits, while the initialization stage is just performed on the beginning of a simulation session. It can be notice though that this flowchart is significantly simpler compared to the

simulation loop flowchart presented in Chapter 4. The reason is that the early version of the proposed framework did not include physics-based modeling and simulation of interactions between laparoscopic instruments and virtual models. In [157] for instance, we proposed a very simple training task based on the initial ARToolkit/OpenGL setup, that required trainees to touch two virtual spheres using a laparoscopic instrument (Fig. 5.8b). Since no physics-based interaction between instrument and spheres was utilized, the pose of virtual spheres with respect to the training environment was hardcoded and remained constant during task execution while the only parameter affecting their visual characteristics was their distance to the instrument tip. Updated at a per-frame basis, the spheres became green once the distance of the tooltip to the spheres' center became smaller than their radius, indicating that the tooltip was inside one of the spheres. Due to the lack of physics-based interactions, the initial form of the simulation loop consisted of significantly fewer operations. Essentially, it was an extension of the ARToolkit/OpenGL loop shown in Fig. 5.7 with an addition of functions that provided information regarding the instruments' pose. Using which were used for updating visual objects appearance as well as assessing certain conditions for completion of a training task.

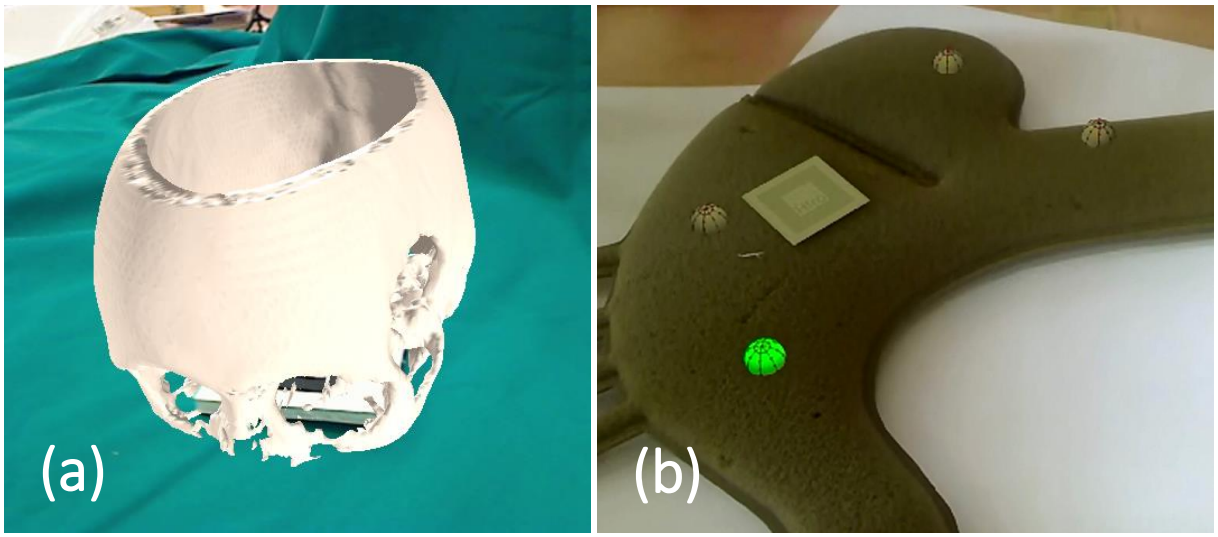


Figure 5.8 (a) A virtual skull rendered on top of a pattern marker using the proposed ARToolkit/OpenGL engine. (b) A simple AR training scenario designed with the ARToolkit/OpenGL version of the presented simulation framework.

5.3 Ogre3D Integration

Based on personal qualitative analysis and user feedback obtained during preliminary experimentations with the ARToolkit/OpenGL setup, we concluded that producing improved AR graphics in terms of visual realism was a necessity towards the development of the proposed simulation platform. Although the initial setup OpenGL allowed the creation of basic training scenarios, the lack of visual characteristics such as realistic materials and texturing, produced a somehow unrealistic visual outcome. Most importantly, the lack of shadows created undesired

effects, making it difficult for users to identify the actual positions of virtual objects with respect to physical objects of the training scene. OpenGL offered all the essential tools for simulating visual effects such as light, shadows and textures. However, creating and optimizing the appropriate rendering algorithms would be a very time-demanding task. Since reinventing the computer graphics wheel was not amongst the goals of the presented thesis, we decided to focus on the integration of a high-level computer graphics library that would include algorithms for simulating the aforementioned effects.

An extra motivation for this decision was our goals regarding the general architecture that we wanted to follow regarding the software interface of the proposed framework. As stated in Chapter 4, from the beginning of the development stage we had decided to follow an object-oriented approach using modular classes for each software component of the framework. This design pattern should be also reflected in the design of the virtual scene. The goal was to make a framework that would allow the creation and simulation of AR scenes involving multiple virtual elements. The most efficient method for implementing these setups in computer graphics applications is the *scene graph* approach. A scene graph is a general data structure commonly used in modern computer games, which arranges virtual scene elements as a collection of nodes in a graph or a tree structure. Scene graphs are particularly useful for organizing/controlling scenes that involve multiple virtual elements. Being linked by a parent-child relationship, virtual objects of a scene graph can be manipulated in groups as easily as manipulating a single object. For instance, applying a transformation to a parent node corresponding to a virtual object will result in the equivalent transformation of every other object declared as a child of this node. As later Chapters will describe, this property of scene graph architecture proved very practical for the integration of real-time physics in the presented framework, since it allowed the definition of advanced virtual structures to the training environment, such as laparoscopic instrument representations consisting of multiple sub-parts etc.

Based on the aforementioned considerations, we concluded on redesigning the presented AR graphics engine based on an open-source computer graphics library called Ogre3D. The Ogre3D library provided a highly optimized implementation of the scene graph approach as well as support of advanced visual effects such as real-time shadows, GLSL shaders[6] etc. Ogre3D follows a different rendering approach compared to OpenGL where virtual objects are defined as independent entities within the rendering loop. In Ogre3D, scenes are built as collections of virtual actors called *scene nodes*, organized in a tree structure. A scene node is a logical element of the scene graph hierarchy, used to organize objects in a scene. It can have multiple child nodes attached to it. Movable virtual objects such as 3D models, cameras and lights are also attached to scene nodes.

[6] OpenGL Shading Language (GLSL) is a high-level shading language based on the syntax of C/C++. GLSL is designed to give developers direct control of the graphics pipeline without having to use hardware-specific languages.

Consequently, rather than moving these individual items around, you move the nodes they are attached to.

Each scene node is a C++ object defined by a large set of properties. The most important piece of information though is the position of the node with respect to the virtual scene. These properties of scene nodes are defined at the initialization stage of the rendering process but can also be updated at runtime. The top hierarchical parent in the Ogre3D scene graph is also a scene node not assigned with a movable object, called *root scene node*. Every Ogre3D node is defined by a 3D pose, expressed in respect with the coordinate system of its parent node. Hence the coordinate system of the root scene node is effectively equivalent to the global coordinate system of VR scenes described in previous sections.

In order to add a virtual object to the scene, this must be described in terms of its visual characteristics, and then assigned to a scene node. An important difference between OpenGL and Ogre3D however is the way of defining visual models. As already mentioned, OpenGL is a low level library that does not provide functionalities for loading visual models. 3D shapes, including their vertex and polygon information, are defined at runtime. On the contrary, visual model loading in Ogre3D is performed using a specific format called *Ogre Mesh.* An Ogre mesh is a structured data file (similar to XML files) containing all the necessary information regarding the visual characteristics of a 3D model such as shape, color, material and texture information. Using mesh files, visual models can be assigned to scene nodes at the initialization stage using just a single Ogre3D function call. However, Ogre3D offers also the option of dynamically updating the shape of a virtual model at runtime the same way as in OpenGL, an option particularly useful for the implementation of virtual model deformations. Details regarding the use of dynamically updated visual shapes and real-time simulation of deformable bodies in the proposed framework are discussed in the following chapter.

5.4 ARToolkit & Ogre3D Rendering

Although the sequence of operations for real-time AR rendering are the same regardless of the rendering library, contrary to OpenGL, an implementation of AR graphics in Ogre3D did not exist at the time the presented framework was developed. Hence, in order to create AR graphics using Ogre3D and ARToolkit, the routines performing the essential rendering steps presented in previous sections were devised for the purposes of the present thesis.

5.4.1 Initialization of Camera and Virtual Scene

The initialization stage of ARToolkit is the same as in ARToolkit/OpenGL integration; a connection to the camera is established using built-in ARToolkit methods, by providing camera ID and intrinsic parameters. On the same time, an Ogre3D rendering window is initialized along with the scene

graph that includes all the virtual models required for the AR simulation session. As in OpenGL, the dimensions of the render window are equal to the dimensions of a camera frame.

5.4.2 Generating Camera Background Plane in Ogre3D

To optimize rendering operations, Ogre3D divides virtual scene components into render groups, using a flag called *RenderQueueGroupID*. Using these enumerations, groups of objects having the same *RenderQueueGroupID* are rendered in a specific order, allowing for other software events to be performed between rendering each of these groups. In addition, render queue ids are used to divide virtual elements into three layers; virtual scene objects, background objects and overlays. The last two categories are used to define 2D objects that will be added to the rendering queue right after conversion from camera space to clip space coordinates, projected either in the near or the far clipping plane. Consequently, the dimensions of background and overlay objects are directly defined in NDC space instead of camera space.

To create a 2D plane for projecting the camera frame, a rectangle with dimensions ranging from -1 to 1 across each axis is created using the Rectangle2D class of Ogre3D. Assigning this rectangle with the appropriate render queue id notifies Ogre that this rectangle is a background object:

```
// Declaration & Initialization of background rectangle
Ogre::Rectangle2D* mBackgroundRect = new Rectangle2D(true);
mBackgroundRect->setCorners(-1.0, 1.0, 1.0, -1.0);
mBackgroundRect->setRenderQueueGroup(RENDER_QUEUE_BACKGROUND);
```

In addition, using the following command, a texture buffer equal to the size of the camera frame is declared and reserved in the GPU memory.

```
// Texture pointer initialization (size equal to camera frame dimensions)
Ogre::TexturePtr* texturePtr = TextureManager::getSingleton().createManual(
	"BkgTexture",
	ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME,
	TEX_TYPE_2D, Camera->Xsize, Camera->Ysize, 0, PF_R8G8B8,
		TU_DYNAMIC_WRITE_ONLY_DISCARDABLE); // Enable dynamic update
```

Finally, the 2D background rectangle is assigned with a material pointer indicating that the rectangle will be rendered using texture data from the texture buffer:

```
// Create a pointer to a new Ogre material
Ogre::MaterialPtr material = Ogre::MaterialManager::getSingleton().create(
	"Background Material", ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME);
```

```
// Create a new material technique
Ogre::Technique* matTechnique = material->createTechnique();
// Create a new material pass
matTechnique->createPass();
// Disable depth check
material->getTechnique(0)->getPass(0)->setDepthCheckEnabled(false);
// Disable lighting
material->getTechnique(0)->getPass(0)->setLightingEnabled(false);
// Assign background texture
material->getTechnique(0)->getPass(0)->createTextureUnitState("BkgTexture");
```

In order to project camera frames on the background of the virtual scene, the hardware buffer storing background texture information must be updated with new pixel color values whenever a new camera frame is acquired. This is achieved by looping through the new frame pixels and assigning their RGBA values to the corresponding memory locations of the texture buffer. This is procedure is performed using the following set of commands:

```
Ogre::HardwarePixelBufferSharedPtr mPixelBuffer;

	// Acquire pointer to the texture buffer
	mPixelBuffer = texturePtr->getBuffer();

	// Lock the pixel buffer and get a pixel box
	mPixelBuffer->lock(Ogre::HardwareBuffer::HBL_DISCARD);
	const Ogre::PixelBox& mPixelBox = mPixelBuffer->getCurrentLock();

	pDest = static_cast<Ogre::uchar*> (mPixelBox.data);
	pSour = static_cast<Ogre::uchar*> (Camera->newFrame);

	unsigned int jmax = (unsigned)Camera->Xsize;
		unsigned int imax = (unsigned)Camera->Ysize;
			for (size_t j = 0; j < jmax; j++)
				for (size_t i = 0; i < imax; i++)
				{
					*pDest++ = *pSour++; // B value
					*pDest++ = *pSour++; // G B value
					*pDest++ = *pSour++; // R B value
					*pDest++; // A value = empty
				}

	// Unlock the pixel buffer
	mPixelBuffer->unlock();
```

5.4.3 Defining a custom virtual camera

Being a high-level graphics library mostly intended for use in 3D computer games, Ogre3D provides functionalities for the creation of virtual cameras that can be directly controlled via mouse and keyboard inputs. Responsible for controlling virtual cameras is a C++ object named *Ogre::SdkCameraMan*. However, the lower level functions for the manual definition of a virtual

camera are exposed and thus directly accessible. Using these, virtual camera initialization along with projection and view matrix definitions are achieved by the following set of commands:

```
// Initialize a new virtual camera
mCamera = Ogre::SceneMgr->createCamera("CAMERA");
// Set custom projection matrix mode and provide matrix value
mCamera->setCustomProjectionMatrix(true,ARToolkit->ProjectionMatrix);
// Set custom view matrix mode and provide matrix value
mCamera->setCustomViewMatrix(true,ARToolkit->PatternMarkerPose);
```

The last command, which sets the view matrix of the virtual camera, must be called at a per-frame basis in order to update the camera view matrix with appropriate values. As described in previous sections, the value of the view matrix is obtained from the pattern-marker tracking algorithms of ARToolkit.

5.4.4 Implementing shadows in an Ogre3D AR scene

Shadows are an important aspect in rendering a visually realistic virtual scene. Shadows provide a more tangible feel, aiding the viewer in understanding the spatial relationship between virtual objects. In an AR environment, shadows are essential for creating a realistic mixture between virtual and real objects. However, to achieve such effect, shadows of virtual objects must be cast onto real ones. Real-time shadow generation is an open field of research in computer graphics, and no universal solution exists. Several techniques exist in the bibliography, each coming with advantages and disadvantages. As is the case for VR rendering in general, visual realism of shadows is inversely proportional to rendering performance. The Ogre3D engine provides multiple shadow implementations, divided into two broad categories, *stencil shadows* and *texture shadows*. According to Ogre3D API reference manual, stencil shadows are a method by which a 'mask' is created for the screen using a feature called the stencil buffer. This mask can be used to exclude areas of the screen from subsequent renders, and thus it can be used to either include or exclude areas in shadow, while texture shadows involve rendering shadow casters from the point of view of the light into a texture, which is then projected onto shadow receivers. Regarding our AR graphics setup, we experimented with both stencil and texture shadows. The latter demonstrated better results in terms of rendering performance (less computationally expensive due to a parallel GPU implementation), utilized in our hardware setup, it produced artifacts and hence poor visual results. Stencil shadows demonstrated a slightly reduced rendering performance (approximately 20-30 fps less in scene that was rendered on an average of 500 fps). However, the Ogre3D stencil shadow technique produced a better visual outcome with no artifacts or aliasing, and hence improved visual realism.

Stencil shadows are calculated from the spatial relationship between a virtual light, the object that is casting the shadow and the object that receives the shadow. The final shape of the shadow is calculated as a projection of the casting object to the receiving object, as seen from the position of the light. Hence, the result is a binary shadow mask with distinct borders. In real life however, shadows have smoother boundaries since real lights are never produced by a single point in space. Simulation of this effect in computer graphics is called *soft shadow mapping*. To achieve *soft shadow mapping*, instead of using single light sources at the desired 3D locations, single point lights are substituted by a group of point lights, circularly arranged around the desired location of the light source. The visual outcome of the stencil shadow technique, utilized in an AR scene inside the box-trainer, is illustrated in Fig. 5.9.

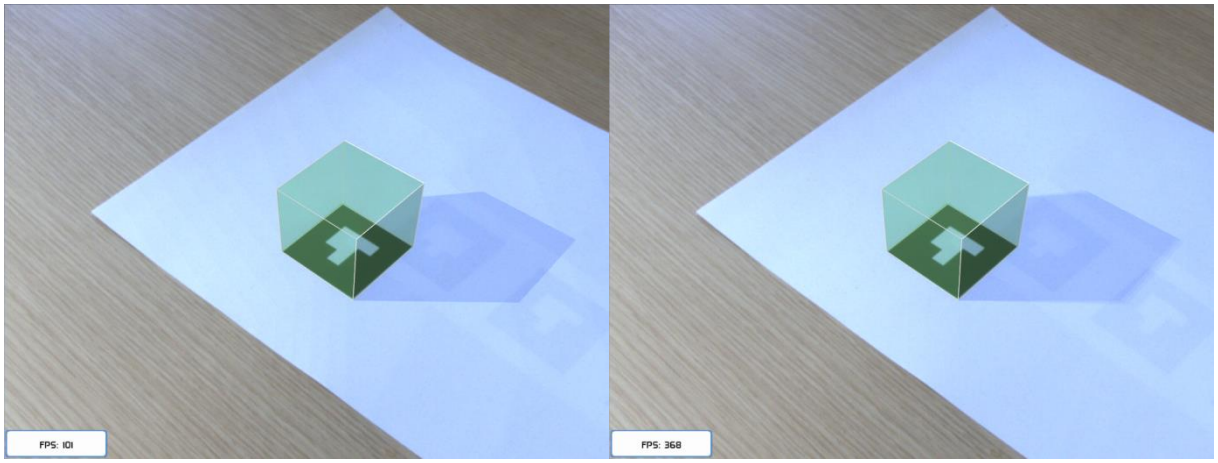


Figure 5.9 A virtual shadow projected onto the bottom plane of a real box-trainer. (a)Shadows produced using a single light source. (b)Soft shadows produced using multiple light sources.

5.4.5 The ARToolkit/Ogre3D rendering loop

The flow of operations for rendering an AR scene in the presented ARToolkit/Ogre3D setup is illustrated in Fig. 5.10. It can be noticed that the sequence of operations is similar to the sequence illustrated in Fig. 5.7, with only minor alterations of the initialization stage; the scene, including virtual models, is created outside the rendering loop and updated at a per-frame basis within the loop.

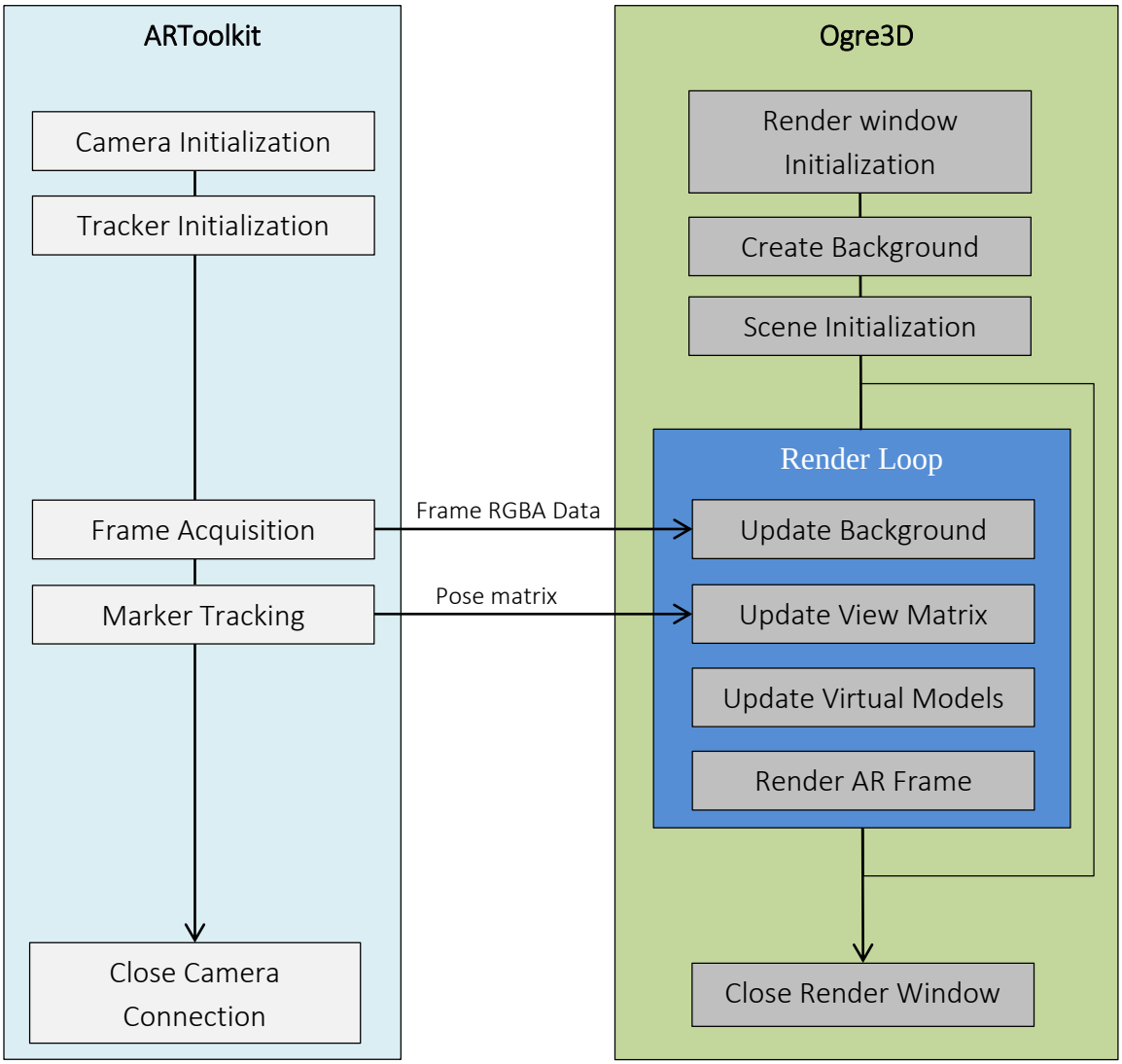


Figure 5.10 Flowchart of operations for AR scene rendering, as performed the early Ogre3D-based version of the proposed AR graphics module.

5.1.5 Shadow Casting and Occlusion Handling for Real Objects

Adding virtual shadows in an AR environment is a more complicated procedure compared to pure VR, since virtual objects need to cast shadows onto physical objects of the real world scene. To achieve this effect in our setup, a 'ghost object' of every real object is created and added to the virtual world. These 'ghost' objects are defined by a 3D shape, and assigned with a colorless (transparent) material that however can receive shadows. For instance, the bottom plane of the box-trainer shown in Fig. 5.9 is in essence a virtual 3D plane introduced to the scene as any other virtual object, assigned however with the appropriate colorless material. The 'ghost' objects material is a GLSL script, written in an Ogre3D material format:

```
material GhostObjectMaterial
{
	receive_shadows on // Receive shadows (general)
	transparency_casts_shadows on // Receive shadows (transparent)
	technique TransparentShadowTechnique
	{
		pass TransShadowTex
		{	// Define shadow color
			ambient 0.698039 0.698039 0.698039 0
			diffuse 0.698039 0.698039 0.698039 0
			specular 0.898039 0.898039 0.898039 0 20
			scene_blend src_alpha dest_alpha
		}
	}
}
```

Another significant issue arising from the coexistence of virtual and real objects within the same scene is the *occlusion problem*. This term describes a fundamental problem of AR rendering: In a real world scene, objects located closer to the viewer obscure objects located behind them. In AR though, VR elements are by default superimposed on top of the camera frame. Consequently, virtual objects appear to be in front of the real ones regardless of their relative poses in the 3D space. This is a potential problem that can produce undesired effects in the final AR rendering outcome, confusing the viewer and significantly affecting the visual realism an AR application. This is a known problem of AR and several solutions have been proposed, most of which however aim to deal with uncontrolled environments where real world objects have undefined shapes and poses. The most widely known solutions involve utilization of stereoscopic cameras that produces depth maps of the real world, and use depth data to handle occlusion of virtual objects from real objects [169, 170].

Within the controlled environment of the box-trainer however, the only real objects that could potentially obscure virtual elements of the scene are the laparoscopic tools. Since information regarding the pose of the tools with respect to the scene is acquired using instrument tracking techniques, the solution implemented in our setup to handle with the occlusion problem is relatively simple and does not require utilization of stereo vision apparatus. Similar to the cast shadowing technique described earlier, a colorless representation of each laparoscopic instrument is added to the AR scene, using pose information acquired by instrument tracking techniques. The idea behind this method is relatively simple; shape and pose information of real scene objects are included in the depth buffer but excluded from the color buffer. Consequently, during the depth checking process (automatically performed in the rendering process), parts of virtual objects obscured by real ones will fail the depth check testing and consequently, not rendered in the final 2D image of the scene. Figure 5.11 illustrates visual results of the proposed occlusion handling technique.

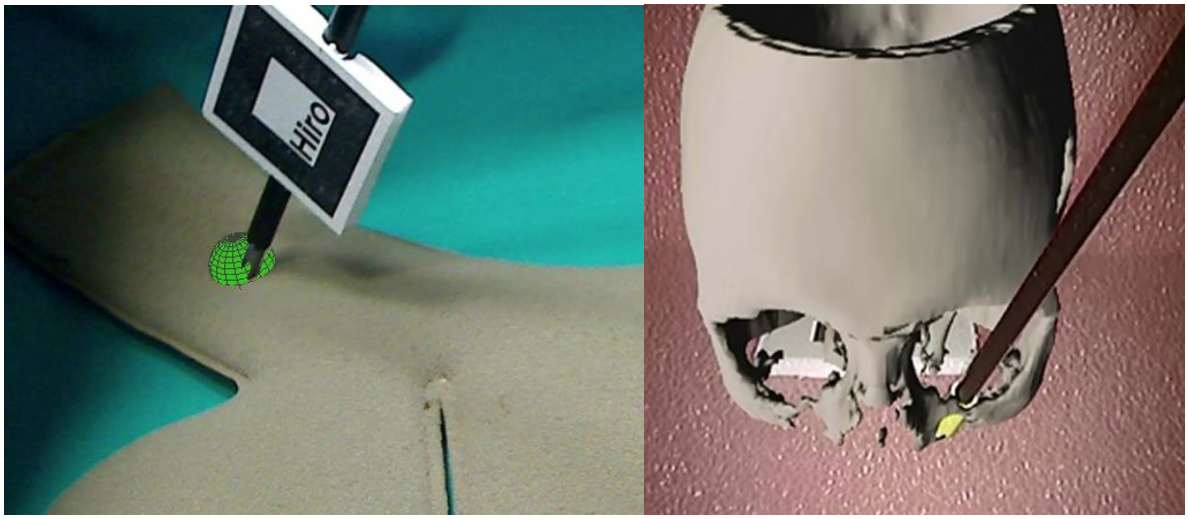


Figure 5.11 Visual illustration of the occlusion handling technique implemented in the proposed framework. (a) A real laparoscopic grasper obscures a virtual sphere. (b) A laparoscopic instrument obscures the 3D model of a human skull.

The practical implementation of the presented occlusion handling method in Ogre3D is achieved with a minor modification of the *render queue* operations. Specifically, instead of the default *render queue groups* available in Ogre3D, an additional group is defined to characterize the colorless representations of laparoscopic instruments. Ogre3D provides the option of creating a *render queue listener* using the abstract class *RenderQueueListener.* The *renderQueueStarted* function of this listener is registered into the render queue operations of Ogre3D pipeline, executed at a per-frame basis. Instructions regarding the way non-default queue groups will be rendered can be included in this function. In our setup, the following code snippet provides the instructions for rendering real object representations, disabling writing in the color buffer and hence producing depth buffer masks[7]:

```
// Rendering instructions for real objects
if (queueGroupId == Ogre::RENDER_QUEUE_REAL_OBJECT){
        Ogre::RenderSystem * rs = Ogre::Root::getSingleton().getRenderSystem();
        // Disable writing to color buffer
        rs->_setColourBufferWriteEnabled(false, false, false, false);
        rs->setStencilCheckEnabled(true);
        rs->setStencilBufferParams(Ogre::CMPF_EQUAL,0x1,0xFFFFFFFF,
       Ogre::SOP_KEEP,Ogre::SOP_KEEP,Ogre::SOP_KEEP,false);
}
```

[7] The occlusion handling method presented in Section 5.1.5 can also be applied in OpenGL, using the commands *glColorMask(false)* and *glColorMask(true)* before and after defining the pose and shape of each laparoscopic instrument.

5.1.6 Multimarker tracking

Previous sections discussed the practical implementation of AR graphics within the proposed surgical simulation framework. A significant piece of information regarding the presented implementations is the pose of the pattern marker with respect to the camera. This information is utilized for the construction of a view matrix, essential for accurate mixture of VR graphics and real camera images. Acquiring the marker pose is achieved using ARToolkit pattern marker tracking routines. In our setup, the camera is moving and hence the pattern marker needs to be tracked at each frame. However, since laparoscopic instruments might obscure the pattern marker and hence cause failure of the tracking algorithms, in the final version of the proposed software engine we utilized a multimarker tracking functionality provided by the professional version of ARToolkit. Using multiple markers at predefined relative positions, this ARToolkit functionality allows pose tracking even if some of these markers are occluded.

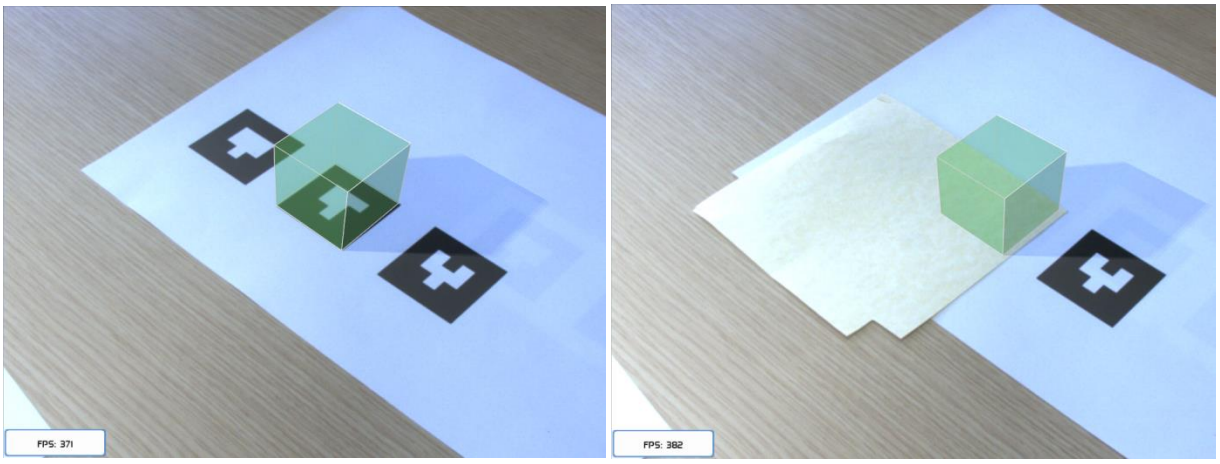


Figure 5.12 (a) Rendering an AR scene using a multiple markers. (b) Occlusion of markers does not affect pose tracking as long as one of these markers remains visible.

Chapter 6

Integration of Real-Time Physics

Physics-based modeling is a feature of instrumental importance in a surgical simulation framework. Every discussion on the use of real-time physics in VR/AR applications should begin with a reference on Sir Isaac Newton. The Newton's laws of motion are utilized in every VR application involving physical simulation of virtual objects kinematics. In surgical simulation, realistic interactions between virtual objects and laparoscopic instruments are achieved as a result of Newton's laws application, integrated in the simulation process with the use of advanced numerical methods. In this thesis, we present a surgical simulation framework that involves real-time simulation of physics-based interactions in a laparoscopic box-trainer environment enhanced with the superimposition of AR graphics. A core component of this framework is a physics engine built upon Bullet Dynamics library. This module is responsible for adding collision detection and rigid/soft body dynamics calculations to an AR environment. This Chapter describes the building blocks, the architecture and the principle of operation behind the physics engine of the proposed simulation framework.

6.1 Introduction

Previous Chapter focused solely on rendering, illustrating the practical implementation of AR visualizations, achieved using a graphics engine built upon ARToolkit/Ogre3D integration. In applications involving real-time physics however as is laparoscopic simulation, rendering is a dynamics driven process where the pose or even the shape (in cases of deformable bodies) of virtual models is acquired via the simulation of the laws of physics. Any surgical simulator involves, to some extent, the simulation of interactions between surgical instrument and virtual models. Such models can vary from simple virtual pegs to complicated anatomical structures. Responsible for simulating the aforementioned interactions is a physics engine that computes the dynamics of simulated rigid/soft bodies, performs collision detection, computes collision response and solves the constraint equations required for simulation of joints/constraints. When it comes to real-time applications such a surgical simulator, the most important aspects regarding physics-based computations are accuracy and performance. Profoundly, both aspects depend on the efficient implementation of the physics engine responsible for performing such computations.

Development and optimization of a physics engine is a tremendous task, requiring deep background knowledge of physics, implementation of optimized numerical methods as well as development of advanced software engineering solutions. For this reasons, commercial applications such as modern computer games are built upon already existing physics engines, modify them according to suit specific needs, instead of creating application-dedicated engines from scratch. To integrate real-time physics in the proposed framework, we also focused on developing a software module based on an existing physics engine, responsible for providing rigid/soft body dynamics. Integration of this module into the simulation loop of our framework would allow implementation of training scenarios that provide real-time interaction between surgical instruments and virtual elements of the training scene.

6.2 Physics Engine Selection

Nowadays, a wide collection of open-source and closed-source physics engines is available. In our pre-selection of potential candidates for integration in our propose framework, we excluded some very widely used but relatively out-aged engines such as ODE [171], as well as engines under experimental development at the time, such as SOFA [172]. Following existing reviews and comparison studies [173–175] we targeted on four specific engines, all under active development and support: NVidia PhysX [176], Havok Physics [177], Bullet [160] and Newton Dynamics [178]. NVIDIA PhysX (formerly Novodex, created by AGEA), is distributed as closed source under the terms of its own EULA, widely adopted by the PC gaming industry (used at the time in more than 200 game releases). Similar to PhysX, Havok is a commercial physics engine with its own end-user license agreement (EULA), used at the time in more than 150 commercial game releases. Newton is an open-source engine distributed under the Zlib license, utilizing a deterministic solver

approach promising increased computational accuracy [175]. Bullet or Bullet Collision Detection and Rigid/Soft Body Dynamics Library is a free and open source physics toolkit distributed as a successor of ODE under the Zlib license, utilizing impulse-based dynamics with LCP iterative solving. According to comparison studies [173], [174] and [175], none of these engines outperformed the rest since each indicated certain strong points and weaknesses. All however demonstrated sufficient accuracy and performance for utilization in a surgical simulator [173]. PhysX and Havok provided the highest speed of computations as a result of optimized exploitation of multi-parallel GPU architecture, Newton provided the highest accuracy with an expense of slower performance, while Bullet provided the highest reliability in constraint solving and advanced collision/friction benchmarks, demonstrating however medium computation performance [179].

Table 6.1 Comparison of four physics engines considered for integration in the proposed simulation framework.

	PhysX	Havok	Bullet	Newton
Open-source	N	N	Y	Y
GPU Support	Y	Y	E	Y
Multithreading	Y	Y	Y	Y
Rigid Bodies	Y	Y	Y	Y
Soft Bodies	Y	Y	Y	N
Constraints	Y	Y	Y	Y
Collisions	Y	Y	Y	Y
Y = Yes, N = No, E = Experimental				

The final decision was based on the features provided by each engine, as summarized in Table 6.1. PhysX and Havok provided all the perquisites for integration in a surgical simulation. However, an important disadvantage of both is the closed-source distribution that does not provide flexibility in core algorithm modifications. Newton on the other hand, is an open-source solution but lacks support of soft body dynamics which are of particular importance in surgical simulation. Finally, Bullet included all the essential features of a physics engine such as rigid/soft body dynamics, constraint solving, collision detection, supported hardware acceleration and multithreading, providing at the same time flexibility in modifications due to its open-source distribution. Consequently, the final decision was to integrate Bullet into our proposed framework.

The Bullet Physics Library, created by Erin Coumans, is a C/C++ library distributed under the Zlib license, supporting various operating systems. Bullet supports continuous and discrete collision detection using impulse-based methods for collision response, allows integration of simple geometric shapes as well as convex and concave meshes, supports multiple constraint types such as point constraints, hinges, sliders and 6 d.o.f. constraints. At the time the presented framework

was developed, the last stable version of the Bullet library was version 2.80 which we integrated into our system.

6.3 Integration of Bullet Dynamics Engine into the presented Framework

Chapter 5 provided a detailed description of the sequence of operations required for achieving AR visualizations in a box-trainer environment, implemented into a loop that utilizes ARToolkit for pattern marker tracking and Ogre3D for rendering. The rendering loop illustrated in this chapter is rather simple compared to the simulation loop of a training task, as described in Chapter 4. Up to this point, we have discussed the implementation of AR graphics and the creation of AR-based training scenarios, using virtual models introduced at predefined locations of the training scene, not taking into account any interaction with the surgical instruments whatsoever. However, training scenarios designed using the final version of the proposed framework follow a different approach; trainees can actually interact with virtual objects, applying actions such as grasping and lifting, pushing, cutting etc.

To achieve such interactions, the pose of each virtual object involved in the training scene is calculated as a result of physics simulation and collision detection performed within the physics engine module. Consequently, the sequence of operations for rendering a scene is also different. Before a new frame is rendered, the physics module performs computations that provide new poses of virtual objects, and this information is fed to the graphics module so that the visual representations of these objects are updated accordingly. Prior to going into details regarding specific issues that we dealt with towards the creation of laparoscopic simulation training scenarios using real-time physics, it is essential to provide some general information regarding the operating principles of Bullet as utilized in the presented framework.

6.3.1 The Dynamics World and the Physics Pipeline

In Bullet as in most physics libraries, similar to the virtual world required for the creation of VR applications, a dynamics world called *btDiscreteDynamicsWorld* must to be instantiated. The definition of *btDiscreteDynamicsWorld* includes a central coordinate system and a force of gravity. Gravity is defined by a 3D direction vector and an acceleration value. Each virtual object involved in the simulation is added to the dynamics world by defining its physical characteristics (shape, mass, restitution etc.) and its pose with respect to the central coordinate system (*world transform*). By default, Bullet assumes units to be in meters and time in seconds. This is however implementation dependent, since the developer is allowed to change the default units, being responsible however for maintaining a physically correct unit system; if for instance units are set to mm, objects shapes must be provided in mm, while the acceleration of gravity must be set at 9810 mm/s^2 otherwise physics simulation will appear in slow motion. In the presented framework, a restriction regarding the desired unit system arises from the use of ARToolkit for

pattern marker tracking. ARToolkit provides distance measurements in mm based on the physical dimensions of pattern markers. Consequently, the ARToolkit/Ogre3D graphics engine presented in the previous chapter is designed to express dimensions in mm. To maintain uniformity between graphics and physics, we define mm as the default unit of distance in our Bullet implementation. In addition the central coordinate space of the *btDiscreteDynamicsWorld* is set at the same location and orientation with respect to the box-trainer environment as the central coordinate system of the graphics world described in the previous Chapter, providing a direct connection between the graphics and the physics world. This allows a single definition of virtual objects' poses; the same transformation/pose matrices can be utilized for defining the location of both visual models and physics objects with respect to the box-trainer environment.

As depicted by the sequence of simulation loop operations, illustrated in Section 4.3.3 of Chapter 4, physics and dynamics computations are performed before each frame is rendered, following the acquisition of a new camera image and instruments' pose. Bullet performs these computations in large variable time steps, divided into smaller fixed time steps. At each fixed time step, the pose of a physics object is computed as a result of gravity, collisions with other objects and forces applied to it due to certain constraints. The exact sequence of operations for deriving with updated positions of physics objects is shown in Fig. 6.1, illustrating the entire physics pipeline computation and the data structure types included in the dynamics world of Bullet.

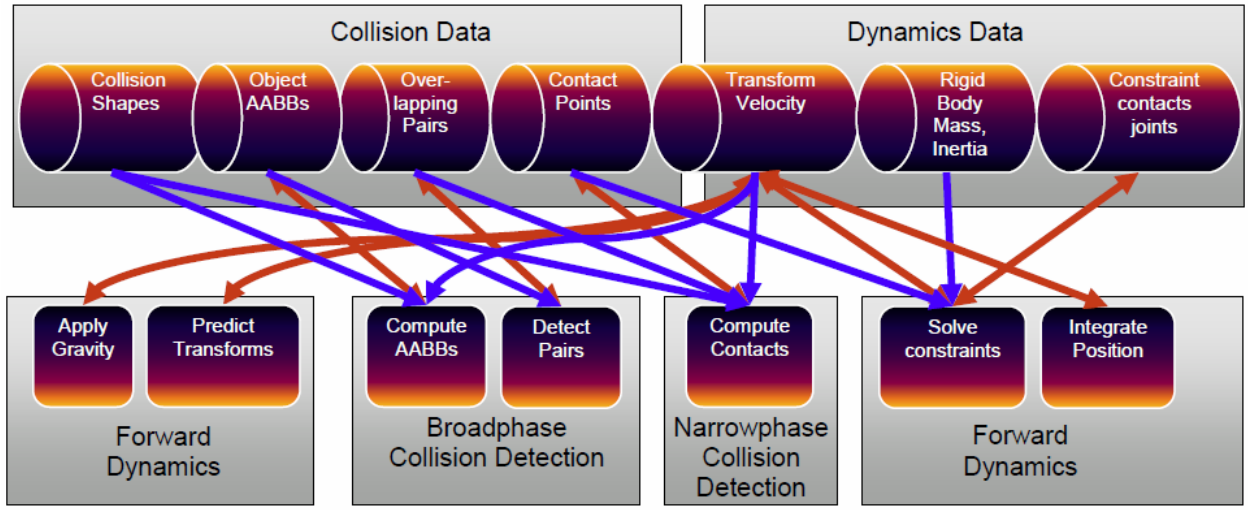


Figure 6.1 The Bullet Physics Engine Pipeline [180].

As illustrated in this figure, the physics pipeline begins with applying gravity and ends by position integration, which provides the resulting position of objects within the physics world. The sequence of operations illustrated in this figure is performed on each fixed time step. A simulation step is performed by calling *btDynamicsWorld::stepSimulation()* function of Bullet, passing the following three parameters:

- *TotalTimeStep*: The amount of time to step the simulation by. In real-time applications as the one presented in this thesis, the *total time step* has a variable value that must correspond to the amount of time passed between two consecutive calls of *stepSimulation*. To measure this value, a C++ software timer is utilized in our simulation loop providing the time lapse between each *stepSimulation* call in milliseconds.

- *MaxNumOfSubsteps*: Bullet divides the total time step into smaller fixed time steps. This is pivotally important for framerate independence since the *TotalTimeStep* is variable. The *MaxNumOfSubsteps* parameter defines the maximum number of sub-steps that the total time step can be divided by.

- *FixedTimeStep*: This parameter defines the internal *time step* of Bullet, required in order to keep the actual length of ticks constant. By default, Bullet operates an internal fixed *time step* of 60 Hertz, which however can be changed to suit the specific needs of each application. Decreasing the *FixedTimeStep* value "increases" the resolution of the simulation. However, since each simulation step requires a large number of operations (Fig. 6.1), setting a very small *FixedTimeStep* also increases the processing time required by Bullet and consequently, reduces the framerate performance of the application.

As a general rule of thumb, the *FixedTimeStep* multiplied by the *MaxNumOfSubsteps* should be less than the *TotalTimeStep*. In our setup we used a *FixedTimeStep* of 240 Hertz and since we aimed to maintain frame-rates of at least 30 Hertz, we used a maximum of six sub-steps (*MaxNumOfSubsteps*).

6.3.2 Modeling of Virtual Objects Physical Representation

It is a common practice in interactive computer graphics applications to utilize two different representations of virtual objects, one defining the visual characteristics of an object and one defining the object's physical properties and behavior. The visual representation, as thoroughly explained in Chapters 3 and 5, includes the definition of an object's shape as well as additional visual attributes such as color and texture. The behavioral model on the other hand includes the definition of an object's collision shape as well as additional physical properties such as mass and restitution for rigid objects or elasticity and stiffness for deformable objects.

In the presented framework, physics simulation was integrated after the ARToolkit/Ogre3D implementation of the graphics engine. Hence, to derive with a common design pattern, the physics module was built on the same object-oriented approach as the AR graphics module. Similar to the *SceneNodes* of Ogre3D used to define visual models, physics objects in Bullet are implemented as C++ classes that provide all the essential properties and methods for defining a virtual objects behavioral model, the most important of which are its collision shape and its pose

with respect to the origin of the dynamics world. In Bullet, these objects are called *bodies,* characterized by a general type (*btRigidBody* or *btSoftBody*) which indicates whether these are rigid or deformable objects. In the presented framework, the visual and physical representations of virtual objects are combined into a single higher level structure including all the essential classes and their properties, as illustrated in Table 6.2. As shown in this table, every virtual element of the AR training scene is effectively two C++ objects combined into a single entity. The first object holds information regarding the visual properties of the virtual object, hereby referred to as visual model. The second object holds information regarding the physical properties of a virtual object, hereby referred to as physics body or body.

Table 6.2 Structure of a virtual object

Visual model	Physics Body
OgreSceneNode	*btRigidBody* or *btSoftBody*
- *Visual shape* - *Pose* - *Color/Material* - *Additional properties*	- *Collision shape* - *Pose* - *Additional properties*

6.3.2.1 Collision shapes

Ideally, the collision shape of a virtual object's physical representation would be identical to its visual model. In physics simulation however, the time required to perform a physics simulation step is proportional to the levels of detail of the collision shapes involved in the simulation. Bullet supports a large variety of default collision shapes, also allowing the option of designing custom shapes based on the specific needs of each application. Figure 6.2 illustrates a diagram for selecting a collision shape, taken from the official user manual of Bullet.

The connection between levels of detail and speed of calculations can be explained by observing the sequence of operations taking place in Bullet during a simulation step, illustrated in Fig. 6.2. As this figure depicts, a core operation of the physics pipeline is collision detection, which is performed in two phases; the *broadphase* and the *narrowphase*. As demonstrated in Fig. 6.3, the *broadphase* identifies collisions between two bodies using Axis Aligned Bounding Boxes (AABBs). Broadphase collision detection is a relatively simple operation since it only involves comparison between box-shaped collision volumes, and it is performed in order to avoid costly collision checking for bodies that are far from each other. In the second phase however, called the *narrowphase*, collision pairs are analyzed using computationally expensive algorithms that take into account the exact collision bodies geometries in order to derive with accurate results regarding the contact points of colliding bodies. Hence, utilizing collision shapes of high levels of

detail increase the total number of computations performed during the narrowphase, consequently resulting in a decrease of the physics simulation framerate.

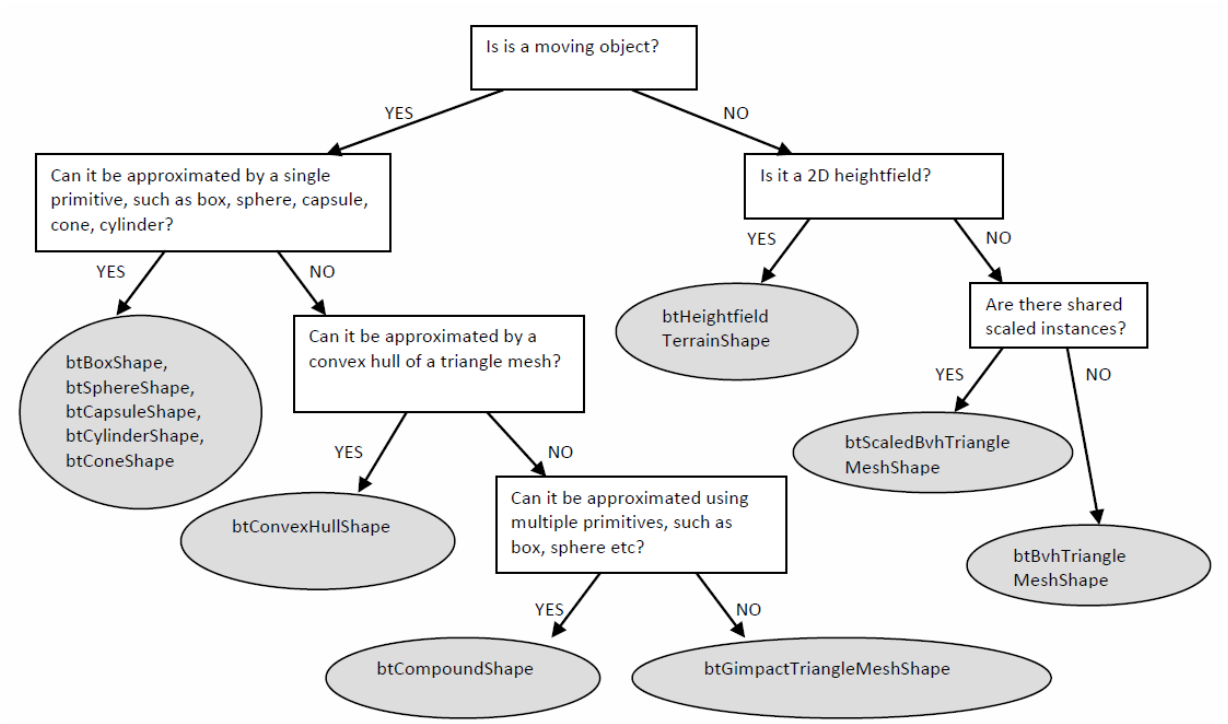


Figure 6.2 Bullet collision shape selection diagram, according to the type and behavior of each virtual object [180]

In a real-time application such as a surgical simulator, maintaining a high frame rate is extremely important. Imagine a surgical simulator where the trainee would perform an action with the surgical instrument, as for instance closing the instrument's handle to cut a virtual artery, and the result of this action would be applied in the VR artery with a time delay. Such delays would produce a sense of a non-responsive simulation, making it practically impossible for trainees to complete training scenarios. To avoid such effects, surgical simulators are designed to maintain a minimum frame rate of 30 fps that is the lowest limit for an application to be considered real-time.

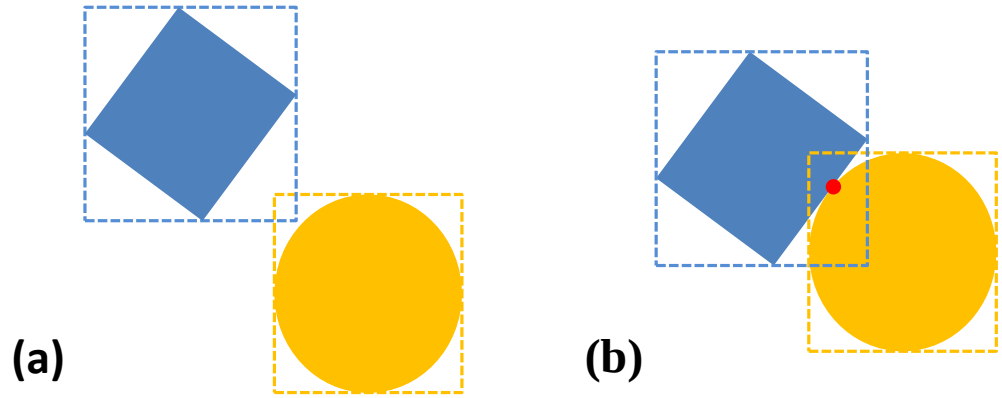


Figure 6.3 The two phases of collision detection. (a) Broadphase collision detection using AABBs (b) Narrowphase collision detection using collision shape geometries.

As mentioned in previous Chapters, high resolution virtual graphics are a bottleneck for the rendering process, forcing game developers to a compromise between visual realism and speed optimization. This logic also implies in physics computations. Performing a physics simulation step is the most computationally demanding task of the simulation loop, and hence a compromise between detail and performance is required when designing a virtual object's behavioral model. During the development of the presented physics module, a large period of time was invested on testing several variations of collision shapes for of the virtual objects involved in our training scenarios. The purpose was to conclude on collision shapes that behaved as expected, maintaining on the same time a relatively low collision shape detail in order to ensure simulation frame rates over the 30 fps threshold.

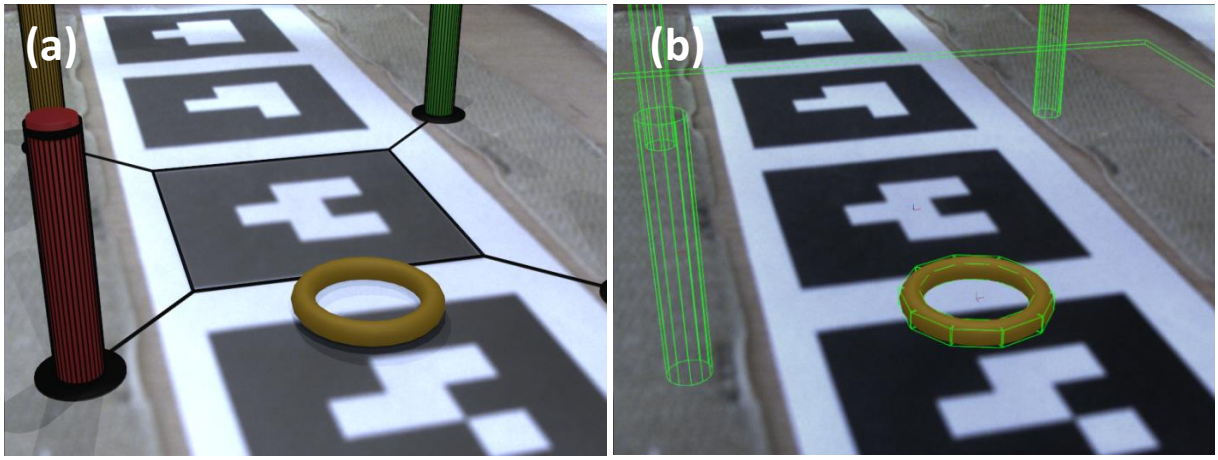


Figure 6.4 (a) An AR training task implemented in the proposed framework. (b) Low resolution collision shapes, maintaining an approximation of the original virtual objects' shapes, allow realistic simulation of physical behavior at a lower computational cost.

Figure 6.4 illustrates an example, taken from an AR training task implemented in the presented simulation framework. This task requires trainees to lift virtual torus-shaped objects and place them at virtual cylinders of the same color. Both the cylinders and the torus-shaped objects are highly detailed virtual models, consisting of thousands of vertices. The corresponding collision shapes however, as illustrated in Fig. 6.4b, are defined as low resolution primitive shapes, providing an approximation of the objects' visual shapes. This allows realistic simulation regarding the physical behavior of these objects without increasing the computational cost of narrowphase collision detection.

6.3.2.2 Updating visual models using motion states

As discussed in Section 6.3.2, visual representations of virtual object involved in a training scene are Ogre3D scene nodes assigned with visual characteristics (3D shape, color, material etc.). The pose of these nodes is expressed with respect to the central reference frame of the box-trainer environment. Additionally, in Section 6.3.1 of the current Chapter we mentioned that the pose of physics bodies belonging to the simulation scene is also expressed with respect to the central reference frame. Finally, we have already discussed that the pose of visual models is updated following a physics simulation step which provides new information for each virtual object involved in a scene. Following a simple approach, visual models would be updated by looping through their corresponding physics bodies and obtaining new pose information. However, since framerate optimization is important in our setup, costly loop operations should be avoided as much as possible.

With this in mind, we utilized a feature of Bullet called *motion state* for updating visual models after each physics simulation step. A motion state is a tool used by Bullet to define the pose of a physics body at the initialization stage, and update the pose of this body after each simulation step. Each body introduced to the physics world is assigned a motion state, which is implemented as a C++ class called *btMotionState.* This class includes a *setWorldTransform* and a *getWorldTransform* function. The first is called upon initialization to define the initial pose of a physics body while the latter is called after each simulation step, only if the specific body has been moved during this step.

In the proposed engine, an inherited version of the default *btMotionState* class is used, the *setWorldTransform* and *getWorldTransform* functions of which are overridden. Passing a pointer to the corresponding Ogre scene node of each physics body and consequently on the memory structure holding information regarding the node's pose with respect to the training scene, the latter can be updated each time the *getWorldTransform* of the motion state is called. The C++ code of this custom motion state is provided in the following code snippet:

```
	//Custom motion state to automatically update ogre nodes from a physics body's
	class arOgreBulletMotionState : public btMotionState {
	public:
	arOgreBulletMotionState (const btTransform &initialpos, Ogre::SceneNode *node) {
		mVisualModel = node;
		mPose = initialpos;
	}

	virtual ~OgreBulletMotionState() {
	}

	void setNode(Ogre::SceneNode *node) {
		mVisualModel = node;
	}

	virtual void getWorldTransform(btTransform &worldTrans) const {
		worldTrans = mPose;
	}

	virtual void setWorldTransform(const btTransform &worldTrans) {
		if(NULL == mVisualModel)
		return; // silently return before we set a node
		btQuaternion rot = worldTrans.getRotation();
		mVisualModel ->setOrientation(rot.w(), rot.x(), rot.y(), rot.z());
		btVector3 pos = worldTrans.getOrigin();
		mVisualModel ->setPosition(pos.x(), pos.y(), pos.z());
	}

	protected:
		Ogre::SceneNode *mVisualModel;
		btTransform mPose;
};
```

It is important to note that motion states are not utilized by Bullet in cases of kinematic bodies. Consequently, the visual models of kinematic bodies involved in our setup need to be manually updated at each frame, using inputs from the instrument tracking module. In the presented framework, kinematic bodies where used in the simulation of laparoscopic instruments physical behavior, which is described in the following sections.

6.3.2.3 Collision Types

Except for shape, both rigid and soft bodies in Bullet are characterized by a collision type. The type of a physics body describes its "nature", indicating if a body reacts to collisions if its pose is affected by external forces. Bullet supports three collision types, *dynamic*, *static* and *kinematic* bodies, each defined by a *collision flag*. Dynamic bodies are the bodies affected by forces such as gravitational pull or forces caused by collision with other bodies. Static bodies on the other hand, are those not affected by any force, remaining fixed throughout the simulation. Finally kinematic bodies are movable objects that do not react to collisions or external forces. The pose of these objects is not algorithmically calculated but directly provided as an input to the physics simulation step. It is

common in computer games to use kinematic bodies for virtual objects that are controlled by user inputs, such as a car in a racing game or an airplane in a flight simulator game.

In the proposed framework, dynamic bodies were used to simulate movable components of a training scenario such as the torus-shaped object of Fig. 6.4, while static bodies were used to represent rigid parts of a training environment such as the colored cylinders of Fig. 6.4.Finally kinematic bodies where used for simulation of user controlled components of the training environment, which are the laparoscopic instruments. In addition to the default collision types of Bullet, we created an extra type called *hybrid*, which was assigned to moving bodies that should collide with kinematic as well as dynamic bodies, but not with static ones. For reasons described in later section of this chapter, this collision type was essential for the adding a physical representation of the bottom plane of the box-trainer and some other static objects involved in training scenarios. Table 6.3 presents the collision types utilized in the presented physics module, the corresponding collision flags and where these types where used.

Table 6.3 The Bullet collision types utilized in the presented physics engine module

Type	*Collision Flag*	*Collides with*	*Usage*
Dynamic	1	1 &2 & 3 & 4	Pegs, rings etc.
Kinematic	2	1	Instruments
Static	3	1	Pegboard etc.
Hybrid	4	1 & 4	Bottom plane

6.3.3 Simulating the physical behavior of laparoscopic instruments

In the presented framework, we have created and evaluated the construct validity of AR-based training tasks using four different types of laparoscopic instruments, demonstrated in Fig. 6.5. The first is a simple touch instrument with a cone-shaped tooltip, used for the implementation of simple instrument navigation tasks. The other three are a typical laparoscopic instruments utilized in most of the commercial laparoscopic simulation platforms, representing the real instruments used in surgical operations. These are the laparoscopic graspers (Fig. 6.5b), scissors (Fig. 6.5c) and clip applicators (Figs. 6.5d).

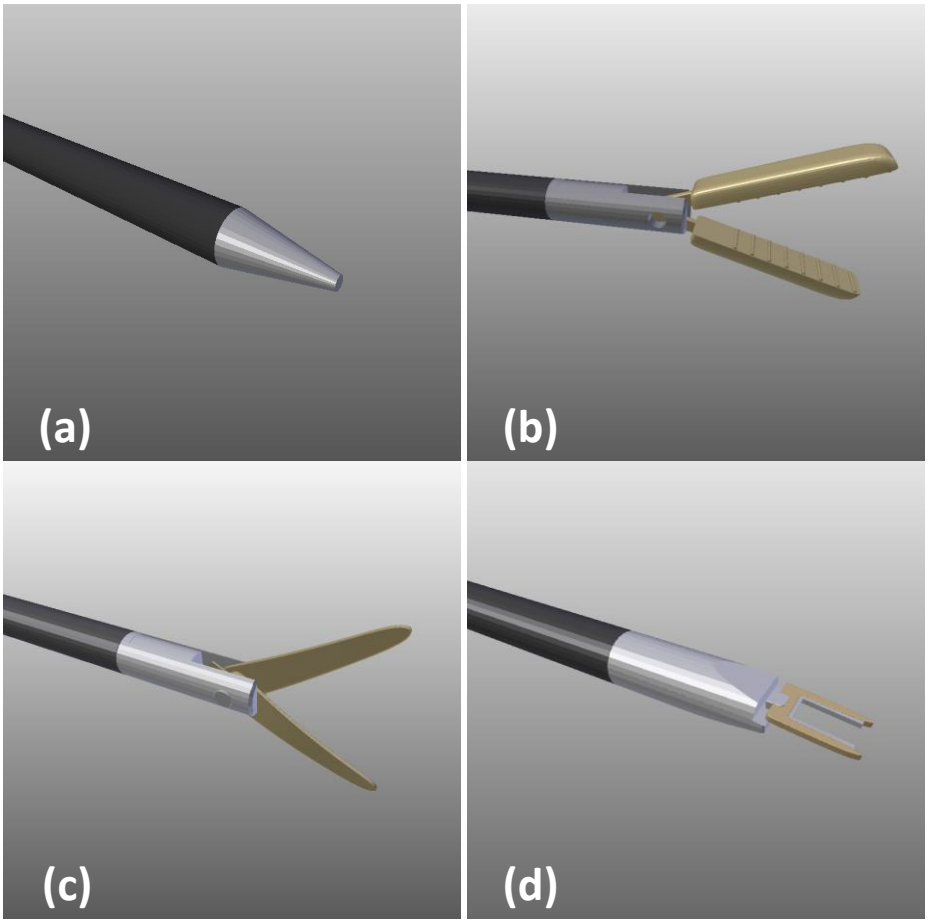


Figure 6.5 The four types of laparoscopic instruments supported by the presented surgical simulation framework. (a) Cone-shaped instrument used for touching virtual objects (b) grasper (c) scissors and (c) clip applicator

To allow the creation of training scenarios using the aforementioned instruments, the physics module of the presented framework should be able to realistically simulate their behavior during interactions with virtual objects. As with any other object involved in a training scenario, the physical characteristics of instruments must be accurately described in order to achieve realistic simulation of their physical behavior.

Integration of real-time physics in the proposed surgical simulation framework was implemented at the final stage of the development process. Prior to this stage, we had developed AR-based training tasks that did not require interaction between surgical instruments and virtual objects. Implementation of such tasks was achieved using pattern marker [157] or image processing [156] instrument tracking techniques. The first technique calculates the pose of an instrument by tracking a pattern marker attached on the instrument's shaft, as illustrated in Fig. 6.6a. The second technique estimates the 3D pose of laparoscopic instruments via an inverse projection of their edges, using a combined Hough-Kalman line detection algorithm to detect edges in the camera image (Fig. 6.6b). As discussed in Chapters 7 and 8, these tracking techniques provide the pose of

laparoscopic instruments with respect to the central reference frame of the training environment. This information is adequate for implementing simple tasks that only require knowledge of the tooltip position. In addition, since these tasks required using simple instruments of cylindrical shape (or instruments with the tooltip locked at a closed position), knowledge of their pose was adequate for solving the augmented reality occlusion issue.

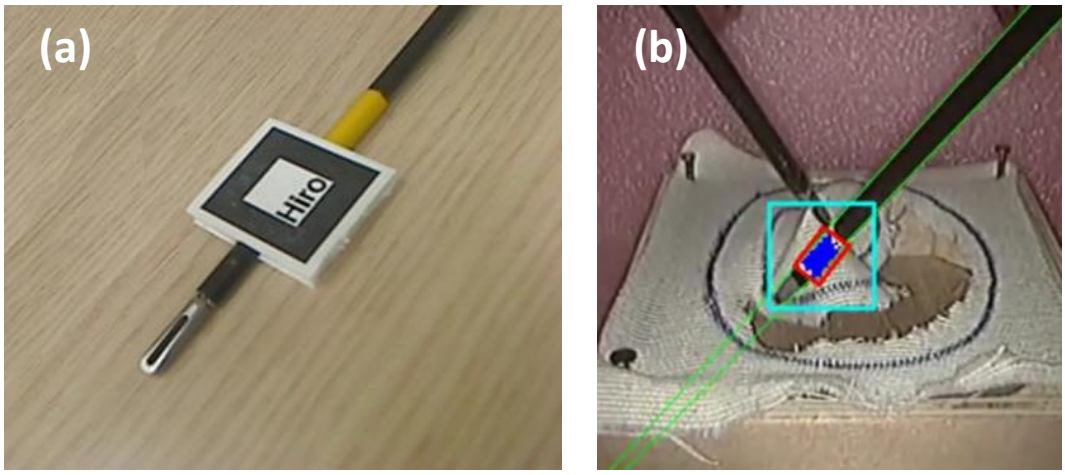


Figure 6.6 (a) Pose tracking of laparoscopic instruments using pattern markers. (b) Pose tracking of laparoscopic instruments using image processing algorithms.

The ultimate goal of the presented thesis however, was the creation of an AR-based surgical simulation framework that would provide a basis for the implementation of more advanced training scenarios, involving complex interactions between instruments and virtual objects such as grabbing, lifting, cutting etc. Towards the fulfillment of this goal, we concluded on using virtual representations of laparoscopic instruments, controlled however by manipulation of real instruments. This would be achieved by integrating alternative instrument tracking techniques that would provide more information regarding such as the tooltip opening angle and the instrument shaft rotation.

6.3.3.1 Acquisition of instruments' pose and state

At its final form, our framework utilized an array of sensors attached on each instrument, providing real-time instrument's pose and state information. This complete sensory setup and its operating principle is thoroughly discussed in two publications [76, 155] presented in Chapters 9 and 10 of this dissertation respectively. However, to explain how physical simulation of laparoscopic instruments is achieved, it is essential to give some details regarding the operating principle of the instrument tracking module and the outputs that this module provides.

This module is a C++ object designed to communicate with sensors attached on the instruments, returning relevant information at a per-frame basis. In order allow experimentation with different tracking techniques in the future, this module is designed to operate as a black-box, triggered by

a single input and returning a single output regardless of the training scenario and the types of instruments involved. The input is a single trigger call, ordering the module to acquire new measurements. The output is a structure containing measurements from three different sensors, providing the following information:

1. Instrument pose (M_I): Electromagnetic sensors attached on the handles provide the pose of laparoscopic instruments with respect to the central coordinate system of the training environment, in the form of a 4x4 homogeneous transformation matrix.

2. Tooltip opening angle (θ_t): IR proximity sensors attached on the handles provide a scalar value corresponding to the opening angle of the instrument's tooltip.

3. Shaft rotation (θ_s): Rotary encoders attached between the handles and the instrument's shaft provides a scalar value corresponding to the rotation of the shaft around its centerline.

6.3.3.2 Instrument modeling

A real laparoscopic instrument consists of several small and large mechanical components. Ideally, fully describing the physical characteristics of such instrument would require a detailed modeling each of its components. In surgical simulation training however, only a portion of the instrument enters the box-trainer environment during practicing, and hence involved in the simulation process; the instrument's shaft and the tooltip. To reduce the total number of involved physics bodies in our simulation, and to simplify the overall instrument modeling process, modeling of surgical instruments in the proposed framework focuses only on modeling the instrument's shaft and tooltip. And although the tooltip varies between different instruments, the three types of instruments supported by the presented simulation platform have an almost identical structure; a cylindrical shaft connected with the two parts of the tooltip, which have the same shape, facing however in opposite directions.

In order to replicate this structure in the presented framework, instead of using a single entity, instruments are modeled as a group of three physics bodies connected to each other. These bodies are linked to their equivalent visual models, implementing the motion state technique described earlier. An example of such structure is shown in Figure 6.7, illustrating a virtual laparoscopic grasper. The grasper shaft is highlighted in yellow, while the two parts of the tooltip are highlighted in red. As every 3D object, each of these parts has its own local coordinate system, used to define its geometry and its spatial relationship with the central reference frame.

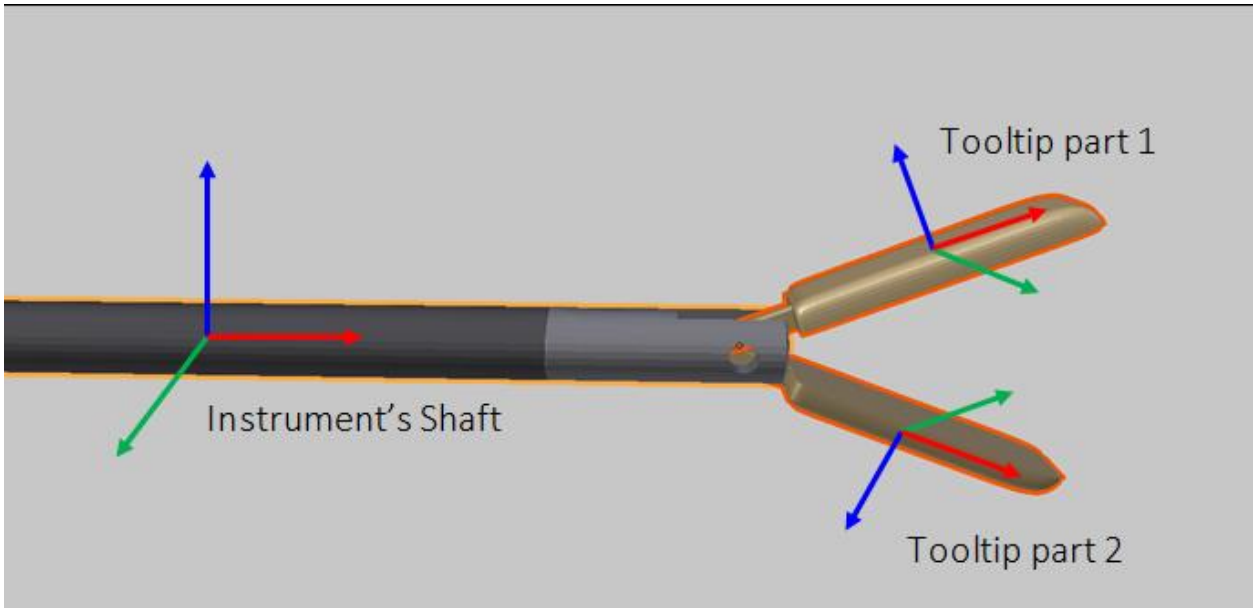


Figure 6.7 A virtual model of a laparoscopic grasper, illustrating its three components and their corresponding coordinate systems.

During training, the virtual representation of an instrument and consequently its components, are controlled by the actions applied to the real laparoscopic instruments. The latter are acquired using sensory devices, as described earlier. Based on this fact, in our initial experimentations we used *kinematic* bodies for the physical simulation of instruments, manually calculating their parts pose with respect to the dynamics world at the beginning of each simulation loop. This design approach though proved problematic for a certain reason; in Bullet, the movement of kinematic bodies is not affected by collisions with *dynamic* bodies during simulation. Consequently, in situations where the instrument's tooltip is closed in order to grab or cut (in case of deformable bodies' simulation) a dynamic object, manual collision checking should be applied in order to avoid interpenetration of the tooltip parts and this object. On the contrary, if the instrument parts where simulated as dynamic objects, collision detection would be automatically performed within the physics simulation step of Bullet, and hence penetrations would be avoided. However, defining instrument parts as dynamic objects also proved problematic, since manually setting the pose of dynamic objects produces inaccurate simulation results in Bullet.

Finally, we concluded in implementing a hybrid of the *god-object* technique [181], used in applications that include haptic-feedback. This technique suggests that virtual objects controlled by user interactions should be modeled as constraint-controlled dynamic bodies. In physics engines, including Bullet, constraints are used to restrict the movements of a dynamic body, or to anchor physics bodies to each other, totally or partially restricting relative movements between them. In the *god-object* technique, each user-controlled object is defined as a dynamic body attached to a constraint, and instead of directly setting the pose of the body itself user actions are applied on the constraint. Regarding the simulation of laparoscopic instruments, we followed a

hybrid approach, defining the shaft as a kinematic body and the tooltip parts as dynamic bodies connected to the shaft via constraints.

6.3.3.3 Simulation of laparoscopic instruments using constraints and motors

The most widely used constraints for computer games are those that model the behavior of real life systems; prismatic, revolute, and point constraints. Prismatic constraints (or slider constraints) allow translation along a specified axis, and no rotational movement. Revolute constraints (or hinge constraints) allow rotation only around a specific axis. Finally, point constraints allow free rotation around the three Cartesian axes, maintaining however a fixed connection point between constrained bodies. The most common constraint usage paradigm is the ragdoll objects used to represent human models in video-games. The individual parts of these ragdolls (hands, feet and head) are connected to the body of the ragdoll using point constraints. These maintain the connection between the moving parts of the ragdoll and the body, allowing freedom of rotation along certain angle ranges.

The two parts of a laparoscopic instrument tip are mechanically connected in such way that they can only rotate around a common axis. This mechanical setup can be simulated using two dynamic bodies linked with a hinge constraint that only allows rotation around a specific axis, as illustrated in Fig. 6.8. Bullet supports hinge constraint using the *btHingeConstraint* class. The hinge constraint implementation of Bullet also provides the option of setting limits to the constraint, restricting rotation between bodies within a desired angle range. Additionally Bullet hinge constraints can be equipped with motors, used to drive a hinge at a prescribed speed until a prescribed angle limit is reached. Constraint motors can be used to control position by specifying a velocity that is proportional to the difference between the actual and desired position.

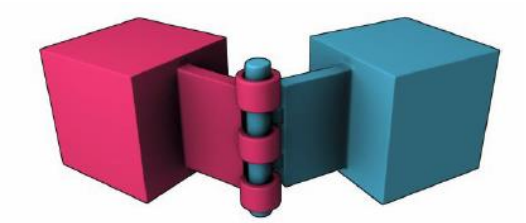


Figure 6.8 A hinge constraint [180], restricting relative additional degrees of freedom and allowing only rotation motion around a single axis.

In the constraint-based approach used to simulate the physical behavior of laparoscopic instruments in the present framework, the kinematic body representing the instrument's shaft is connected with the dynamic bodies corresponding to the tooltip parts using two hinges, one for each part, as illustrated in Fig. 6.9. The angular limit of the hinge motors is updated before each physics simulation step, using the value of angle θ_t acquired from the instrument tracking module.

Since however this value refers to the total angle between the two tooltip parts, each motor hinge limit is set to $\theta_t/2$.

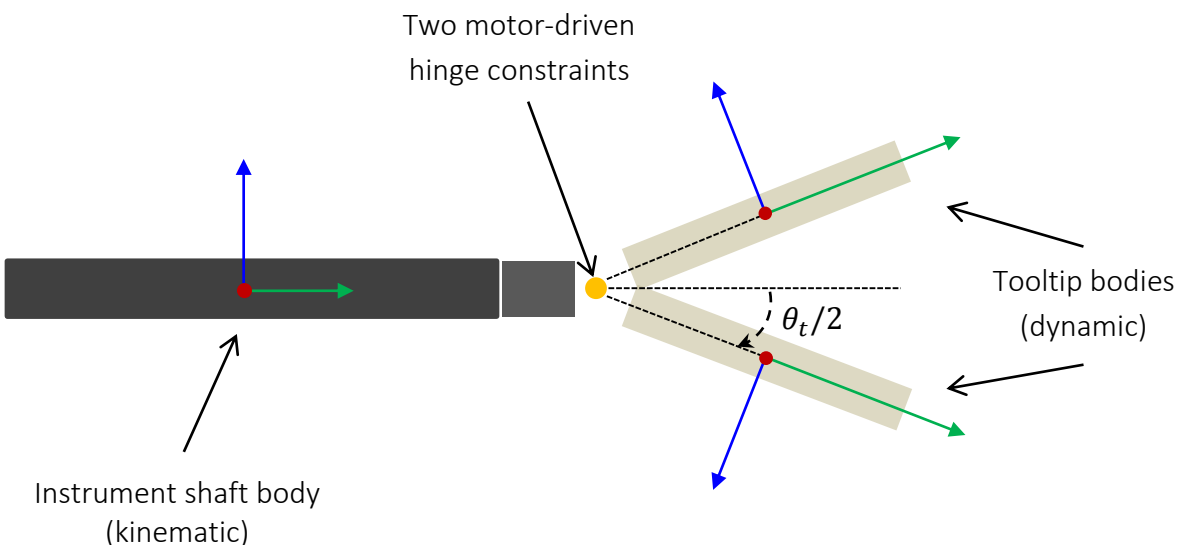


Figure 6.9 Connecting physics bodies corresponding to laparoscopic instrument's part, using a hinge constraint (*btHingeConstraint*).

As we mentioned earlier, the primary reason for using constraints instead of directly applying instrument actions to kinematic bodies, was the automated simulation of *grasping*. By this term, we refer to the action of "catching" and lifting a virtual object with laparoscopic graspers. This is a very common action in laparoscopic training, and realistic simulation of such actions is a necessity for every VR/AR surgical simulation framework. A feature of vital importance for achieving this effect using Bullet motor-driven constraints is the option of setting a prescribed value regarding the force of the motor driving a hinge constraint, provided by Bullet (*maxMotorImpulse*). To achieve stable grasping of virtual objects such as pegs etc. in our training scenarios, we experimented with relatively large values regarding *maxMotorImpulse*. However, stable grasping depends on two parameters, illustrated in Fig. 6.10. The first is the force of the motor driving the hinge. The second is the amount of friction created between the surfaces of the instruments tooltip and the object grabbed by the instrument. Using a hinge with extremely large motor forces produces poor grasping results, since the force applied by the motor by far exceeds the friction keeping objects "locked" between the two parts of the tooltip. On the other hand, a small motor force produces unstable grasping, especially when lifting heavier objects which tend to slip away from the tooltip. Deriving with a correct *maxMotorImpulse* value was a process of trial and error, depended of the training scenario and the types of virtual objects involved. As an example however, for the transfer task illustrated in Fig.6.4 where the masses of the movable rings was set

to 10.0[8], while the *maxMotorImpulse* was set to 5000.0 and the dynamic friction coefficient of both rings and tooltip parts was set to 100.0.

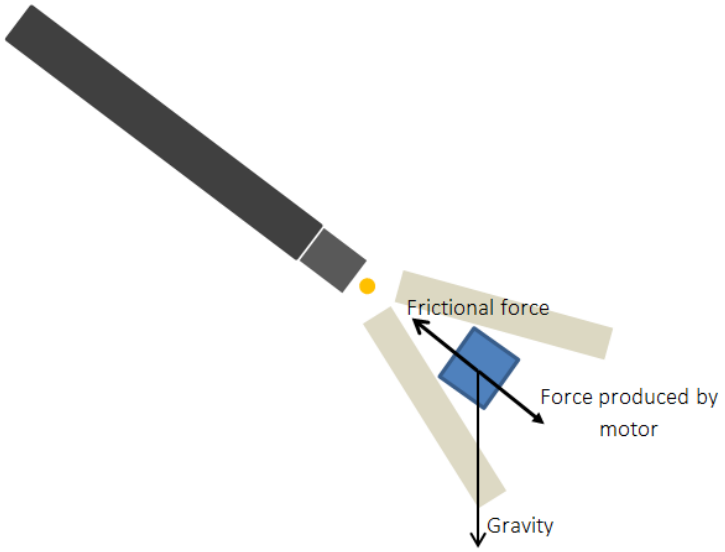


Figure 6.10 Grasping a movable object with a motor-driven constrained laparoscopic instrument.

6.3.3.4 Updating the pose of constraint-based laparoscopic instruments

Due to the motor-driven constraint-based approach described earlier, the dynamic bodies corresponding to the instruments tooltip are attached to the instrument's shaft, allowed only to rotate around a certain axis within a certain angle range. This practically means that any motion applied on the kinematic body corresponding to the instrument's shaft produces relative movements of the tooltip parts. As Fig. 6.7 showed, the instrument's shaft has its own local coordinate system.

The pose of the shaft coordinate system ($M_I{}'$) with respect to the central reference frame of the simulation environment, is calculated using the two measurements (M_I and θ_s) provided by the instrument tracking module. M_I corresponds to the pose of the real instrument's shaft, calculated with the origin defined at the instruments handles, as Fig.6.11 illustrates. The local coordinate system of the virtual object utilized to represent the shaft, is defined at the center of its geometry, at a distance equal to half the instrument's length (l_I) across the y-axis. The value of this length is acquired through a calibration process [155] applied for each instrument, as presented in Chapter 10. Additionally, the shaft coordinate system rotates around its y-axis by an angle equal to θ_s.

[8] Bullet, as most of the physics engines, is unitless. For example, the mass value of an object can correspond to Kilograms or just grams. Maintaining a logical connection between units and sizes of physics bodies is a responsibility of game developers.

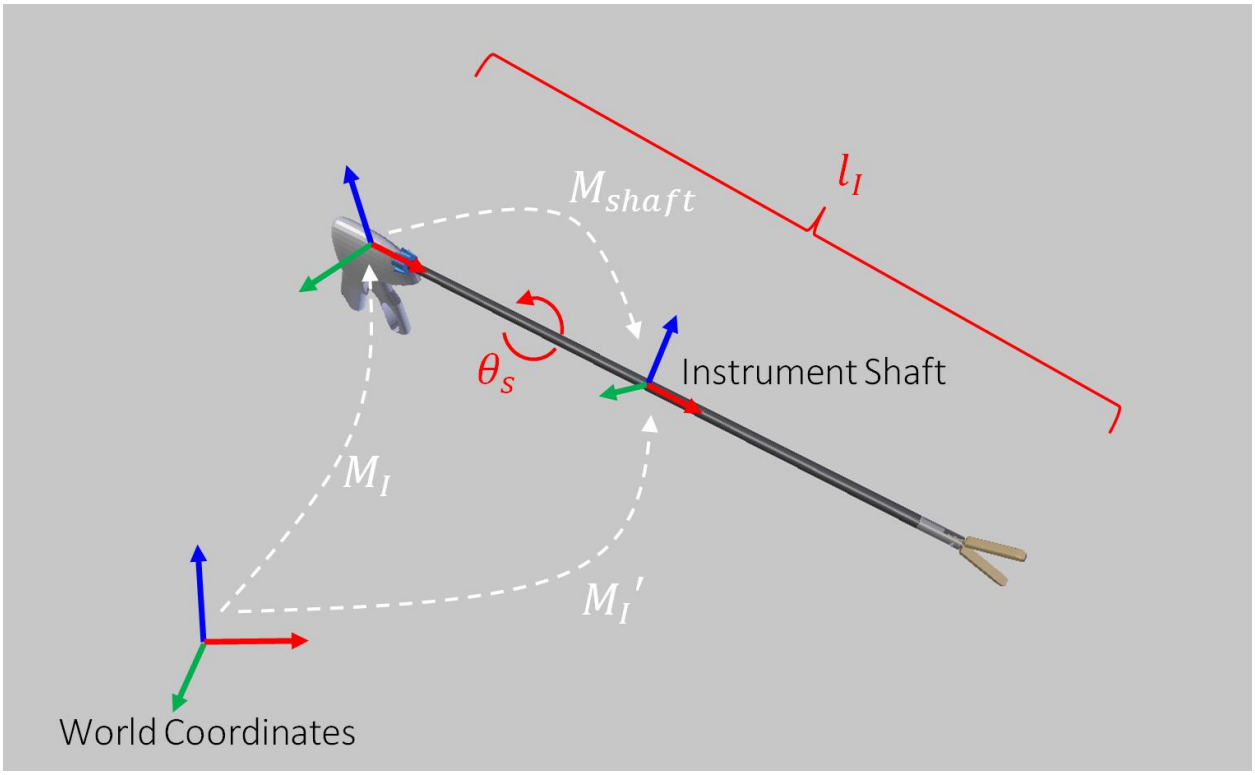


Figure 6.11 Derivation of the laparoscopic instrument's shaft global pose.

Hence, to derive with a transformation matrix that corresponds to the pose of the instrument's shaft, another pose matrix must be calculated, denoted as M_{shaft} in Fig. 6.11. This matrix is a product of a rotation by θ_s around the y-axis and a translation by $l_I/2$ along the same axis. Finally, the instrument's shaft pose is defined as:

$$M_I' = M_I \cdot M_{shaft} \tag{6.1}$$

This calculation is performed at a per-frame basis within the simulation loop, updating the pose of the laparoscopic instruments before execution of the physics simulation step.

6.4 Compensating for the lack of haptic feedback

In the presented AR framework, haptic feedback was not implemented due to several design considerations. Not integrating haptic feedback apparatus in the presented surgical simulation framework however, produced some critical issues that we had to deal with during the development of training tasks. In VR simulation platforms, haptic devices provide a feedback to trainees regarding collision of instruments with other elements of the training environment. Force feedback prevents trainees from performing actions that would be practically impossible in a real world environment, such as rigid-rigid objects penetrations etc. A crucial weakness arising from the lack of haptic feedback in the presented platform is that the trainee is allowed to perform

otherwise unrealistic actions with the instruments, as for instance forcing the instrument tip instrument against static rigid bodies or forcing a dynamic (movable) rigid body to penetrate on static rigid body. Since the forces applied in dynamic bodies of the training environment are calculated as a result of collisions with laparoscopic instruments and other physics objects of the training environment, such unrealistic actions could occasionally result in catastrophic failures of the overall simulation experience. Specifically, forcing a dynamic body against a static body, as for instance the torus-shaped peg illustrated in Fig.6.3 against the bottom plane of the box-trainer, can produce totally unrealistic effects such as rigid-rigid body penetration or a sudden high speed dislocation of a movable object caused by a sudden increase of the force applied on it.

6.4.1 Adding visual feedback to the simulation experience

To overcome such issues, commercially available VR simulators not equipped with haptic feedback implement custom "tricks". The basic commercial version of LapMentor [182] for instance, not equipped with haptic feedback apparatus, is calibrated in such way that the purely virtual instruments can never reach the bottom plane of the training environment, or other static virtual objects involved in a training scenario. In another commercial simulator called Lap-X [183], once the virtual instruments come in contact with static virtual objects, the simulation loop is paused and the user is asked to fully retract the instruments in order to continue the exercise. We have considered both of the aforementioned solutions and derived with the conclusion that the first could not be implemented in an AR training environment while the second was a rather poor solution in terms of simulation realism. In the presented framework, we decided to follow a different design approach where visual feedback would compensate for the lack of haptic feedback, forcing the trainees not to perform unrealistic actions with the laparoscopic instruments.

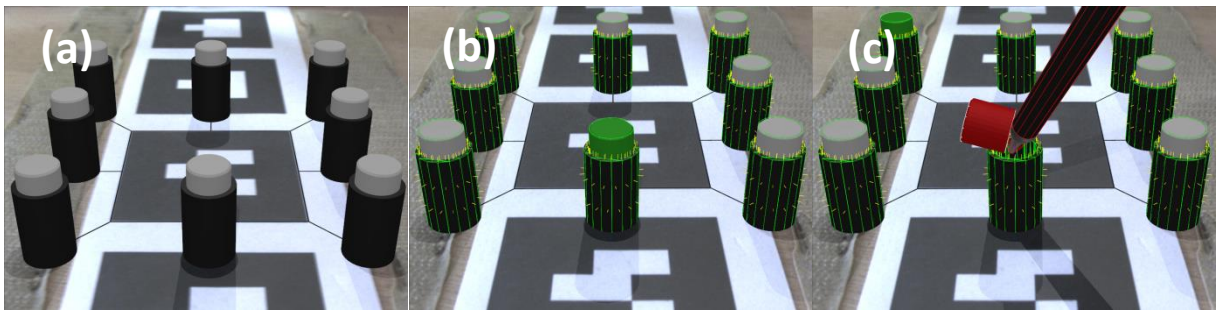


Figure 6.12 (a) Instrument navigation task requiring trainees to hit on virtual buttons. (b) Buttons are modeled as small cylinder-shaped movable bodies positioned into static tube-shaped bases. (c) Due to the absence of haptic feedback, trainees can force buttons out of their bases.

An example illustrating the integration of visual feedback is the instrument navigation training task implemented in our framework. In this task, trainees are asked to hit virtual buttons using a laparoscopic instrument, as illustrated in Fig. 6.12a. To simulate the behavior of these virtual

buttons, each is defined as a cylinder-shaped dynamic object, positioned into a static tube-shaped base (Fig. 6.12b).

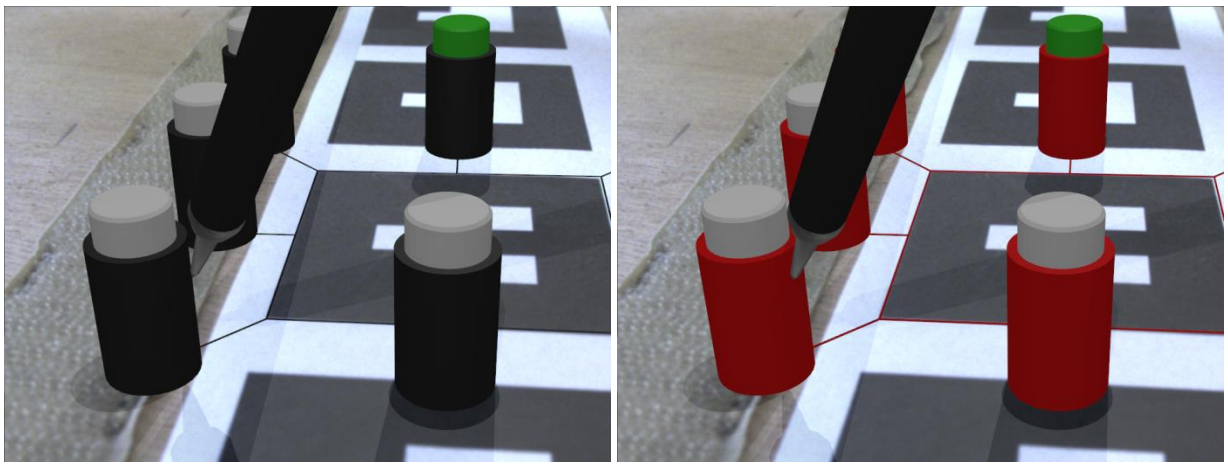


Figure 6.13 Visual feedback motivating trainees not to perform unrealistic actions; when the instrument touches or penetrates a static rigid object, the latter becomes red, denoting that an error has occurred.

The button and its base are connected with a point constraint (*btPoint2PointConstraint*) that acts as a spring, forcing a button to return to its initial location when no external forces are applied on it. Since each button is constrained by the inner walls of the static tube, it is only allowed to move up and down on the vertical direction. However, no haptic feedback exists and hence nothing can stop the trainee from applying forces on the sides of the button, forcing it to penetrate the walls of the tube and pop out (Fig. 6.12c).

Utilizing the idea of visual feedback, the training task is designed in such way that when a laparoscopic instrument touches the sides of a static tube, the latter become red as illustrated in Fig. 6.13, visually notifying the trainee that an error has occurred. Additionally, the instructions of the training scenario clearly state that touching these tubes with an instrument or forcing a button out of its base is counted as an error on the final performance analysis. Using this technique, trainees are motivated not to perform unrealistic actions.

6.4.2 Substituting static objects with dynamic

Visual feedback, described in the previous section, is one way of compensating for the lack of haptic feedback, preventing trainees from performing actions that they should not perform. The visual feedback idea motivates trainees not to perform unrealistic actions, but does not however ensure catastrophic failures won't be caused as a result of trainees ignoring this type of feedback. Additionally, actions such as pushing movable objects against static rigid objects cannot be avoided in some training tasks. In the ring transfer task presented earlier for instance, in order to grasp and lift a virtual ring, the trainee has to push this ring against the rigid body corresponding

to the bottom plane of the training environment, referred to as *ground body* in the following paragraphs.

In our initial experimentations on the development of this training task, the ground body was defined as a box-shaped Bullet static body, located below the "ground" of the simulation environment, with its top surface exactly aligned with the bottom plane of the box-trainer. Trying to grasp and lift an object lying on the ground plane as in the aforementioned example of the ring transfer task, penetrations between the rigid movable body of the ring and the ground body were unavoidable, resulting in the ring finally passing through the bottom plane, destroying the overall sense of realism of the simulation. To avoid such effects, we defined the ground body as a *hybrid* (movable), anchored to the correct location using constraints. As we described in Section 6.3.2.3, hybrid bodies collide with moving bodies but not with kinematic ones. The result is a ground body that collides with falling objects such as rings, pegs etc., but does not collide with the laparoscopic instruments.

Since this body must remain at the correct location throughout the simulation, it is anchored at its original position with four strong point-to-point constraints located at its corners. A point-to-point (p2p) constraint is effectively a spring, anchoring a dynamic body at a certain 3D position. The parameters of the constraint define its hardness and elasticity, allowing their use for replicating spring-damper systems. Using Bullet p2p constraint implementation (btP2pConstraint) and setting the appropriate constraint parameters (BT_CONSTRAINT_CFM = 0.9, BT_CONSTRAINT_STOP_ERP = 0.1), we ended up with very hard p2p constraints. A hard p2p constraint means that in order to move the constrained body, significant amounts of forces must be applied on it. Additionally, setting a high number (120) of constraint solver iterations (the number of iterations performed by Bullet internal solver at each simulation step), the constraint body returned to relaxation at its initial location without performing significant oscillations.

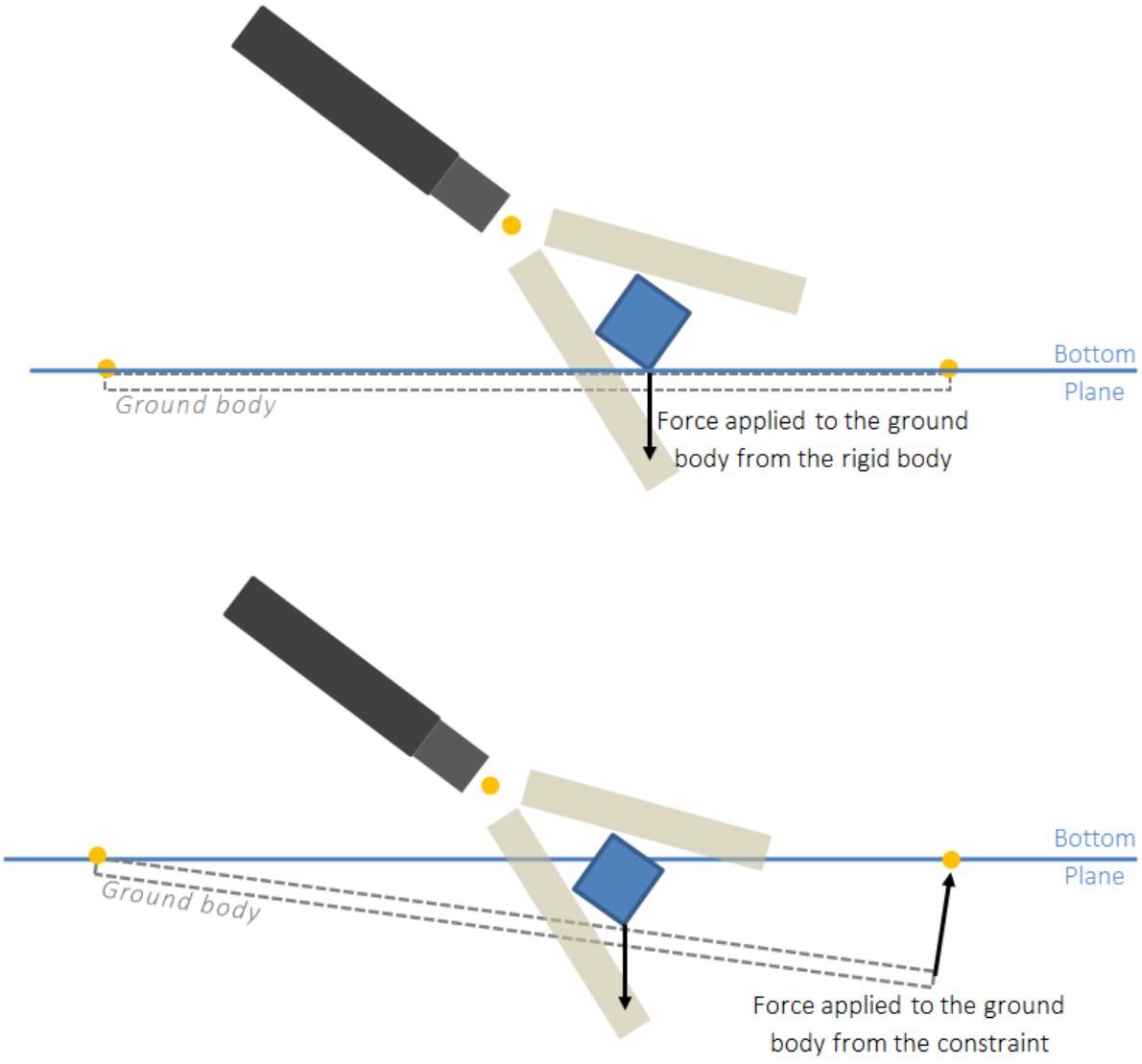


Figure 6.14 Illustration of the movable constrained ground body concept. Point-two-point constraints, indicated as yellow dots, force the ground body to return at its initial location.

The constrained movable ground body idea is illustrated in Fig. 6.14. As it can be seen from this figure, forces applied from a grasped object to the ground body result in a slight dislocation, but the hard p2p constraints force it to return to its initial location. In essence, the constraints strong and hence the ground body behaves as a static body, allowing however some slight movement tolerance that prevents rigid-rigid penetrations. As Fig. 6.15 shows, implementing this idea in practice, we ended up with a far smoother simulation, allowing trainees to grasp objects from the box-trainer bottom in a realistic way.

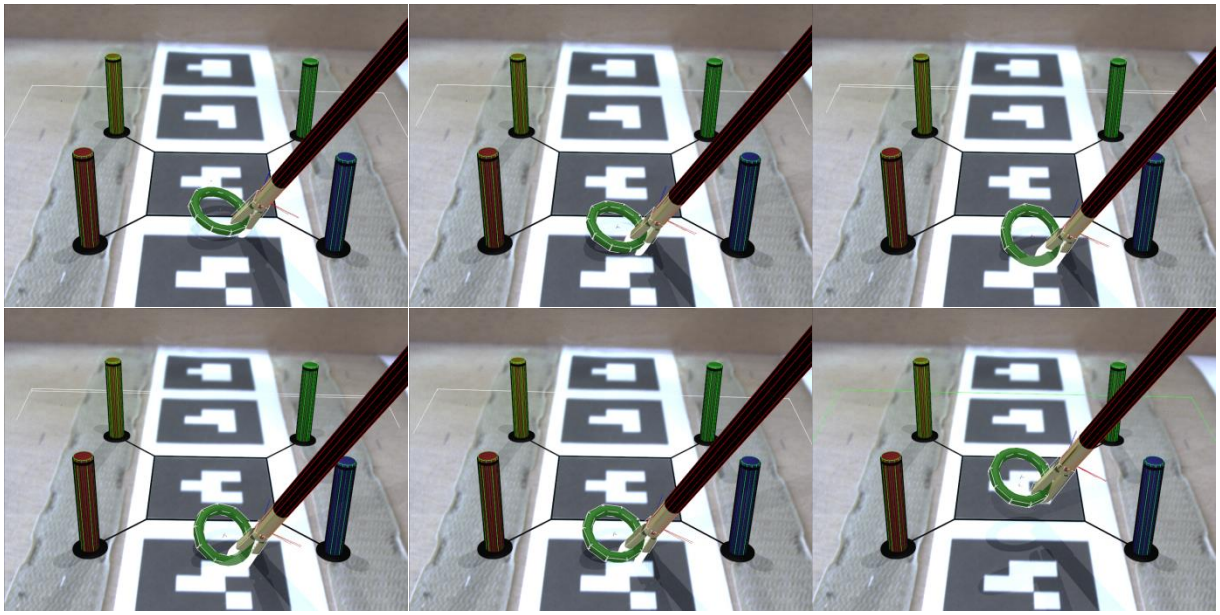


Figure 6.15 An AR ring-transfer task implemented in the proposed framework. Introduction of a movable-constrained ground object resulted in smoother simulation of grasping.

6.5 Soft body dynamics

At the final stages of the development process, we experimented with the soft-body simulation capabilities of Bullet, aiming to create training tasks that would involve interactions between laparoscopic instruments and deformable objects such human anatomical parts. Our goal was to evaluate the potentials of the proposed platform for implementing sophisticated training exercises that simulate real surgeries, such as laparoscopic cholecystectomies etc. Such exercises are implemented in state-of-the-art commercially available training platforms. Simulation of deformable bodies is a highly demanding task requiring heavy modification of the Bullet soft body implementation. Our goal however was not to create a counterpart to commercial platforms, rather than evaluate the potential utilization of deformable bodies' simulation in an AR environment. In this context, we created a very basic training scenario, requiring the user to clip a virtual artery and then cut it at a predefined location. Some screenshots of this training task are shown in Fig. 6.16.

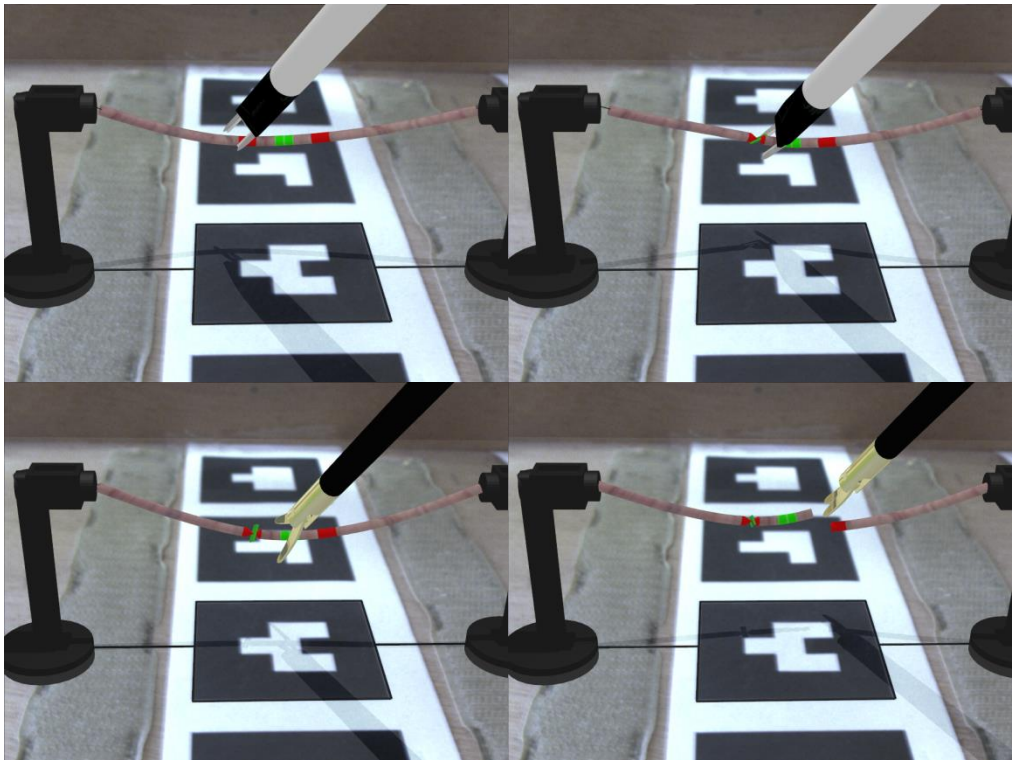


Figure 6.16 Screenshots of the clipping (top) & cutting (bottom) training task implemented in the presented framework.

6.5.1 Soft bodies integration in the presented framework

Implementation of soft bodies in Bullet is achieved using the *btSoftBody* class, which is the equivalent of the *btRigidBody* class mentioned in previous sections. Similar to rigid bodies, soft bodies also are defined by a collision shape and additional physical characteristics. The collision shape of a soft body is described as a collection of 3D points called *nodes*, connected with *links* and forming triangles (called *faces*), exactly the same way as described in Chapter 3 for the definition of 3D shapes. However, the connection between a Bullet soft body and an Ogre3D scene node, used for virtual objects visualization, differs from what we have described in previous sections of this Chapter.

6.5.1.1 Linking btSoftBody to Ogre3D SceneNode

Integration and visualization of deformable body simulation in the presented framework required a different sequence of physics/rendering operations compared to what we have described in the present thesis up to this section. The main difference of soft bodies is the definition of their location within the simulation environment. Unlike rigid bodies, soft bodies in Bullet don't have a single world transform. The location of each node is specified in respect with the central reference frame of the dynamics world. Additionally, the location of each node and hence the shape of the deformable body changes dynamically during simulation, while the shape of virtual objects that

we have used up to this point, is set at the initialization stage of the simulation and remains unaffected throughout the training session. Consequently, both the structure presented in Table 6.2 and the motion state technique described in section 6.3.2.2, are not directly applicable for the integration of soft bodies in our framework.

Table 6.4 General structure of Ogre meshes and Bullet soft bodies

Ogre mesh		Bullet soft body		
Type	Size	Type	Size	Information
Vertex buffer	6 floats/ vertex	Nodes list	Nodes number	Node info (location vector, normal vector)
Index buffer	3 integers/triangle	Faces list	Faces number	Face info (pointer to face nodes)

In Section 5.4 of the previous chapter we have mentioned that Ogre3D provides the option of dynamically updating the visual shape of a *SceneNode* at runtime. This can be achieved in two ways. The first is by using an Ogre3D *manual mesh*. This option allows the manual creation of a mesh that can be redefined by changing the 3D location of each individual vertex as well as the list of mesh triangles at a per-frame basis, thus allowing the visualization of virtual objects whose shape changes overtime, exactly as it happens with soft bodies. However, the implementation of manual mesh modeling in Ogre3D is not optimized for framerate performance, and produces significant overheads in memory consumption. The second way of dynamically changing the shape of a mesh is by directly accessing the memory buffers used to store mesh information. Although this is not a supported/documented procedure in Ogre3D, it proves to be far more effective in terms of performance and hence, a better solution for a surgical simulation platform.

Meshes in Ogre3D are stored in two memory buffers, as illustrated in Table 6.4. The first is the *vertex buffer*, storing the 3D positions of each mesh vertex as well as the vertex normals. Each vertex is stored in the vertex buffer using 6 floating point numbers (three for location, three for normal). The second is the *index buffer*, storing information regarding the triangles of a mesh. Since each triangle is described by three vertices, triangles are stored in the index buffer as 3 consecutive integers, pointing to the index of these corresponding vertices in the *vertex buffer*. As also shown in Table 6.4, the btSoftBody class of Bullet mainly consists of two C++ containers, one designed to hold node information (node list) and one designed to hold face information (face list). In the btSoftBody implementation, nodes are defined as C++ structures that consist of a vector of floats defining the node's location and a vector of floats storing the node's normal. Faces are defined also as C++ structures, consisting of a vector of memory pointers, pointing at the corresponding nodes in the nodes list of btSoftBody. A visual representation of a soft body, implemented in Bullet and visualized using Ogre3D is shown in Fig. 6.17.

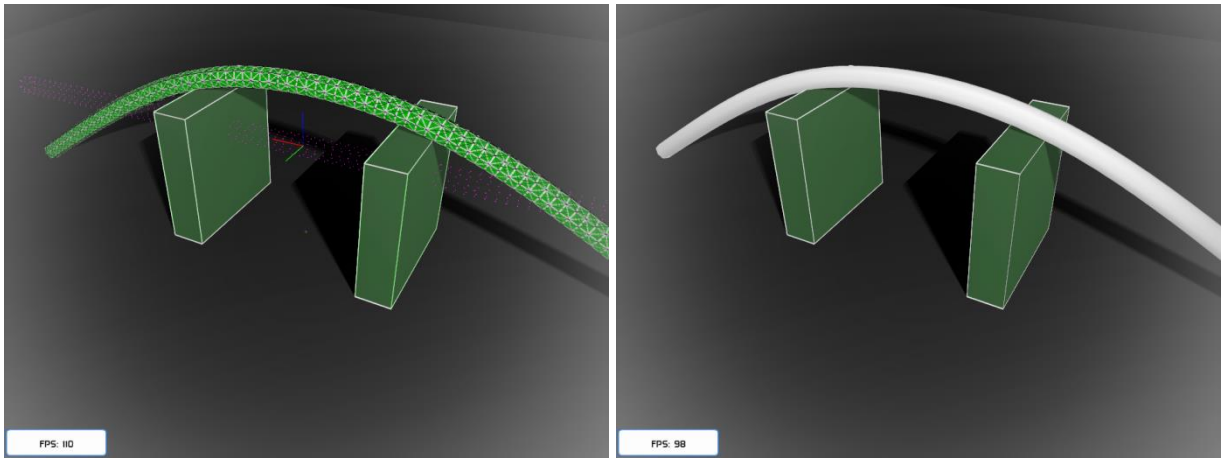


Figure 6.17 Simulation of a deformable cylinder resting on top of two rigid objects. The btSoftBody is illustrated on the left image, while the corresponding visual mesh is shown on the right.

Creating a connection between the Ogre3D mesh and the btSoftBody was the key for the implementation of deformable body simulation/visualization in the presented framework. As illustrated in the flowchart of Fig. 6.18, this connection includes a two-way communication between these two entities; at the initialization stage, the soft body is created using mesh data while at the simulation loops, mesh data are updated using soft body information:

Creation of soft body from mesh: At the initialization stage, an Ogre *SceneNode* is added to the scene manager of Ogre3D, as performed for every virtual object. This node is assigned with a visual shape (*mesh*) and an initial pose. Consequently, each vertex of the SceneNode's mesh is stored in the vertex buffer with the appropriate global coordinates. Then, a btSoftBody is instantiated and its collision shape is defined using vertex data information, obtained of the Ogre SceneNode mesh buffer. This is performed in a loop that creates a node structure for each vertex of the buffer, and adds it to the corresponding list of the btSoftBody. An important detail regarding this process is that Ogre meshes contain duplicate vertices, which are sorted out before the btSoftBody creation.

Mesh update from soft body: After each call of the physics simulation step, the 3D position of btSoftBody nodes is updated. Additionally, the normals of these nodes are also automatically updated by Bullet. Looping through each node and updating the corresponding values of the vertex buffer of Ogre with new vertex locations and normals, the visual representation of the deformable body is dynamically reshaped. It is important to note that in training tasks involving cutting of virtual deformable objects, the index buffer of the Ogre mesh also need to be updated to reflect the btSoftBody faces list, since a cutting operation will remove some of these faces.

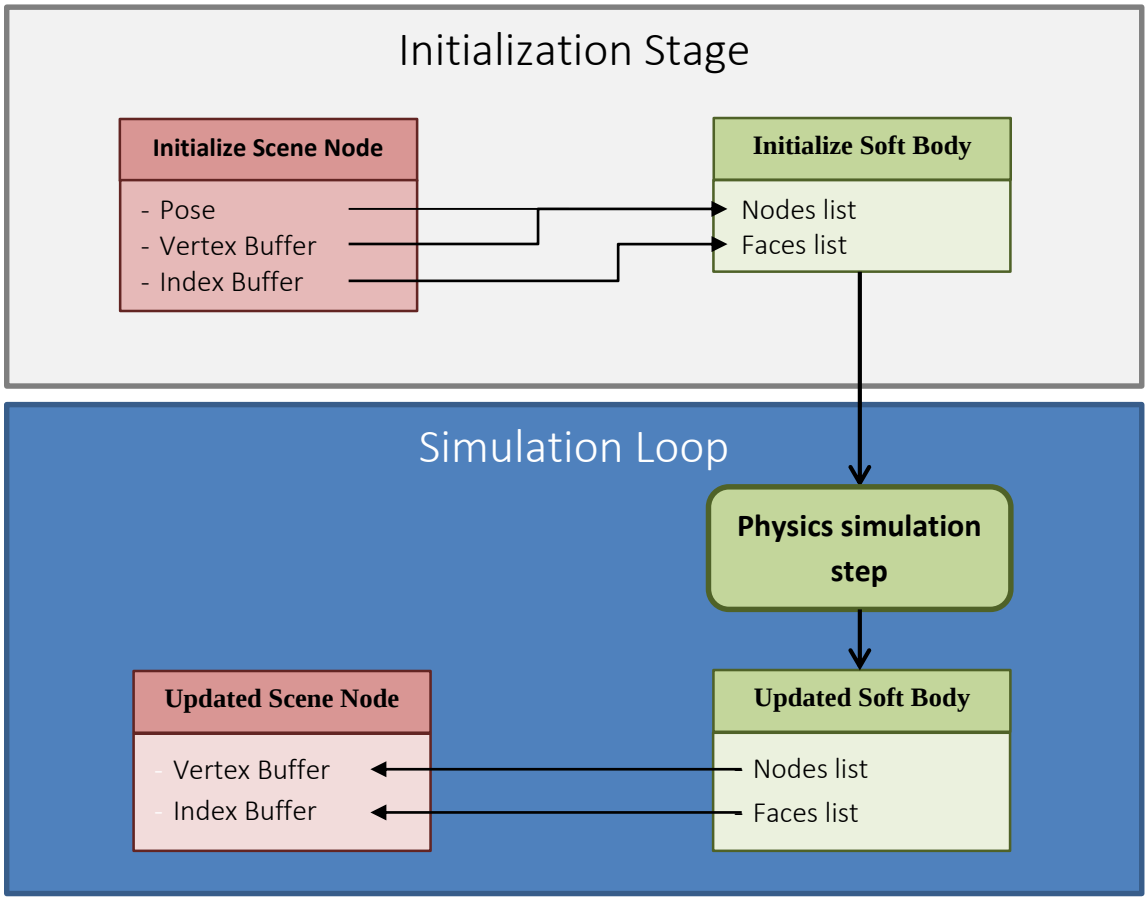


Figure 6.18 Sequence of operations for connecting a *btSoftBody* with the mesh of an Ogre *SceneNode.* At the initialization stage, the soft body is created from Ogre mesh data, while during the simulation loop the mesh is updated using soft body data.

6.5.1.2 Instrument interaction with deformable bodies

The main purpose of soft body implementation in a surgical simulation framework is the creation of training tasks where users can manipulate deformable objects using the laparoscopic instruments. Consequently, the engine responsible for real-time physics must provide a realistic as well as time-efficient simulation of such interactions. By default, collision detection between a Bullet soft body and another dynamic or kinematic object of the dynamics world is performed using the soft body nodes. During the collision detection phase, if a node lies within the collision margin of a dynamic object, a collision is registered.

If however the collision shape of a dynamic object passes through the surface of a soft body without touching a node, the collision is missed, causing rigid-soft body penetrations. Regarding the laparoscopic instruments presented in previous sections, this phenomenon is schematically illustrated in Fig. 6.19.

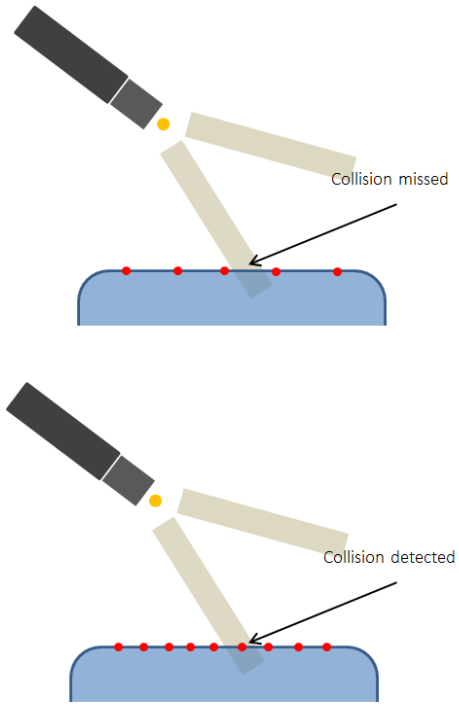


Figure 6.19 Collision detection between soft and rigid bodies is performed using the 3D locations of soft body nodes. If part of the rigid body passes between two connected nodes, collisions are missed.

This means that in order to ensure a correct collision response and reduce penetration of instrument's to deformable virtual objects, soft bodies must be designed using a dense tessellation (a large number of nodes, at small distances to each other). However, increasing the number of soft body nodes increased the computation time required for performing a physics simulation step, and hence, decreases the frame-rate efficiency of the simulation. As a first line of defense, we designed our deformable models enough resolution so that distances between nodes were always small enough for the tooltip of laparoscopic instruments to pass through them. However, when forces are applied on a deformable body, the latter tends to stretch and hence distances between nodes change overtime. Unfortunately, we found no global solution to this issue, one that would allow seamless interaction between laparoscopic instruments and soft bodies. Experimenting with a large number of properties available in the btSoftBody implementation, we have managed to achieve an adequate simulation of simple-shaped deformable virtual objects, such as the virtual artery illustrated in Fig. 6.16. Using such shapes, the presented framework provides the means for developing basic surgical training tasks, used for training simple interactions such as clipping and cutting. Modern laparoscopic simulation however is involved with far more complex scenarios and interactions, simulating real surgical procedures such as laparoscopic cholecystectomy etc. As discussed in the conclusions of this thesis, the soft-body implementation of the presented framework requires further improvements in order to achieve better simulation quality which would allow implementation of training scenarios involving complex soft body geometries and interactions.

Chapter 7

Experiments and Results

As stated in the introduction, the main purpose of the presented PhD thesis was to investigate the potential use of AR in the field of laparoscopic simulation training. Previous Chapters illustrated the important principles of real-time graphics and physics, as applied for the creation of the corresponding software modules responsible for providing simulation and visualization of interactions between laparoscopic instruments and virtual rigid/deformable objects within an AR environment. This Chapter outlines the main development stages towards the realization of the presented simulation framework and includes a detailed description of an evaluation study that we performed during the first development stage.

7.1 Development and Experimentation Stages

Deriving with the final design of the presented framework was a lengthy procedure that included multiple experimentations, modifications and improvements regarding each of the core simulation components; AR graphics, instrument tracking and real-time physics. Specifically, in the present thesis we have implemented, utilized and evaluated the accuracy of three different types of instrument tracking techniques. Moreover, as mentioned in Chapter 5, we have developed two different versions of our AR graphics engine, one based on OpenGL and one based on Ogre3D. Finally, at the final setup of our framework we have employed real-time dynamics calculations using Bullet Physics engine.

These experimentations allowed us to create essential building blocks for the implementation of a fully functional AR-based laparoscopic simulation framework, which served as a basis for the creation of training scenarios of high simulation quality, equivalent to this of commercially available simulators. This process can be divided in three main experimental stages, each employing a different version of our framework in terms of both hardware and software components. Overall, the development process of the presented framework can be split in three stages, as illustrated in Table 7.1.

Table 7.1 The three development stages towards the implementation of the presented laparoscopic simulation framework.

	Graphics	*Instrument Tracking*	*Physics*
Stage 1	OpenGL	Pattern markers	Not employed
Stage 2	OpenGL	Image-based	Not employed
Stage 3	Ogre3D	Sensor-based	Bullet Physics

On each stage, we performed evaluation studies based on training scenarios built upon the current version of our framework, acquiring significant findings regarding the efficiency of the framework in training and assessment of basic surgical skills, as well as indications on its advantages and disadvantages as compared to commercial surgical simulation platforms. Moreover, experimentation with various instrument tracking techniques involved the development of new calibration and tracking algorithms. Specifically, during the second stage of our research process we have developed an image-processing algorithm for real-time instrument tracking, while for the introduction of sensor-based tracking during the third experimentation stage we developed a novel calibration technique for estimating the spatial relationship between an instrument and an EM sensor attached on its' handles. The accuracy and performance of these techniques in estimating the 3D pose of laparoscopic instruments were also evaluated in relevant studies.

This Chapter provides a thorough presentation of the main research and development process stages as denoted by Table 7.1, describing details regarding the implementation of the software

and hardware setup used in each experiment, the validation studies performed and the results obtained, as published in relevant peer-reviewed scientific journals.

7.2 A primary study on laparoscopic skills assessment using AR scenarios

As depicted by Table 7.1 and described with details in Chapter 5, the first version of the presented framework utilized pattern markers for instrument tracking and OpenGL for rendering. To obtain some preliminary findings regarding the potentials and limitations of AR in laparoscopic training application, we implemented an elementary AR scenario that required trainees to identify and place the laparoscopic tools at the position of virtual objects (spheres) was developed. Using this scenario, we conducted a study for the evaluation of our framework efficiency in the assessment of psychomotor laparoscopic skills. The assessment of depth perception was performed using Hidden Markov Models (HMMs) and Dynamic Type Wrapping (DTW). In this study we investigated the potential use of augmented reality for simulation training in laparoscopic surgery.

7.2.1 Experimental setup

Our experimental protocol included two groups (A & B) of eight subjects each. Group A (experts) consisted of general surgeons with extensive experience in laparoscopic surgery and Group B (novices) included medical students with no experience in laparoscopy. Each subject had to perform two times a simple task that evaluates their depth perception and hand-eye coordination skills. For this purpose two virtual spheres were introduced on a computer monitor that showed a live image from a standard USB camera (960×720) mounted at a fixed position inside a box-trainer. The spheres were virtually placed at known positions (15cm apart) on the surface of a simulated synthetic stomach. Each subject had to touch a highlighted sphere with the tip of a laparoscopic instrument and then move the instrument to the location of the second sphere. Touching was defined as the positioning of the tooltip inside the volume of the sphere. The task was completed as soon as the second sphere was touched with the tooltip. The elapsed time and the tip trajectory in the 3D space were used for performance evaluation assessment. A total of 2×8×2= 32 trajectories were collected (16 from each group).

7.2.2 Achieving real-time AR visualizations

An AR algorithm running on top of the camera stream provided the base for realistically projecting the virtual models to the real scene image. This algorithm was built based on the ARToolkitPlus library (*http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php*), using standard id-based markers for virtual augmentations and instrument path monitoring.

7.2.2.1 Coordinate systems and tracking

To introduce virtual spheres into the real scene image required the transformation from the real world coordinate system to the camera coordinate system. To achieve this an ARToolkitPlus marker was placed on the base of a box trainer (surface marker). Tracking this marker at each camera frame returned a transformation matrix (T_{BC}) that gave the 3D position of the marker w.r.t. the camera coordinate system. This matrix consists of a standard orthogonal 3×3 matrix that corresponds to a clockwise rotation using Euler angles, and a Translation vector. The transformation matrix was used as the projection matrix for drawing virtual objects using the OpenGL library. The marker could not be tracked when was occluded by the tool and thus as an approximation we used its position at previous frames to calculate T_{BC}. Considering the fixed position of the camera, this approximation was found adequate for our application.

The same tracking process was applied to detect the position and orientation of the laparoscopic tool relative to the camera coordinate system. An id-based marker attached on the tool at a fixed position with respect to the tooltip provided another transformation matrix (T_{TC}) with the position and orientation of the tool relative to the camera.

Figure 7.1 demonstrates the positioning of the markers in the simulation environment and the resulting transformation matrices. The following matrix operation provides the position of the laparoscopic tool in the coordinate system of the box trainer (T_{TB}).

$$T_{TB} = T_{BC}^{-1} \times T_{TC} \tag{7.1}$$

Using T_{TB} and the known relative position between the tooltip and the tool marker, we were able to track the path of the tooltip in reference to the box trainer environment.

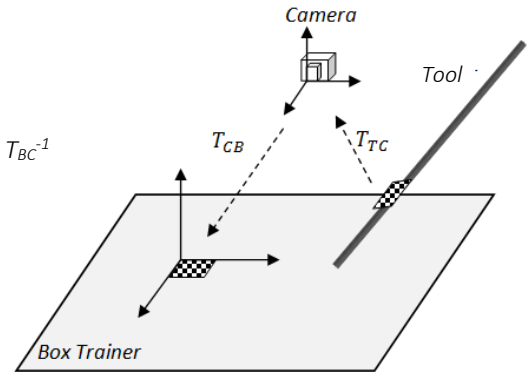


Figure 7.1 Tracking and transformation between the camera and laparoscopic instrument, box-trainer, using id-based markers.

7.2.2.2 Occlusion handling

In this application occlusion problems were encountered in occasions where the tool was placed in front of a virtual sphere. Real-time occlusion handling was achieved by using an approximate 3D model of the instrument's shaft and its pose (position and orientation) estimation at each frame. The approximate 3D model was then added to the OpenGL depth buffer in real time using the pose information.

7.2.3 Signal processing

The 3D trajectories were processed using two different versions of an HMM algorithm (discrete and mixture of Gaussian outputs), and a standard Dynamic Time Wrapping (DTW) algorithm.

7.2.3.1 Dynamic time warping

The DTW [184] is both a time-invariant similarity measure and a method to synchronize two time series by finding a non-linear warping path. Each point in one time series is wrapped onto at least one point in the other time series while respecting the temporal order. This is done for minimizing the sum of the distances between all points that are warped onto each other. DTW has been applied in various domains, including laparoscopic surgery, to synchronize series of application-dependent feature vectors [185]. The input to the algorithm was the Centroid Distance Function (CDF) extracted from the 3D trajectories. The CDF offers a view- and affine-invariant representation[186], and it has been successfully employed for assessment of similar movement trajectories in MIS [187]. The CDF essentially represents a time series of the Cartesian distance of each data point in the trajectory from the centroid (i.e. the weighted average of all the points in a particular trajectory). Each subject's CDF was classified into either of the two groups after calculating the minimum DTW-distance to all other CDFs (leave-one-out cross validation). Sensitivity and specificity were used as estimates of the classification performance.

7.2.3.2 HMMs

As an alternative approach, the subjects' trajectories were modeled with HMMs [188]. In brief, a HMM can be described by a model of three parameters representing the relationship between the observed data (y) and a number of hidden states (s_i):

$$m = \{\pi_i, a_{ij}, p(y_t / s_i)\} \tag{7.2}$$

where π_i is the prior ith state probability, α_{ij} is the state-transition matrix, and $p(y_t/s_i)$ is the probability of generating an observation y_t given the hidden state s_i. Given an observation vector from a subject, the model parameters were obtained with the Baum-Welch optimization (HMM training, [188]).

A separate HMM was developed for each subject's trajectory (i.e. 32 different models), which was then classified into each group by measuring the statistical distance (similarity) to each of the other models. The statistical distance between two HMMs m_1, m_2 was defined as [189]:

$$D(m_1,m_2) = \frac{1}{T_{y_2}}(\ln p(y_2 / m_1) - \ln p(y_2 / m_2)) \quad (7.3)$$

$$D(m_2,m_1) = \frac{1}{T_{y_1}}(\ln p(y_1 / m_1) - \ln p(y_1 / m_2)) \quad (7.4)$$

where $D(m_1,m_2)$ is a measure of how well m_1 fits observations generated by m_2 relative to how well m_2 fits the observations generated by itself, and T_{y1}, T_{y2} are the lengths of y_1, y_2 respectively. The marginal probabilities $p(y/m)$ were calculated via the forward-backward procedure [188]. Then the two distances were averaged to generate the final symmetrical distance used for trajectory classification. Similarly to DTW the sensitivity and specificity were used to evaluate the classification accuracy.

We employed two different HMM designs, each one with a different set of features: discrete and Gaussian outputs. For the first case as primitives movements we considered the horizontal deviation (*hd*, *x-y* plane) and elevation (*dz*, *z* coordinate) of each data point from the line connecting the virtual spheres. Each movement was discretized into three different symbols as shown in Table 7.2, leading to a total of 9 symbols. A trajectory was then transformed into an equal length vector where each entry corresponded to a different symbol. The feature vector of a trajectory was used as input to the HMM.

Table 7.2 HMM Motion vocabulary consisting of 9 discrete symbols.

Symbol		Sym1	Sym2	Sym3	Sym4	Sym5	Sym6	Sym7	Sym8	Sym9
HD	+	•	•	•						
	-				•	•	•			
	0							•	•	•
DZ	+	•			•			•		
	-		•			•			•	
	0			•			•			•

The primitives motions were discretized according to their sign. For example, for elevation it is dz=sign(z_t-z_{t-1}), where z_t is the distance of the tooltip from the spheres connecting line in the z direction at time t. A zero elevation (no movement in z direction) was considered when -0.7<dz<0.7 mm. The same range of values was used for the horizontal deviation hd.
HD (Horizontal deviation). DZ (Elevation)

For the second HMM we used the normalized CDF of each trajectory as an observation vector. The CDF was modeled with a mixture of two Gaussians: $p(y_t/s_i)=\Sigma\mathcal{N}_i$, where y_t is the CDF value at time t, and $\mathcal{N}_i$ ($i=1,2$), is a Gaussian distribution with mean and standard deviation: μ_i,σ_i respectively. Note that compared to the discrete outputs where an observation can take only one of nine

distinct values, the second HMM employs vectors of real numbers. The mean and standard deviation values were calculated during HMM training.

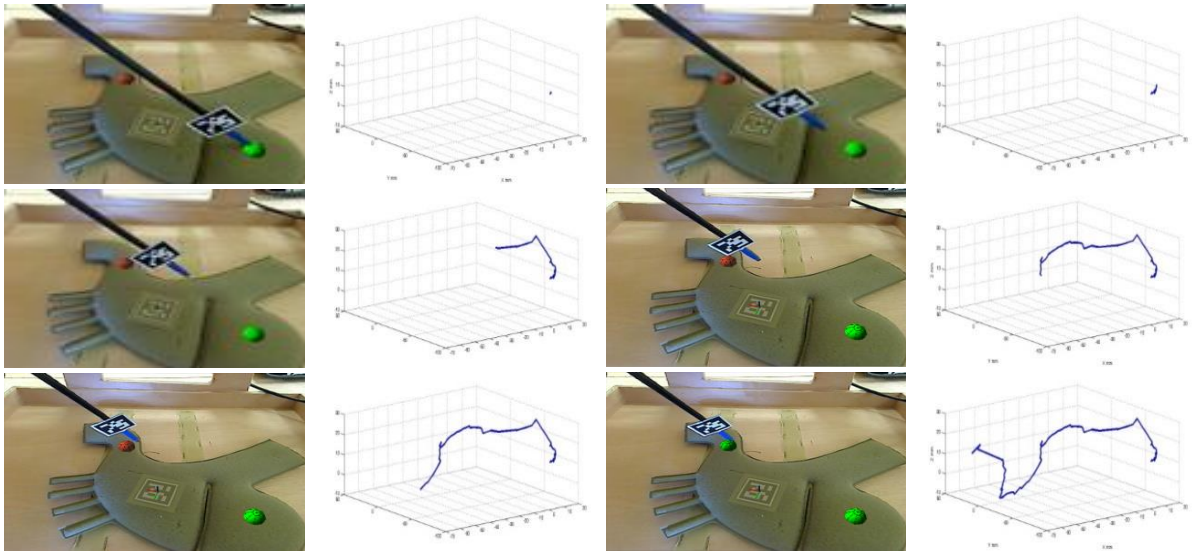


Figure 7.2 Image frames of the AR training task and the corresponding instrument path. The frames demonstrate in a row-wise manner the movement of the tool. The color of the sphere in the first frame (top-left) was changed to green as soon as a contact with the tooltip was identified. The same applies also to the other sphere (bottom-right frame).

7.2.4 Results

The image acquisition rate was fixed to 30Hz throughout the experimental sessions. Figure 7.2 illustrates a series of frames along the elapsed trajectory of the instrument tip during the experiment. The subject first touches the nearest sphere (A), and then moves to the other one (B). The marker shown on the surface of the stomach was used to project the spheres at rigid positions w.r.t. the marker. The other marker, attached near the instrument tip, was used to acquire the 3D position of the instrument tip w.r.t. the surface marker. Each sphere changed color as soon as it was touched with the instrument tip, providing a feedback of successful performance to the user. For each experimental session the trajectory was saved for further processing. ARToolkitPlus was proved very robust during tracking of the instrument's marker; the number of 'missing frames' (i.e. unable to track the instrument marker) was typically less than 20 out of 90-160 frames. The tip position in consecutive missing frames was calculated retrospectively by a linear interpolation of the tip position between the two extreme known frames.

Figure 7.3 shows examples of the instrument trajectory from an expert and a novice (top row). Each of these trajectories was processed to generate the discretized horizontal deviation and elevation, as well as the CDF projection (normalized to zero mean and unit variance).

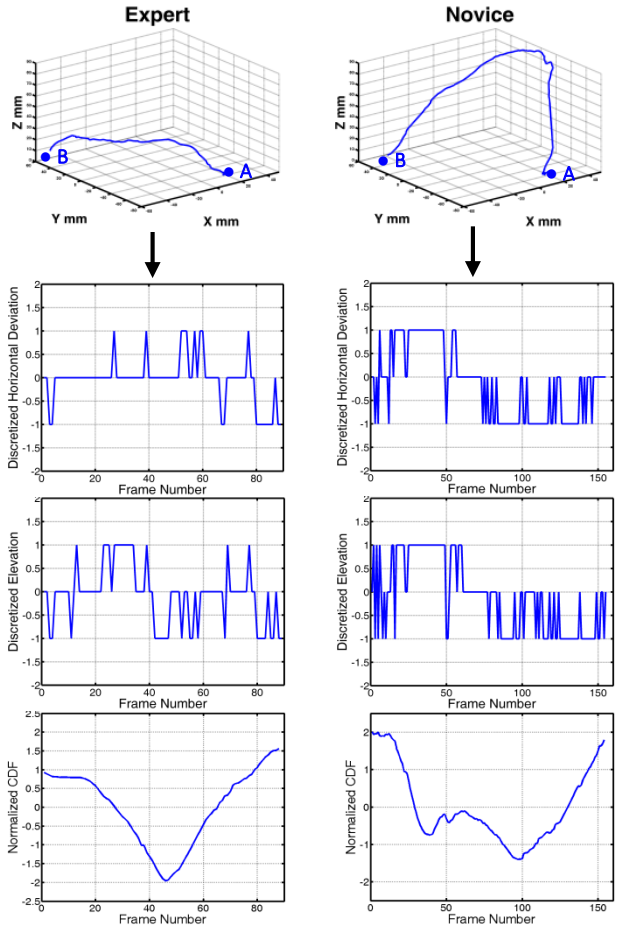


Figure 7.3 Examples of instrument trajectories from expert and novice participants (top row). These trajectories were used to compute the discretized horizontal deviation, elevation, and normalized CDF (see text for details).

The former two and the latter were used as input to the discrete and Gaussian mixture HMM respectively, as described in the Methods. Looking at the discretized deviation and elevation graphs it is clear that in contrast to novices the experts remain within the tolerance band ([-0.7,0.7] mm) for longer time. For the horizontal deviation this means that experts move the instrument tip almost in parallel to the line connecting the two spheres. For the elevation it is clear that novices required to pull the instrument for achieving better depth perception before reaching the second (distant) sphere, which consequently resulted in longer intervals outside the tolerance band in the z direction.

Table 7.3 summarizes the results produced after classification of the trajectories with the three different approaches d (DTW, $HMM_{discrete}$, and $HMM_{Gaussian}$). Three states were used as the optimum order of the model which was determined by the Bayesian information criterion ([190], see Fig. 7.4).The results essentially show the average success rates over all trajectory sessions each of which was classified with a leave-one-out cross-validation.

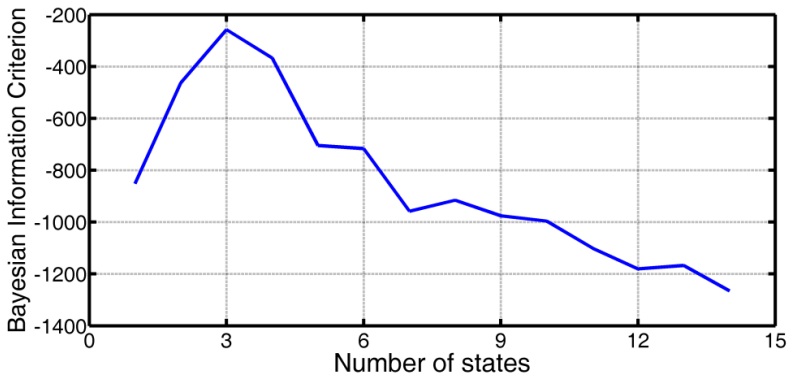


Figure 7.4 Model order selection using the Bayesian information criterion.

All three methods demonstrate a significant recognition of surgical performance with rates ≥80%. HMMs seem to outperform the DTW-based classification generating a classification score greater than 93%. This may well be due to the increased ability of the HMM to model the movement trajectory under the pre-assumed observation framework (discrete/Gaussian mixture).

Table 7.3 Classification Results

Method	Sensitivity	Specificity
Dynamic Time Warping	85	80
HMM (discrete outputs)	95	93
HMM (Gaussian outputs)	94	96
Variance estimates ≈ 2%		

7.3 Discussion of study results

In this study we investigated the potential use of AR in the field of laparoscopic training and assessment. Our results demonstrated the potential benefits that AR provides, indicating that objective assessment of psychomotor skills is feasible through the implementation of effective AR training scenarios. Accurate trajectory recording and real-time occlusion handling were achieved using ARToolkit markers to obtain the exact positioning of the laparoscopic tool w.r.t the simulation environment. Analysis of the instrument trajectory via HMMs provided a robust method for classifying expertise in laparoscopic surgery. Overall, this study provided promising results regarding the potentials of AR, indicating however a significant limitation of our framework design that derived from the use of pattern-markers for instrument tracking. Although tracking accuracy was sufficient, using pattern-markers reduced the trainees' field of view. Additionally, this method produced limitations regarding the movements a trainee could apply, since the markers should always be visible from the camera. Considering these two significant drawbacks,

as a next step in our research and development process, we decided to focus on a more flexible instrument tracking method, as presented in the following Chapter.

Chapter 8

Implementation and Evaluation of Image-based Instrument Tracking

Based on the conclusions of the study presented in the previous chapter, during the second development stage we focused our attention on the development and evaluation of an image-based instrument tracking technique for introduction into our framework. Using this technique combined with real-time AR rendering, we were able to implement laparoscopic training scenarios focusing on depth-perception and hand-eye coordination, two fundamental skills of laparoscopic surgery. To assess the efficiency of the instrument tracking method itself, as well as the potential use of image-based tracking in an AR surgical simulator, a trial study was conducted. This Chapter provides a detailed description of our method, and illustrates the results of the aforementioned trial study.

8.1 A short review of image-based laparoscopic instrument tracking

In computer-aided surgery a number of research methods, prototypes, and commercial systems has utilized AR technology to enhance organ visualization [77, 191], treatment planning [99, 192] and tool positioning [128, 134]. In surgical education the incorporation of similar technological advances has been rather limited, despite their potential to elevate the quality of training. The very few AR simulators currently available [135, 193] lack the ability to interact with the graphical objects, which are simply superimposed over the camera image. Other issues such as pose and occlusion consistency between the virtual and real objects are rarely addressed, although they ultimately affect the educational value of the simulator.

A bold step to overcome these limitations is to address the problem of real-time tracking and 3D pose estimation of the instrument. In laparoscopic simulation this has been approached with the use of motion sensors, such as optical [116] or electromagnetic [194, 195], attached to the instrument. However, these studies have been largely focused on the objective assessment of trainees' performance, rather than the development of AR tasks [137, 196, 197]. Moreover, attaching sensors to delicate equipment such as the one encountered in MIS imposes some ergonomic and operational constraints.

Computer vision constitutes a valuable alternative as it is based solely on visual information obtained from the camera. Wei et al.'s [198] was one of the first studies suggesting the use of a color mark fixed on the instrument shaft. In a similar manner Tonet et al. used a colored strip at the distal part of the shaft to facilitate image segmentation [199]. A pose estimation method was proposed based on the estimation of the apparent width and orientation of the tool. However, the reporting results concerned only derivation of experimental curves for tuning the algorithm parameters rather than measuring the accuracy of tracking and pose. Doignon et al. proposed an algorithm based on three optical markers (LEDs) added on the instrument tip [200]. The method was based on Haralick's & Sharipo's method on pose recovery with $n \geq 3$ known collinear points [201]. In the same direction, Shin et al. used four band-type fiducial markers for measuring 3D pose as well as the opening angle of the endoscopic tool [127].

Canno et al. suggested the derivation of some geometrical features of the tool [117, 119], but they did not describe how these features could be technically extracted from the image data. Allen et al. proposed a method for instrument detection via thresholding, and pose estimation based on the vanishing point of the shaft's edges [118]. Voros et al. relied on the known position of the instrument's insertion point with respect to the camera [202]. Using thresholding and a variant of Hough transform, they were able to demonstrate successful results on static surgical images. A potential limitation of this method is that although the trocar is positioned firmly on the abdominal wall, the instrument may make transversal movements inside the trocar. Recently, a method employing a rectangular pattern marker attached to the instrument tip was presented by our group [157]. Although this approach provided acceptable results in terms of specificity, sensitivity

was low due to the size of the marker required for being detectable (a big marker increases sensitivity but simultaneously obscures the field of view), and the insertion angle of the instrument (the marker cannot be always detectable due to the insertion angle and continuous rotation of the shaft). Although the aforementioned image-based techniques provide important insights into the tool detection problem [117–119, 127, 157, 198, 199, 202], the accompanied results are mainly focused on static frames and not on real-time applications where a performer is involved into a surgical task.

8.1.1 Instrument tracking in the presented AR framework

Moreover, important issues arisen when the tool is lost or mistracked are not addressed. For the presented framework, we implemented an integrated methodology for instrument tracking and pose estimation with emphasis to AR applications in laparoscopic training. The tracking algorithms are designed to operate under illumination variation, partial occlusion of the tool, and a moving camera. Though important, these conditions are also under-examined in the related literature. Our approach consisted of two main algorithms. The first tracker segments a color marker attached to the distal end of the instrument and then obtains its position with respect to the camera coordinate system. In case the marker is lost or mistracked a recovery process is activated. The second algorithm runs concurrently with the first one and targets the line detection of the two edges of the instrument shaft (edge-lines). This method applies a combined approach based on Kalman filtering and Hough transform (Hough-Kalman approach). The depth position of the tip is calculated from the scaling of the segmented color marker. To obtain the angle between the instrument and the camera plane we propose a method that utilizes geometrical features extracted from the edge-lines.

8.2 Method description

8.2.1 Experimental Setup & Camera Calibration

The experiments were performed using a thoracic trainer (size: 50 × 30 × 20 cm3, Annex Art, Anglesey, UK), and an endoscope (¼″ 3CCD) connected to the digital processor of a laparoscopic tower (OTV-SP1, Olympus, DE). The pixel size at the image plane was 0.8 × 0.8 μm. Lighting was provided from a Xenon light source connected to the endoscope (Olympus E180). A strip of color marker (size: 6 × 18 mm2) was attached to the distal end of the shaft of the endoscopic instrument. The marker was selected to have as uniform color, low specularity and Lambertian reflectance as possible. Among various materials tested, a surgical instrument identification color tape was found adequate for this purpose.

For data acquisition we used a multipurpose frame grabber (UFG-05 32, Unigraf, Finland) connected to the digital processor. The image data were transferred through an S-video channel at resolution 720 × 576. Figure 8.1 shows a general overview of the experimental setup.

Prior to processing, the endoscopic camera was calibrated using the OpenCV library [203]. Based on a pinhole camera model and a standard chessboard grid, we obtained the parameters of the camera geometry (f_x, f_y the horizontal and vertical focal lengths respectively, and c_x, c_x the horizontal and vertical displacements of the optical axis from the center of coordinates) and a distortion model of the lens. The pixel coordinates were rectified prior to processing based on the derived distortion of the lens, which was crucial since the ultimate goal was the continuous estimation of the instrument's location and pose in physical units.

Figure 8.1 The experimental setup used for performing the simulation experiments.

8.2.2 Marker Tracking

To localize the color marker we designed a tracker that attempts to position a subwindow (mask) in the image so that the center of the marker remains always inside. This process aims to find the pixels belonging to the marker (called its *support* after [204]) by limiting the search inside the mask, which facilitates computational processing. In the following paragraphs we describe how the algorithm creates an adaptive model of the marker's color.

8.2.2.1 Color Marker Model

The tracker starts by manually sampling some pixels from the marker so as to obtain the initial color values of its support (mean, $\boldsymbol{\mu_0}$, and covariance, $\boldsymbol{\Sigma_0}$). Here, we used the HSI (hue, saturation, intensity) color model since RGB results in a non-uniform chromaticity scale [205]. HSI is also related to the psychological perceptual attributes and in the past was found adequate for the segmentation of surgical instruments [206]. The color density of a pixel $\boldsymbol{x} = [h, s, i]^T$ in the mask window O_t is modeled as a Gaussian distribution:

$$p(x/O_t) = \frac{1}{2\pi^{1.5}|\boldsymbol{\Sigma_t}|^{0.5}} e^{-0.5(\boldsymbol{x}-\boldsymbol{\mu_t})^T \Sigma_t^{-1} (\boldsymbol{x}-\boldsymbol{\mu_t})} \tag{8.1}$$

where the subscript t denotes a particular time step, implying that the mask location and the Gaussian parameters change with time (more details are provided in the next section).

In the HSI color space the distribution of the marker's color is represented by an ellipsoid. The shape and location of this ellipsoid depends on the covariance matrix $\boldsymbol{\Sigma}$ and the mean vector $\boldsymbol{\mu}$ respectively. From singular value decomposition (SVD), $\boldsymbol{\Sigma}$ may be realized as the outcome of a rotation $\boldsymbol{V}$, a scaling $\boldsymbol{U}$ along the rotated coordinate axes and a second rotation $\boldsymbol{S}$ of a unit sphere, so that:

$$\boldsymbol{\Sigma = U\, S\, V} \tag{8.2}$$

where the diagonal entries of $\boldsymbol{S}$ and the columns of $\boldsymbol{U, V}$ are the eigenvalues and eigenvectors of $\boldsymbol{\Sigma}$ respectively. Following the inverse process, a color pixel is regarded as being in the marker's support if it belongs inside a sphere of radius r [204]:

$$r = \|\boldsymbol{S}^{-1}\, \boldsymbol{U}^{-1}\, (\boldsymbol{x} - \boldsymbol{d})\| \leq r_0 \tag{8.3}$$

where $\|\ .\ \|$ denotes the norm operator, $\boldsymbol{d}$ is the translation of the ellipsoid from the origin (i.e. the mean vector $\boldsymbol{\mu}$), and r_0 denotes how many standard deviations the ellipsoid accounts for the marker's support. In this paper $r_0 = 2$, which is equivalent to ≈ 80% of the total distribution in a 3D space. The SVD process is repeated for each frame, since the color properties of the marker change with the instrument movement. Consequently, Equation 8.3 is applied with new values each time, as a result of the updated estimates for $\boldsymbol{\Sigma}$, $\boldsymbol{\mu}$. This adaptation process is discussed in the next section.

8.2.2.2 Adaptive Color Update

In contrast to other works [198, 199] that consider uniform color properties of the tip marker, here we employ an adaptive process since the marker's color undergoes significant variations while the instrument/endoscope moves inside the trainer box. Thus, setting the matrices $\boldsymbol{S}$, $\boldsymbol{U}$ in Equation 8.3 to fixed values would not be practical for this application. Figure 8.2 shows an example of how the mean color values change while the instrument slowly moves away from the endoscope. These measurements were obtained from a region of interest drawn on the marker. To calculate the distance from the camera we employed a pattern marker attached to the

instrument shaft (for further details see the Results section). It may be seen that while the hue component remains almost constant, saturation and lightness change by 40% and 70% respectively, over a 12 cm distance.

To resolve this variability issue, the marker's color was modeled dynamically by updating the model parameters based on the changing appearance of the marker. In particular, the initial marker's support is found from the parameters μ_0, Σ_0. The rectangular mask enclosing the marker is then calculated as the marker's centroid. In our formulation the size of the mask is fixed, although this may easily vary according to the size of the marker's support. The window is selected so that the marker is always enclosed, but not too large to deteriorate performance.

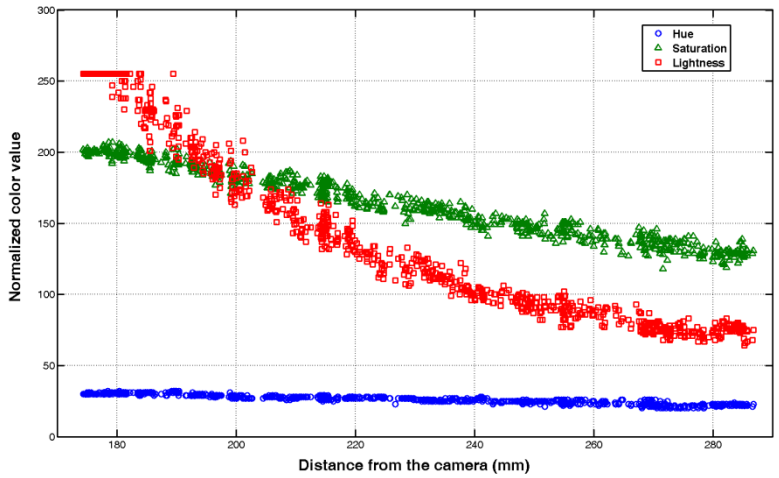


Figure 8.2 Color variability of the marker in relation to its distance from the endoscope.

In each subsequent frame the predicted set of pixels belonging to the marker, Y_t^-, is set to the marker's support, Y_{t-1}^+, found at the previous step (the superscripts - and + denote predicted and corrected estimates respectively). Note that although Y_t^- and Y_{t-1}^+ refer to the same pixel locations, they do not necessarily have the same color properties due to the inherent image noise and camera/instrument motion. The predicted model parameters $\boldsymbol{\mu}_t^-, \boldsymbol{\Sigma}_t^-$ are calculated from Y_t^-. The corrected model parameters are obtained from the L_t+1 most recent predicted estimates:

$$\boldsymbol{\mu}_t^+ = \frac{1}{L_t+1}\sum_{\tau=t-L_t}^{t} \boldsymbol{\mu}_\tau^- \tag{8.4}$$

$$\boldsymbol{\Sigma}_t^+ = \frac{1}{L_t+1}\sum_{\tau=t-L_t}^{t} \boldsymbol{\Sigma}_\tau^- \tag{8.5}$$

These equations obviously require averaging of L_t+1 matrices for each parameter. After some algebraic manipulation one may obtain the following recursive form that makes processing more efficient:

$$\boldsymbol{\psi}_t^+ = \boldsymbol{\psi}_{t-1}^+ + \frac{1}{L_t+1}(\boldsymbol{\psi}_t^- - \boldsymbol{\psi}_{t-L_t-1}^-) \tag{8.6}$$

where $\boldsymbol{\psi}$ denotes either of $\boldsymbol{\mu}$ or $\boldsymbol{\Sigma}$. Hence, to calculate the corrected model parameters only three quantities are required in each time step: the previous recursive estimate, $\boldsymbol{\psi}_{t-1}^{+}$, the estimate based on the new data, $\boldsymbol{\psi}_{t}^{-}$, and the oldest estimate, $\boldsymbol{\psi}_{t-L_t-1}^{-}$. The corrected marker's support, Y_t^{+}, is found by searching all pixels inside O_t that satisfy Equation 8.3, using as model parameters the corrected estimates given by Equation 8.6. In the next frame the mask window, O_{t+1}, is repositioned according to the centroid of the updated region of the marker, Y_t^{+}.

The parameter L_t is updated so that the mean vectors $\boldsymbol{\mu}_t^{-}$ of the L_t+1 most recent frames are approximately uniform (less than 5% color difference) and the maximum past history was set equal to the frame rate of the imaging system. The pseudocode of the color marker tracking algorithm is given in Fig. 8.3.

repeat
%Calculate new model parameters from Y_t
$\boldsymbol{\mu}_t := \boldsymbol{\mu}_t^{-}$; $\boldsymbol{\Sigma}_t := \boldsymbol{\Sigma}_t^{-}$;
%Calculate recent history to consider
$\boldsymbol{L} := \boldsymbol{L}_t$;
%Update model parameters, Eq. (6)
$\boldsymbol{\mu}_{t+} := \boldsymbol{\mu}_t^{+}$; $\boldsymbol{\Sigma}_{t+} := \boldsymbol{\Sigma}_t^{+}$;
%Perform SVD
$\boldsymbol{S} := \boldsymbol{S}_t$; $\boldsymbol{U} := \boldsymbol{U}_t$;
%For all pixels in mask O_t *update marker's support, Eq. (3)*
$Y_t := Y_t^{+}$;
%Calculate the centroid of Y_t
$c_t := c(Y_t)$;
%Update recursive parameters; reposition search mask
$t := t + 1$; $\boldsymbol{\mu}_{t-1}^{+} := \boldsymbol{\mu}_{t+}$; $\boldsymbol{\Sigma}_{t-1}^{+} := \boldsymbol{\Sigma}_{t+}$; $\boldsymbol{\mu}_{t-L_t-1}^{-} := \boldsymbol{\mu}_{t-L}$;
$\boldsymbol{\Sigma}_{t-L_t-1}^{-} := \boldsymbol{\Sigma}_{t-L}$; $Y_t := Y_{t-1}$; $O_t = O(c_t)$;
until ($t := t_{end}$)

Figure 8.3 Pseudocode of the color update algorithm used for marker tracking (symbols are described in the text).

8.2.2.3 Instrument Shaft Tracking

Due to its cylindrical shape and uniform color, the instrument shaft can be represented by two straight lines corresponding to the edges of the instrument (edge-lines). To find these lines an algorithm that combines the Hough transform [207] and the Kalman filter [208] was designed. Although these techniques have been widely applied in computer vision, their combined application has received little attention in the literature. The method presented here applies a standard Kalman filter in the Hough space and has been inspired by the work presented in [209].

8.2.2.4 Preprocessing & Hough Space

The tracker is initialized by identifying the lines that correspond to the two edges of the endoscopic tool. First we use the Canny detector to generate the edge features. Although a detailed description of this method is beyond our scope, the main steps include: (a) image gradient computation: $J_{x,y} = \nabla I_{x,y} = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$, (b) computation of edge gradient magnitude and

orientation: $\vartheta = \mathrm{atan}(J_y/J_x)$, (c) detection of local edges using non-maxima suppression, and (d) edge linking using hysteresis thresholding.

To identify the two edge-lines of the instrument a variant of the Hough transform is employed, where for each edge pixel we use its orientation ϑ_i to compute ρ_i [210]. This leads to a faster approximation since the original transform assumes no knowledge of the edge orientation and thus iterates over all possible values of ϑ. The edge-lines are identified as the two highest peaks in the accumulator (here referred to as Hough-lines). Initially, a large part of the instrument is enclosed in the field of view so as not to adventitiously detect irrelevant lines that may also be present in the field of view.

8.2.2.5 Kalman Filtering

A major concern in real-time applications involving the Hough transform is its computational cost and that the highest peaks may not always correspond to the lines of interest. Even if one tries to apply some rule such as detecting the two strongest peaks that correspond to nearby lines, there is no guarantee that unrelated linear structures would be excluded. To overcome these limitations a Kalman filter is designed that tracks in the Hough space the pair of peaks corresponding to the instrument edge-lines. A Kalman filter is a recursive estimate of an object's state based on a series of measurements [211]. Here, the object denotes the instrument, the state is the instrument's location and velocity, and the measurements are the coordinates of the edge-lines in the Hough accumulator. The Kalman filter also provides information about the covariance of the state estimate, allowing inference about the uncertainty in the estimation process.

In each time frame the Kalman filter is provided with the locations of the two instrument lines $(\rho,\vartheta)_{1,2}$. The peaks corresponding to these lines are assumed to be moving with a constant radial v and rotational ω velocity in the Hough space. Each velocity is assumed to be the same for both lines since they refer to the same moving object. The state vector is defined as:

$$\boldsymbol{s}_t = [\rho_{1,t}, \theta_{1,t}, \rho_{2,t}, \theta_{2,t}, v, \omega]^T = [\rho_{1,t-1} + v,\ \theta_{1,t} + \omega,\ \rho_{2,t-1} + v,\ \theta_{2,t-1} + \omega,\ v,\ \omega]^T \quad (8.7)$$

In each frame the measurements are extracted from a subwindow in the Hough space (more details about this window are given in the end of this section):

$$\boldsymbol{z} = [\rho_{1,t}, \theta_{1,t}, \rho_{2,t}, \theta_{2,t}]^T \quad (8.8)$$

The state-space equations describing our Kalman model are given by:

$$\mathbf{s}_t = \boldsymbol{F}\, \boldsymbol{s}_{t-1} + \boldsymbol{w}_t \quad (8.9)$$

$$\boldsymbol{z}_t = \boldsymbol{H}\, \boldsymbol{s}_t + \boldsymbol{u}_t \quad (8.10)$$

where $\boldsymbol{F}$ is the transition matrix inferred from Equation 8.7, $\boldsymbol{H}$ is the measurement matrix, $\boldsymbol{H}$ = $[\mathsf{I}_{4\times4}|\mathsf{0}_{2\times2}]$, and $\boldsymbol{w}_t$, $\boldsymbol{u}_t$ denote the process and measurement noise vectors which are assumed to be white Gaussian with covariance $\boldsymbol{Q}_t$ and $\boldsymbol{R}_t$ respectively.

The task of Kalman filter is divided into two phases: *prediction* and *correction*. In the first phase, corrected information about the system's state in the previous time frame is used to make a prediction in the next frame:

$$\boldsymbol{s}_t^- = \boldsymbol{F}\ \boldsymbol{s}_{\boldsymbol{t-1}}^+ + \boldsymbol{w}_t \tag{8.11}$$

Similarly, the *a priori* estimate for the error covariance is given by:

$$\boldsymbol{P}_t^- = cov(\boldsymbol{s}_t - \boldsymbol{s}_t^-) = \boldsymbol{F}\,\boldsymbol{P}_{t-1}\,\boldsymbol{F}^T + \boldsymbol{Q}_{t-1} \tag{8.12}$$

where *cov(**x**) =* $E[\boldsymbol{x}\boldsymbol{x}^T]$ and $E[.]$ denotes the expectation operator.

In the next phase we make a measurement $\boldsymbol{z}_t$ and then reconcile that measurement with the prediction made in the previous step to update (i.e. correct) our estimate for the state and error covariance:

$$\boldsymbol{s}_t^+ = \boldsymbol{s}_t^- + \boldsymbol{K}_t\,(\boldsymbol{z}_t - \boldsymbol{H}_t\,\boldsymbol{s}_t^-) \tag{8.13}$$

$$\boldsymbol{P}_t^+ = cov(\boldsymbol{s}_t - \boldsymbol{s}_t^+) = (I - \boldsymbol{K}_t\,\boldsymbol{H}_t)\boldsymbol{P}_t^- \tag{8.14}$$

where $\boldsymbol{K}_t$ is the Kalman gain matrix given by:

$$\boldsymbol{K}_t = \boldsymbol{P}_t^-\,\boldsymbol{H}_t^T\,(\boldsymbol{H}_t\,\boldsymbol{P}_t^-\,\boldsymbol{H}_t^T + \boldsymbol{Q}_t)^{-1} \tag{8.15}$$

Equations 8.11 – 8.14 form the prediction-correction cycle applied in each subsequent frame with the aim to obtain corrected estimates for the location of the two lines: $\rho_{1,t}^+, \theta_{1,t}^+, \rho_{2,t}^+, \theta_{2,t}^+$. The corrected estimates are combined with the posteriori error covariance to restrict the line search inside a subwindow $(\rho \pm k\sigma_\rho \times \theta \pm k\sigma_\theta)$, where each of ρ, ϑ denotes the mean of this parameter for the two lines detected inside the window, σ_ξ is the mean variance for the ξ parameter, and k is some constant (k = 2, as proposed in [209]). Hence, the number of candidate edge-lines as well as the computational cost are reduced.

8.2.2.6 Marker Recovery

A common problem in tracking algorithms is the failure to continue tracking the object due to reasons such as excessive speed or occlusion. In our application there are two different objects being tracked; the instrument and the marker. Although instrument tracking was proved fairly robust, a recovery process is activated when the Hough window peaks, corresponding to the instrument edge-lines, drop below a threshold Thr_L. This parameter denotes the minimum number of pixels that is assumed to form a Hough-line. A value of 40 pixels was found adequate for our application. The instrument recovery process starts by searching for a pair of *strong* and *nearby* lines in the entire Hough space. The first requirement is regulated by Thr_L, and the second one by measuring the distance and angle difference, $d\rho$ and $d\vartheta$, between all candidate lines exceeding an augmented Thr_L value. We found that a minimum of $d\rho$ = 50 pixels, and $d\vartheta = 5^o$ was adequate for this purpose.

The color marker was considered as lost when: (a) its support was outside an expected size range: [Thr_{min}, Thr_{max}] = [20, 1000] pixels, or (b) the distance between its center and the median line of the instrument exceeded a predetermined value, Thr_D = 20 pixels. The median line was calculated as the average of the two Hough-lines. The marker recovery is activated as a separate thread to the instrument tracking process. Since the marker should always lie within the two edge-lines, a search is performed in the enclosed image region with the aim to find those pixels belonging to the marker's support. However, applying of SVD and then Eq. 8.2 for all pixels in this region is very time-consuming, so the image is subsampled; at this point only a rough estimate of the support is desired. If the support's size is enough, the adaptive color update proceeds as normal using the estimated model parameters, otherwise the process is repeated.

Finally, tracking of non-instrument or non-marker pixels may also occur (mistracking). These failures may be prevented by increasing the confidence that the Hough-lines and the marker's support belong to the objects of interest. The parameters Thr_L, Thr_{min}, Thr_{max} and the color distance r_0 provide a means to regulate this confidence limit.

8.2.2.7 3D Pose Estimation

The term 3D pose refers to the position and orientation of an object of known geometry relative to a camera coordinate system. The pose of the endoscopic instrument is provided by the position of its distal end, obtained through marker detection, and the orientation of the shaft with respect to the endoscope. In the following we describe how these parameters are computed.

8.2.2.8 Marker Position

A fundamental property of the pinhole camera model is that the apparent length of any line parallel to the image plane is related to its distance to the camera according to the following rule of scaling:

$$\frac{l'}{f} = \frac{L}{z} \tag{8.16}$$

where L and l' are the actual and apparent lengths of this line respectively, z is the distance between the line and the camera and f is the camera's focal length.

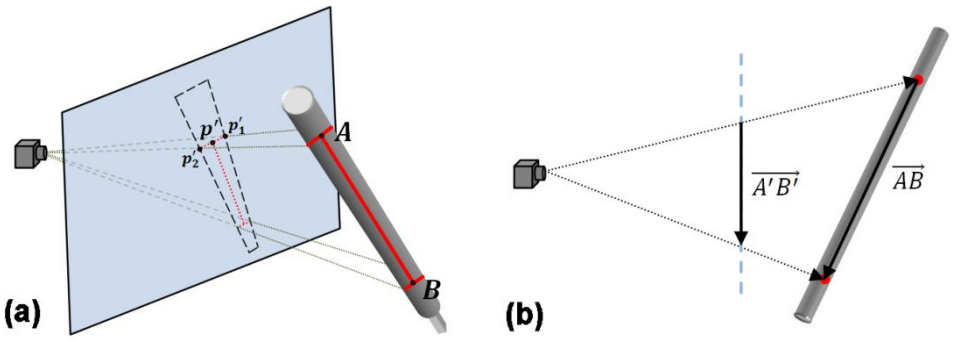


Figure 8.4 Projection of the instrument cross-sections on the image plane (a). The direction vectors $\overrightarrow{\boldsymbol{AB}}$ and $\overrightarrow{\boldsymbol{A'B'}}$ provide the orientation of the instrument with respect to the camera (b).

As depicted in Fig. 8.4A, the apparent width of the instrument can be extracted by the edges of the shaft projected on the image plane. Hence, given the physical width of the marker one may obtain its position using the previous equation. In particular, if $\boldsymbol{p'_1} = [x'_1, y'_1, w]$, $\boldsymbol{p'_2} = [x'_2, y'_2, w]$ define two vertices of the apparent width of the marker and $\boldsymbol{p'} = [x', y', w]$ is the apparent midpoint, for the position of the marker in the physical world, $\boldsymbol{P} = [X, Y, Z]$, we have:

$$X = (x' - c_x)\frac{Z}{f_x} \tag{8.17}$$

$$Y = (y' - c_y)\frac{Z}{f_y} \tag{8.18}$$

and the Z coordinate can be approximated as:

$$Z \approx \frac{f_x f_y L}{\sqrt{[f_y (x'_1 - x'_2)]^2 + [f_x (y'_1 - y'_2)]^2}} \tag{8.19}$$

where L is the physical width of the marker, and f_x, f_y, c_x, c_z are obtained from the camera calibration process. The vertices $\boldsymbol{p'_1}$ and $\boldsymbol{p'_2}$ were obtained by taking the intersection between the midline of the instrument shaft (calculated from the Hough lines) and a vertical line that passes through the midpoint $\boldsymbol{p'}$ of the most extreme, towards the tip, vertices of the minimal rectangle that bounds the marker's support (bounding box). This approach provides an estimation of the depth position of the marker's end, rather than its centroid proposed in [199]. The apparent width of the marker's end is given by the Euclidian distance of the vertices $\boldsymbol{p'_1}$ and $\boldsymbol{p'_2}$.

8.2.2.9 Instrument Orientation

Orientation is usually defined by a standard 3×3 orthogonal matrix that corresponds to three clockwise rotations (Euler angles). For the AR training tasks considered here, the roll angle around the axis of the shaft is not important and hence the problem reduces to calculation of two angles. The first angle, ω, denotes a rotation of the shaft in a direction parallel to the image plane (transversal), whereas the second one, φ, refers to a rotation in a direction perpendicular to the image plane. Rotation ω is essentially given by the angle between the median line of the

instrument and the *x* axis of the image plane. This angle is straightforward to obtain from the Hough-Lines detected with the Hough-Kalman process.

In order to calculate *φ*, we define a pair of distant cross-sections on the Hough-lines: one close to the intersection point of the lines, and the other one further apart (see Fig. 8.4A). A cross-section here is defined as two points that have the same distance perpendicular to the median line of the instrument. Using the method described in the previous section we can derive the real world position of the two cross-sections midpoints $A = [X_A, Y_A, Z_A]$ and $B = [X_B, Y_B, Z_B]$. The direction vector $\overrightarrow{AB}$ of the three-dimensional line that connects A and B is given by:

$$\overrightarrow{AB} = [\, X_B - X_A, Y_B - Y_A, Z_B - Z_A] \tag{8.20}$$

In perspective projection, a pixel $p'(x', y')$ on the image plane is practically a point $p(x', y', f)$ in the camera coordinate system, since the distance to the camera in the *z* direction is equal to the focal length. Consequently, the direction vector of the instrument's projection may be derived via the image plane coordinates $A' = [x'_A, y'_A, f]$ and $B' = [x'_B, y'_B, f]$:

$$\overrightarrow{A'B'} = [\, x'_B - x'_A\,,\, y'_B - y'_A\,, 0] \tag{8.21}$$

where *A'*, *B'* are the projections of *A*, *B* respectively. Figure 8.4B demonstrates vectors $\overrightarrow{AB}$ and $\overrightarrow{A'B'}$ at a top view of the camera space. Since the line *A'B'* is parallel to the image plane, the angle between the direction vectors $\overrightarrow{AB}$ and $\overrightarrow{A'B'}$ represents the desired angle between the instrument shaft and the image plane:

$$\varphi = cos^{-1}\left(\frac{\overrightarrow{AB}\cdot\overrightarrow{A'B'}}{\left\|\overrightarrow{AB}\right\|\cdot\left\|\overrightarrow{A'B'}\right\|}\right) \tag{8.22}$$

Note that this equation does not give a unique solution, so one needs to keep only the solution for which the instruments lie within the field of view of the camera.

8.3 Method evaluation

The following sections present the various experiments performed with regard to instrument tracking, pose estimation, and AR-based training. The AR tasks were designed to allow practice on depth perception and hand-eye coordination. We also present results for the occlusion handling

problem. The algorithms were implemented on a standard 2.9 GHz Core2 Quad PC. Although the goal of this paper is not to describe a complete AR simulator, we illustrate the feasibility of the proposed method in terms of building such a system.

8.3.1 Instrument & Marker Tracking

The outcome of considering only edge features that are near the edge-lines is shown in Fig. 8.5. Using all features in the image resulted in 550 lines, which makes difficult to decide which ones correspond to the shaft. Note that the overall field of view contains also undesired lines such as the edges of the peg-board. With Kalman filtering the search is restricted into a subwindow and the number of peaks is reduced to 60. Mapping this window into the image plane provides a polygon centered around the median line of the instrument as shown in Fig. 8.5(b). The polygon area essentially contains edges that may be part of the peaks inside the Hough space subwindow. The overall effect may be seen more clearly in Fig. 8.6 that shows static frames during instrument tracking. The empty space in the image corresponds to the area outside the Hough space window. Hence, a great amount of irrelevant edges, and consequently lines, are filtered out.

Figure 8.5 Edge detection without (a), and after (b), applying the Hough-Kalman method in a single frame. (c) Detection of the edges of the instrument shaft (edge-lines).

To evaluate the combined application of the two tracking algorithms we carried out two different experiments based on simulation tasks selected from the Fundamentals of Laparoscopic Surgery (FLS) program: peg transfer and pattern cutting [15]. These exercises are designed for training common laparoscopic skills such as depth perception and hand-eye coordination. In the first task the user requires to pick up a series of pegs from the floor and place them into the holes of a pegboard. The second task requires cutting a gauze within a maximum allowable area while the user maintains tension with the other instrument. At the same time another user moves the camera to aid the first one visualize better the area of interest.

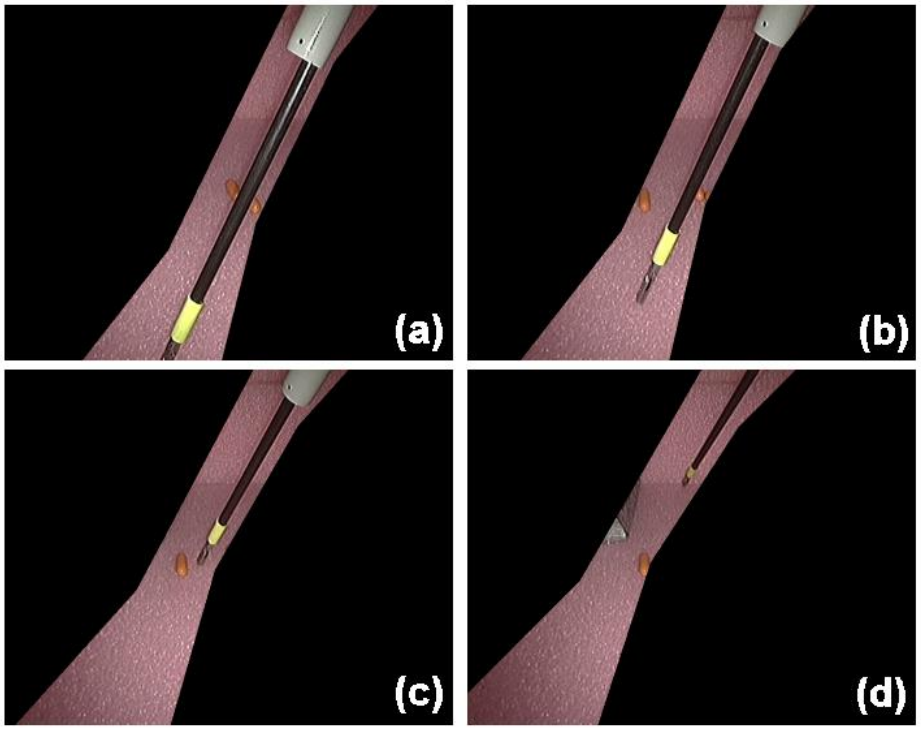


Figure 8.6 The effect of the Hough-Kalman method in the image space. The color area essentially corresponds to the tracking subwindow in the Hough space. Fig. 8.(a)-(d) correspond to static frames as the instrument moves away from the camera.

Both tasks exhibit several challenges such as: continuous rotation and translation of the camera, and instrument movement at variable speeds and rotations. In addition, the marker's color properties also vary with distance from the camera. In Fig. 8.7 we illustrate sample frames during peg transfer. The lines along the edges of the instrument show the outcome of the Hough-Kalman approach, the big rectangle denotes the mask inside which the adaptive model searches for the color marker, the highlighted pixels compose the marker's support extracted from the model, and the small rectangle denotes the marker's bounding box. The algorithm signals that the marker is lost, which is due to the small marker's support detected. The marker's support is partially restored after a few frames, Fig. 8.7(d), and then completely, Fig. 8.7(e). More examples of the recovery process may be found in the accompanied video sequence.

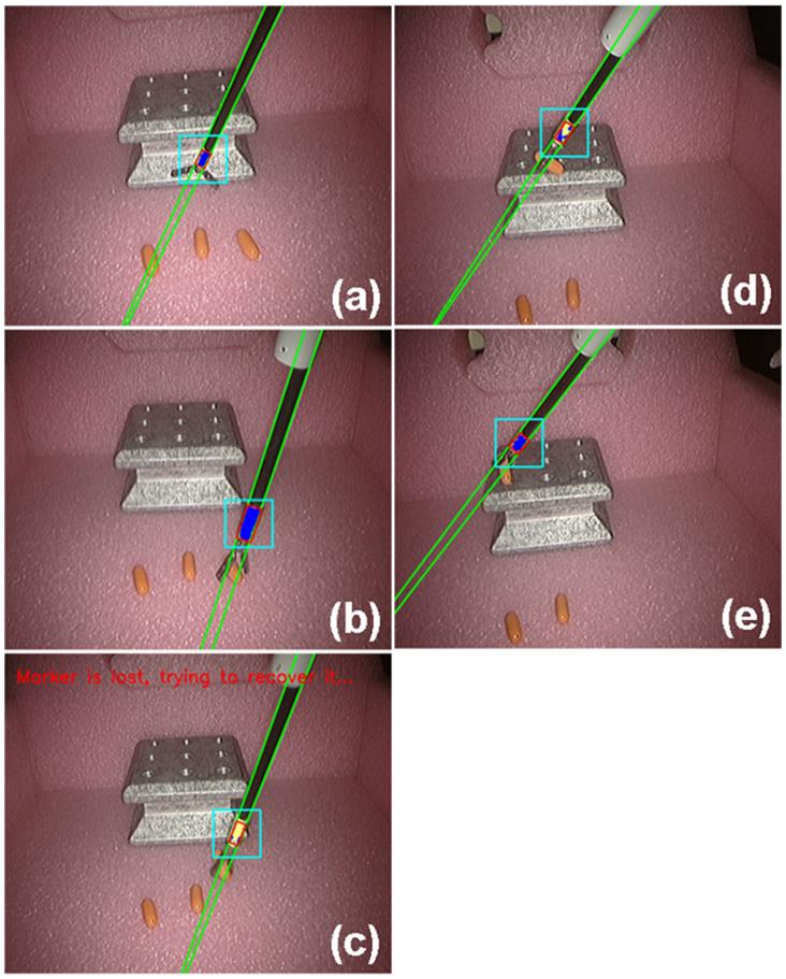


Figure 8.7 Marker and shaft tracking during peg transfer.

Fig. 8.8 shows tracking results for pattern cutting. This task imposes some additional challenges such as: (a) the background is significantly more occluded, (b) the camera and endoscopic tools move continuously with large translations and rotations, and (c) on some occasions the marker is unavoidably occluded by the left instrument. The latter is more clearly shown in Fig. 8.8(e)-(g). Despite the apparent small size of the marker, the tracker localizes it well and its support is completely recovered soon after the occlusion. One may also note that the tracker does not lose the marker completely; some support pixels are detected, but the marker cannot be restored due to the excessive movement of the instrument. The recovery process keeps searching until the marker is fully localized.

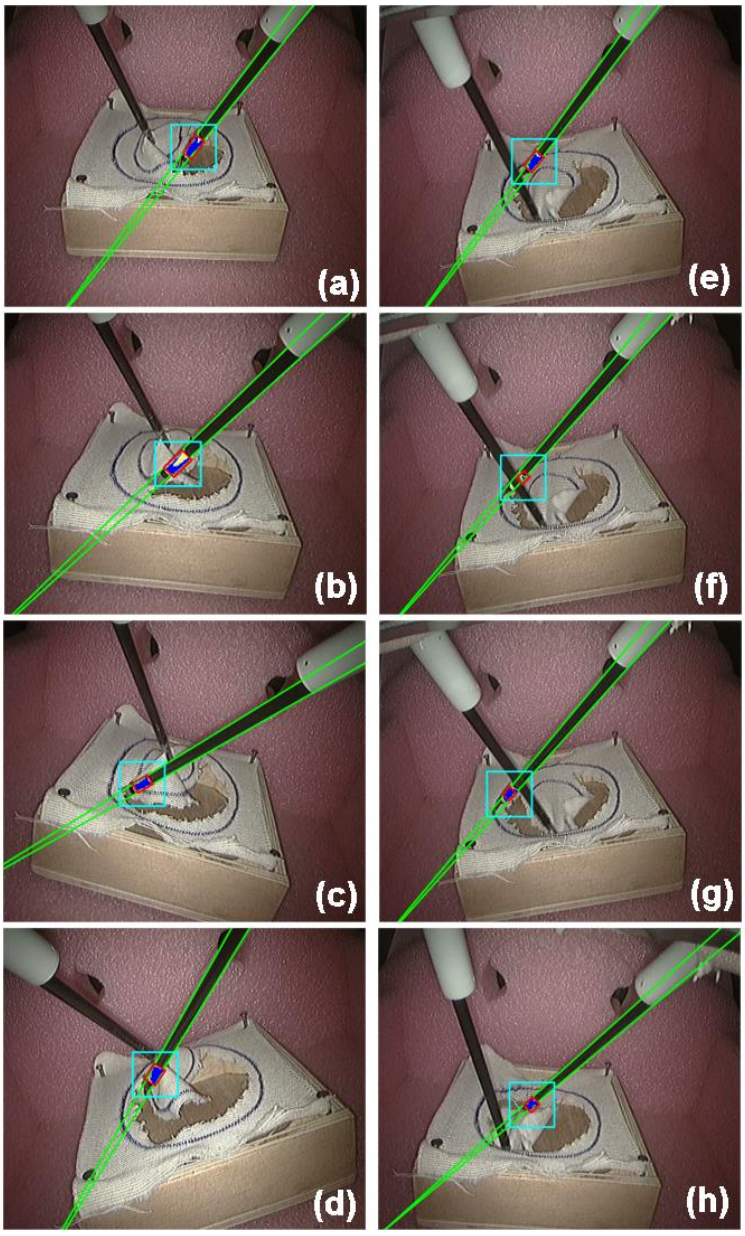


Figure 8.8 Marker and shaft tracking during pattern cutting.

At this point it should be also noted the advantage of applying the Hough-Kalman approach. Despite the presence of the left instrument, the instrument tracker does not get confused. The line search is restricted into a subwindow that encloses the edge-lines of the right instrument. Hence, the edge features corresponding to the left instrument are completely ignored.

8.3.2 Marker Occlusion

To examine further the performance of the tracking algorithm we carried out an additional experiment involving excessive occlusion of the color marker. In particular, while the main user performs peg transfer with the right instrument, another user tries to continuously occlude the

marker using the left instrument. To increase the robustness of the algorithm we implemented an additional Kalman filter to account for the uncertainty of the width, *w*, and height, *h*, of the marker. The state vector in that case was: $\boldsymbol{s}_t = [w_t, h_t, v]^T$, where *v* is the constant velocity shared between the two parameters. Figure 8.9 shows frames from an occlusion occurred during this experiment. To enhance visualization of the tracking results only the area of interest is shown.

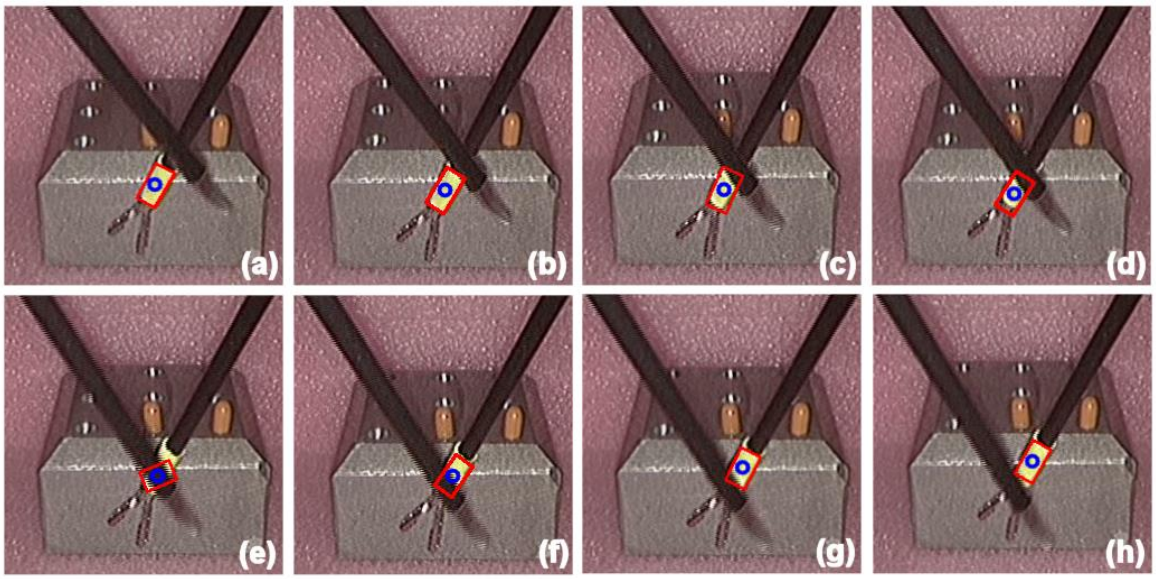


Figure 8.9 Marker tracking during partial occlusion by another instrument.

8.3.3 3D Pose Validation

Before proceeding to the AR tasks it is essential to validate the experimental estimates of the instrument's pose: the *Z* coordinate of the tip (i.e. the marker's end) and the angle, φ. The *X*, *Y* coordinates were not considered as they simply coincide with those of the midpoint of the bounding box's side that is close to the instrument's end. The validation experiment was performed using a pattern marker attached to a fixed position with respect to the color marker as shown in Fig. 8.10. Employing the ARToolkitPlus software library, we were able to calculate the marker's depth position and orientation with submillimeter accuracy (for further information see [79] and the product web page). In particular, the reference values of *Z* (Z_{ref}) were calculated from the angle of the pattern marker (with respect to the camera) and the distance, *D*, between the ARtoolkit marker and the end of the color marker as: $Z_{ref} = Z_{ARToolkit} + D \times sin\varphi_{ARToolkit}$. The reference values of φ were obtained straight from ARToolkitPlus ($\varphi_{ref} = \varphi_{ARToolkit}$). Figure 8.10 shows a single frame from the validation experiment where the small circles indicate the cross-sections used to estimate the experimental values of the angle φ, as described in the Methods. The Hough lines and the bounding box of the color marker used to find the depth position of the tip are also shown.

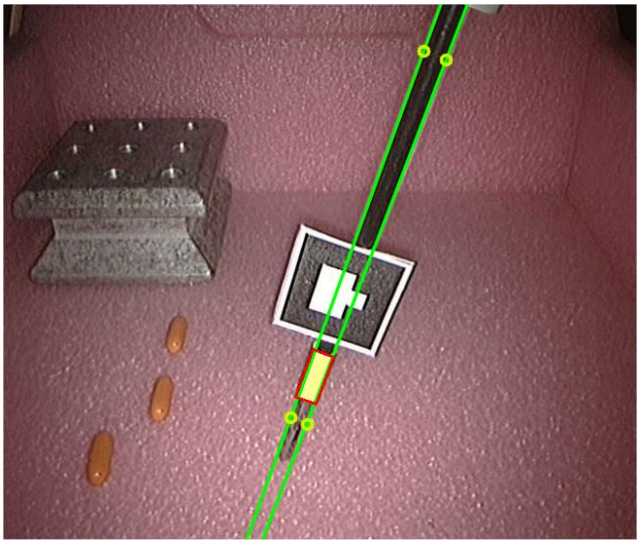


Figure 8.10 Image showing the ARToolkit marker that was used for obtaining the reference estimates of Z, ϕ. The experimental values were derived from the marker's bounding box, and two cross sections of the Hough lines.

In the validation experiment the instrument was moved between a position close to the camera and a distant location inside the thoracic trainer. Note that the instrument was not kept parallel to the endoscope, but it was inserted through the trocar as normal, resulting in a simultaneous variation of the tip position and angle with respect to the camera. Figure 8.11 shows estimates of *Z* for from this experiment. The experimental values are very close to the reference ones, although there is some jitter that increases with distance, as expected. To obtain a better understanding about the significance of this deviation, in Fig. 8.12 we present the absolute errors of *Z* in mm, as well as the corresponding marker width error calculated from the experimental and reference values of the width of the tip ($d_{theor} = f \times L/Z_{th}$, where *L* is the physical width of the marker; see Eq. 8.16). For working distances close to the camera (<17 cm) the average error was found: Z_{err} = 2.5 ± 2.1 mm (± SD), whereas for longer distances: Z_{err} = 6.2 ± 4.5 mm (± SD, 17cm<Z<26cm). From Fig. 8.12 bottom, one can see that with the camera employed this error essentially arises from misestimation of the tip width by almost less than 1.5 pixels. In fact, the average error (in the image plane) across the distance range examined was: w_{err} = 0.6 ± 0.4 pixel (± SD), which approaches the camera resolution. To realize the significance of these results it is recalled that the *Z* values from the ARtoolkit are obtained by using a pattern marker with size 28 × 28 mm^2, whereas the proposed method employs a small color marker the width of which is only 6 mm.

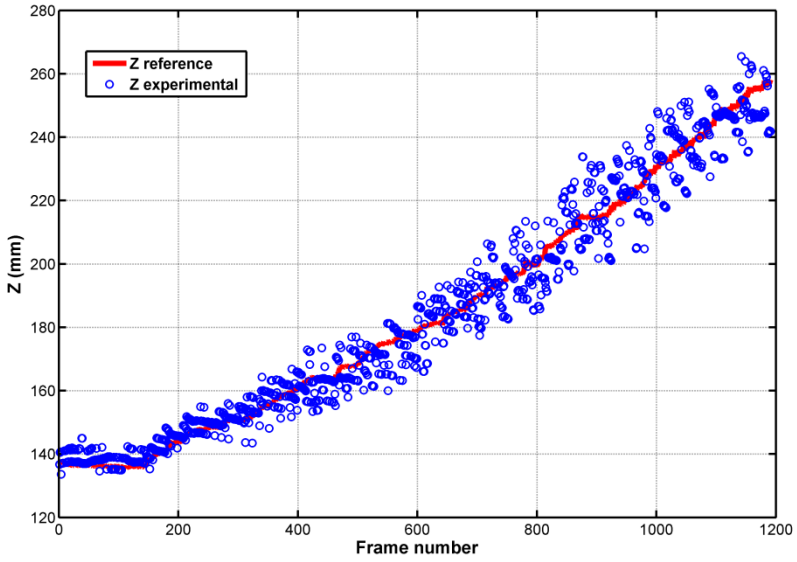


Figure 8.11 Reference and experimental values of *Z* obtained during the instrument's movement inside the trainer.

The values of φ are illustrated in the top graph of Fig. 8.13. It is clear that the experimental estimates appear in good agreement with the reference ones, although the former have a stepwise varying behavior due to the discretized values involved in the calculation of the angle. The bottom graph depicts absolute values of the angle error as a function of *Z*. The average error was: $2^o \pm 1.5^o$ (± SD). The uncertainty is almost uniform across *Z*, something that is not entirely valid for the uncertainty in the estimation of the marker width (see Fig. 8.12). This may be because the width is estimated on image plane coordinates and in distant locations the color marker has an apparent width of only a few pixels that the tracker can easily misestimate leading to a large error.

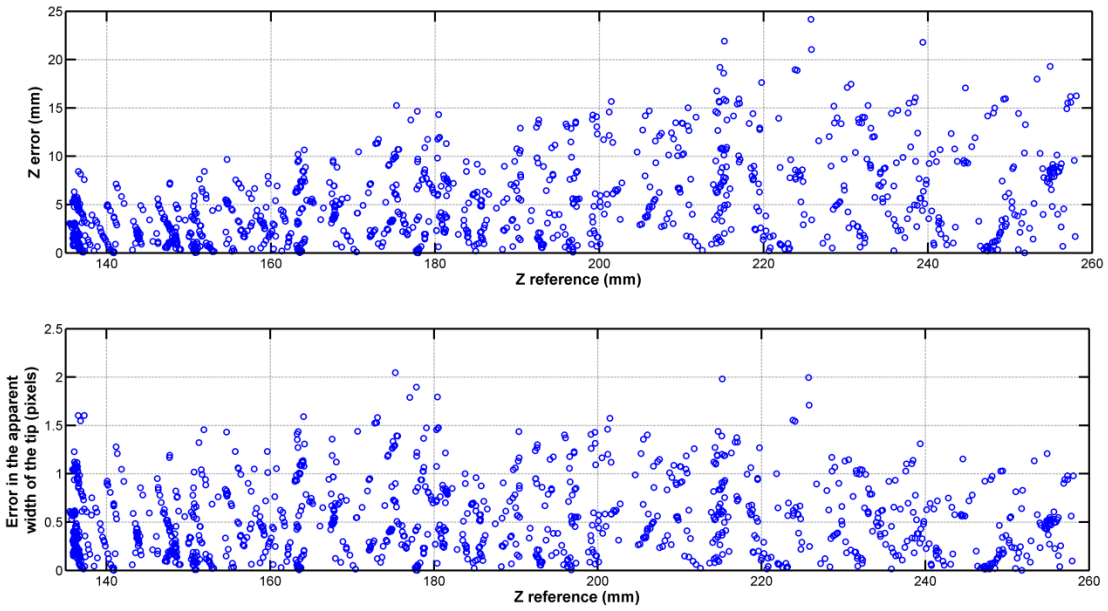


Figure 8.12 Absolute error estimates for: Z (top) and apparent width (bottom), of the instrument tip.

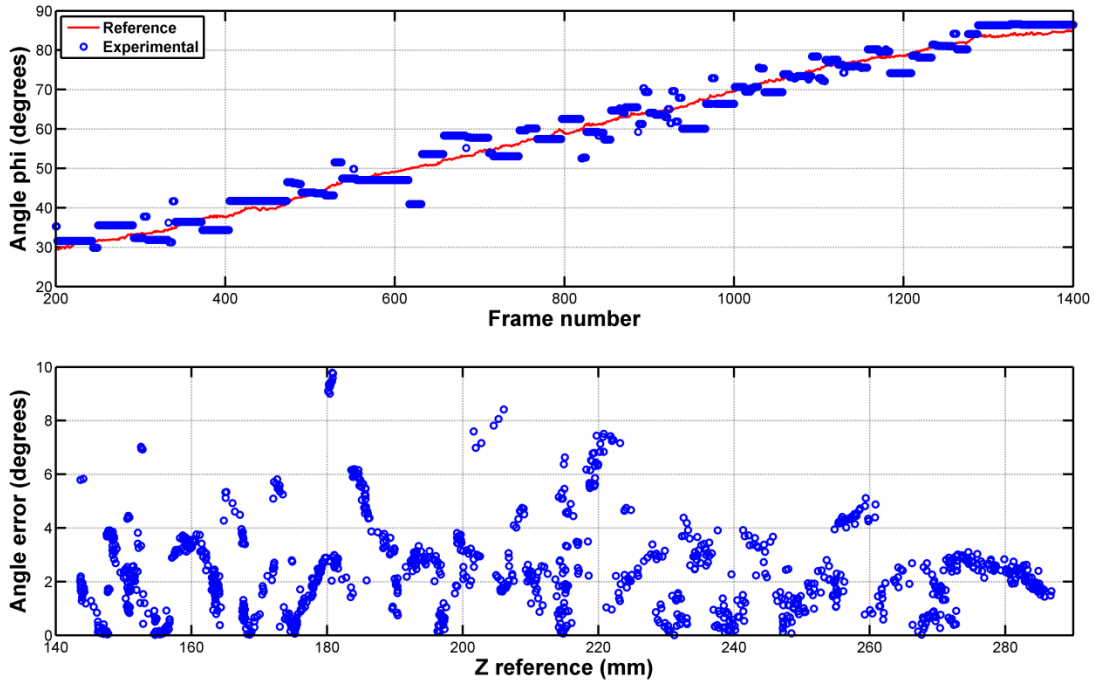


Figure 8.13 Reference and experimental values of the angle ϕ during the motion of the instrument inside the thoracic trainer (top). Absolute error estimates for the angle ϕ as a function of the marker's position from the camera plane (bottom).

8.3.3.1 Approximation in 3D pose validation

As depicted in Fig. 8.14, the apparent width of an object with cylindrical shape is smaller than the projection of its actual diameter. In this case, the apparent radius is given by:

$$R' = R \cdot cos(\varphi) \tag{8.23}$$

where *R*, *R'* denote the actual and apparent radius of the cylinder respectively.

The angle φ can be computed as:

$$\varphi = acos(\frac{\sqrt{D^2 - R^2}}{D}) \tag{8.24}$$

where *D* is the total distance from the camera: $D = \sqrt{d^2 + Z^2}$, *d* denotes the offset from the optical axis, and *Z* is the depth from the camera (the z coordinate). Thus, in the denominator of Eq. 8.19 the quantity: $\sqrt{(x_1' - x_2')^2 + (y_1' - y_2')^2}$ equals the approximated width: 2*R'.

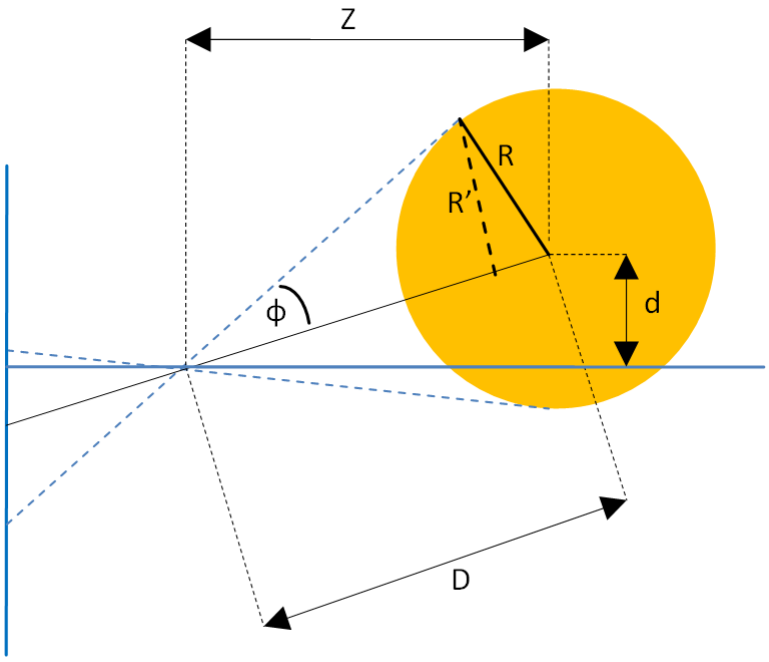


Figure 8.14 Geometrical relation between apparent and true diameter of a cylinder's cross section in perspective projection.

This approximation introduces an error in the estimation of *R* (and consequently *Z*). In our case the instrument's width is 5mm and the examined *Z* ranges: 140 - 260mm. If we consider that *d* may vary between 0 - 30 cm (the width of the pelvic trainer), then the introduced error is: 0.064 -

0.011% and 0.018 - 0.008% for Z = 140 and Z = 260 mm, respectively. Note that the error reduces as the instrument moves away from the camera (in either direction).

8.3.4 AR Applications in Laparoscopic Training

In this section we present two different AR tasks for laparoscopic training. As stated previously, our goal was not to present fully-featured scenarios but rather to develop proof of concept training tasks. The virtual graphics are created in OpenGL. An ARToolkit marker is placed on the floor of the box for providing the 3D pose of the rendered virtual models with respect to the camera.

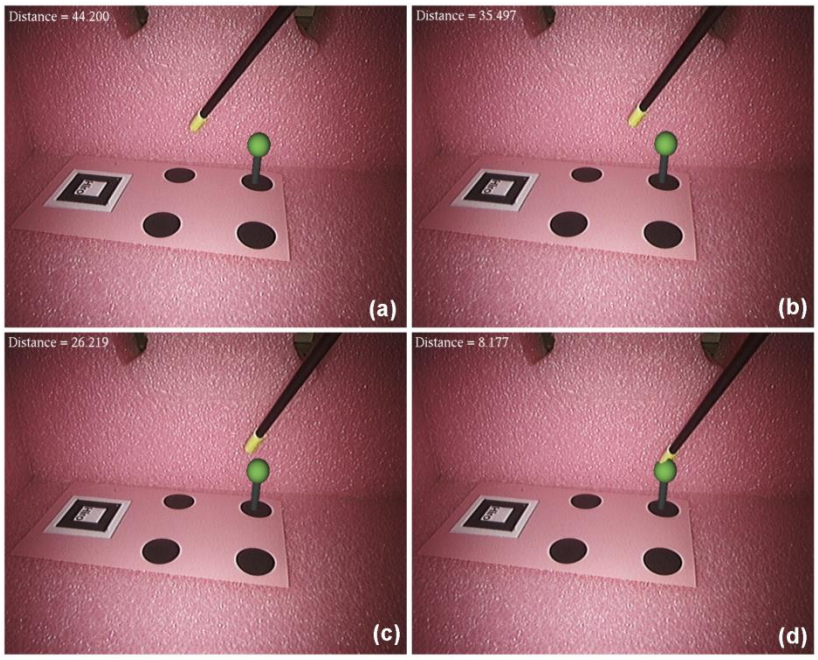


Figure 8.15 Illustration of the sphere touching task. The sphere is approached from behind. The sphere to tip distance is shown in the upper left corner (in mm).

In the first experiment the user is challenged to touch a series of virtual spheres (radius = 8mm) using the instrument tip. The spheres are shown successively in one of four predetermined locations. The sequence and height is selected at random. Figure 8.15 shows a series of frames during performance of this task. The top left corner in each frame displays the distance between the tip and the center of the sphere. When this distance is smaller than the radius of the sphere a new one appears in a different location.

In order to demonstrate the robustness of tracking and depth estimation the task was also performed with simultaneous rotation of the camera. Figure 8.16 shows an example with the instrument approaching a sphere from the front. Here the issue of occlusion was handled by rendering a rectangle in the calculated *Z* location of the tip. The rectangle was defined by two vertices of the bounding box and another two selected from the Hough-lines. Although this was

used only as an approximation, it was found to be adequate for the visualization requirements of these tasks. A more formal approach to the issue of occlusion handling is provided in the next section.

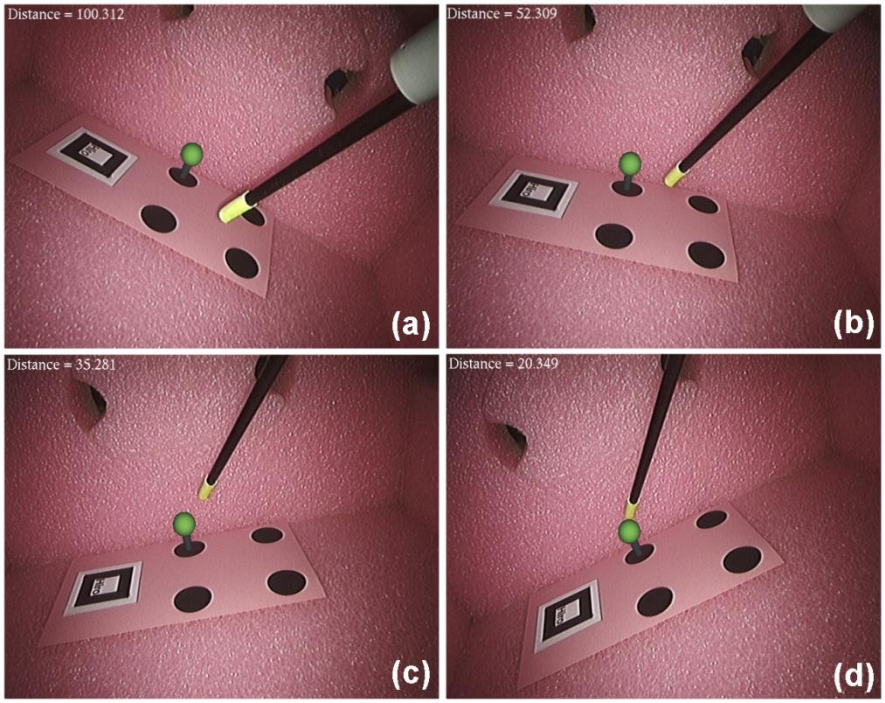


Figure 8.16 Attempting to touch a sphere with simultaneous rotation of the camera.

The second experiment considered a more challenging task that resembles peg transfer. The user is challenged to touch a highlighted sphere and then transfer it to a drop-off location (grid sphere) as shown in Fig. 8.17. As soon as the sphere is touched its color changes notifying the user to begin transferring. In each frame, the top left corner displays the current distance between the tip and either the highlighted sphere, or, once the latter has been touched, the location where the sphere should be transferred to. When the sphere comes close to the target location, determined by a distance threshold, it freezes and the task continues with a new sphere appearing in a different location. It should be noted here that the apparent size and position of the sphere in transfer is not constant but it is rendered according to the 3D position of the tip. This fact implies that if the tip position is not correct, the sphere would appear with a wrong size and/or position during the transfer. However, the rendering is realistic as a result of the robust performance of the tracking and depth estimation algorithm.

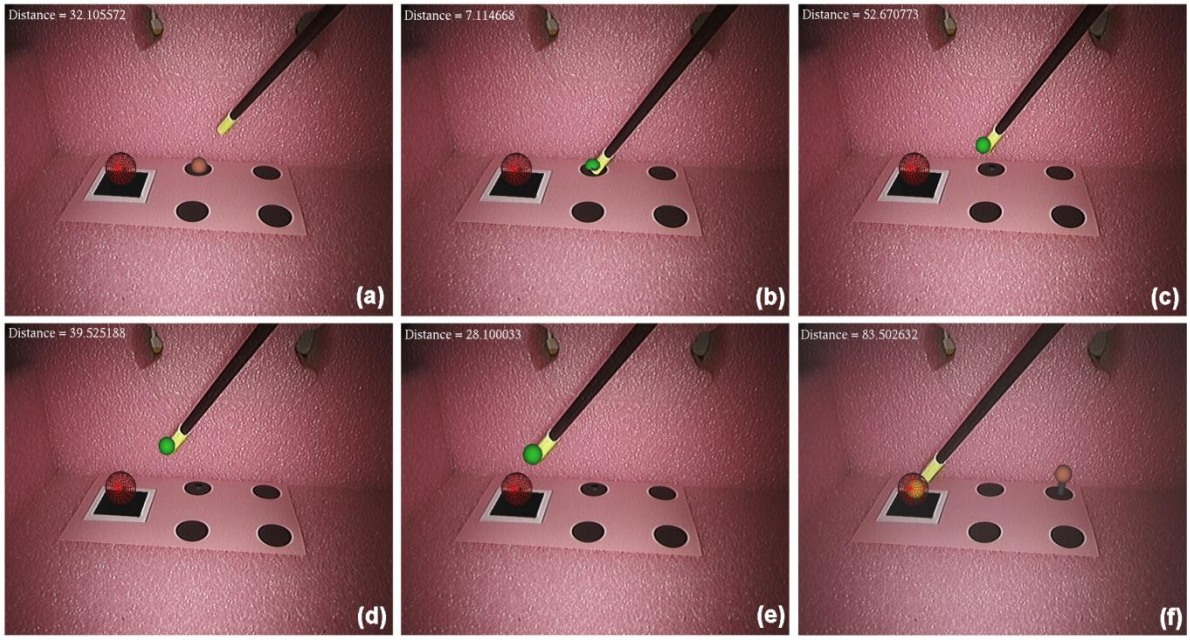


Figure 8.17 Illustration of the sphere transfer task. The sphere (or grid sphere) to tip distance is shown in the upper left corner (in mm).

8.3.5 Occlusion Handling

As stated in the introduction, a critical issue in AR applications is to have both virtual and real objects to coexist seamlessly in the same space. Occluding virtual objects from real ones, when the latter are placed in the foreground, is an inherent problem in AR. To evaluate the proposed method in terms of its occlusion handling accuracy an experiment was performed where a virtual model of a skull was introduced into the camera image using a fixed pattern marker as a reference. Then, the endoscopic tool was navigated inside and around the virtual skull. A pre-constructed virtual model of the instrument was rendered into the image according to its 3D pose as estimated by the proposed method. The virtual representation of the tool essentially served as a mask for properly cropping the virtual skull when the tool was positioned in the foreground.

Fig. 8.18 demonstrates four indicative frames from this experiment. Figure 8.18(a) shows the 3D rendering of the virtual skull and the instrument's mask. In the succeeding frames the mask is not rendered although the instrument's pose was taken into account when rendering the scene. Figures 8.18(b)-(c) illustrate frames in which the instrument is partially occluded by the skull eye sockets. The inset of Fig. 8.18(c) shows a magnified area of the scene where the partial occlusion between the real and the virtual scene is achieved with great detail, providing a highly realistic output. Figure 8.18(d) illustrates how the instrument becomes almost totally occluded when it is positioned inside the virtual skull cavity.

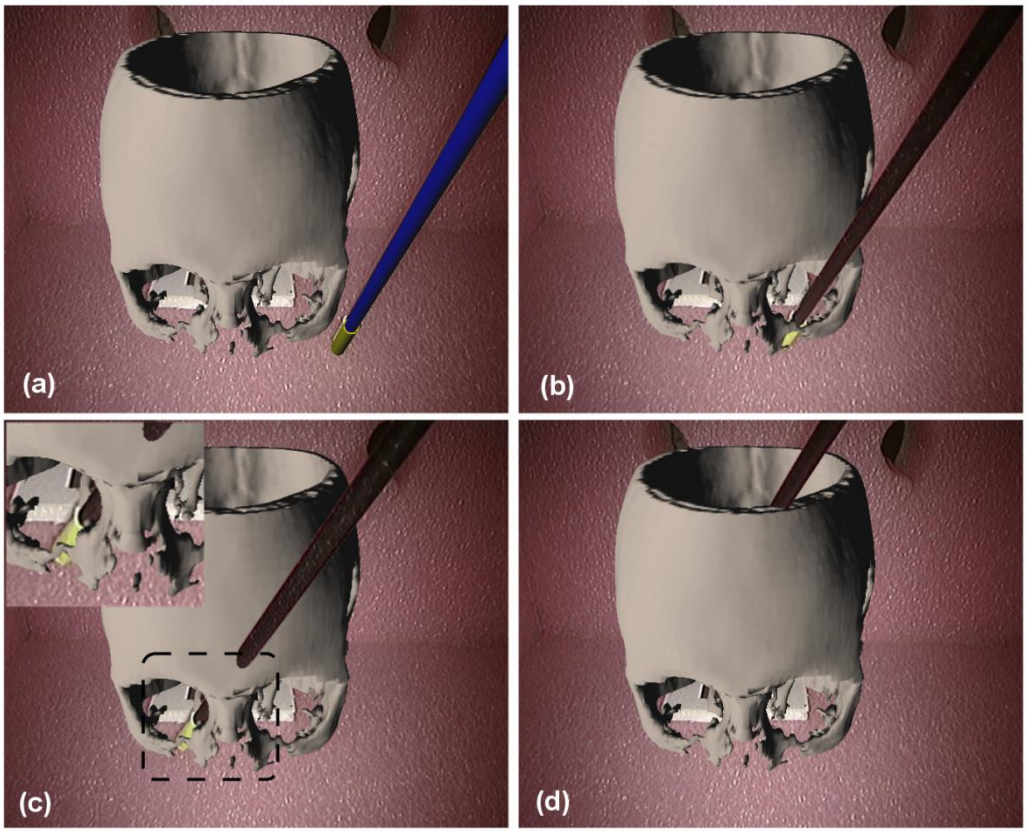


Figure 8.18 Example frames from the occlusion handling experiment. (a) Virtual skull and instrument mask. (b)-(c) The instrument is positioned inside the skull eye sockets. Note the rendering detail achieved in the magnified area. (d) The instrument is positioned inside the skull cavity.

8.4 Discussion of method results

In this Chapter we presented a vision-based algorithm for endoscopic instrument tracking and 3D pose estimation with emphasis in AR simulation training. Compared to other approaches where tracking and pose estimation are treated separately [117, 199, 202], the presented method applied a concurrent approach to both issues. Moreover our method is sensorless and the only requirement is the attachment of a color strip to the end of the tool.

With the Hough-Kalman technique only edge features related to the instrument edges are considered in each frame, improving detection accuracy since the line search is restricted into a subarea of the Hough space. Moreover, the two Hough-lines provide information for calculating the instrument angle with regard to the camera plane, which essentially solves half of the pose estimation problem. The second tracker is based on a color update algorithm that addresses problems such as varying illumination and viewing geometry. These are common conditions in surgical endoscopy where the scene is radiated by a ring source surrounding the endoscopic camera. As the object moves away from the center of the field of view, its chromatic properties

undergo a substantial change. Our experiments showed that the hue component remains relatively constant, but using only one parameter may not be adequate for marker detection, especially in occluded environments.

The *z* coordinate of the marker is determined by the relative change of its width. Alternatively, the height or even the entire size of the marker may be used. The Z position of the camera tip was estimated with an error between: 0.4-17 mm ($mean_{near}$-SD_{near}, $mean_{far}$+SD_{far}) across a respective distance range (near-far): 13-26 cm. In a typical laparoscopic task though the surgeon operates with the camera positioned close to the area of interest, usually less than 20 cm, where the absolute position error is limited to ≈ 2.5 mm. This makes the proposed method promising, especially if one considers that the only information used is the 6 mm width of the color marker. Considering the educational purpose of the AR tasks developed, these results may be adequate for training applications involving the practice of basic skills such as 2D to 3D perception and hand-eye coordination.

It should be noted here that the error in the estimation of the width of the tip may not be entirely a fault of the tracking algorithms. We have noticed that due to the cylindrical shape of the instrument the edges of the marker appear with a significantly darker color compared to the rest of its body. Hence, these peripheral pixels are misinterpreted as not part of the color marker, leading to a minor underestimation of its width (<1.5 pixels). However, due to the small size of the marker, and its distance from the camera, this minor underestimation caused an increasing depth position error. A possible remedy may be to adjust a color triangular prism around the tip so that light scattering, and thus color distribution, becomes more uniform across the marker area. Another error factor is that in the validation experiment we assumed that the two markers are positioned along the same line. This is not entirely correct though as the ARToolkit marker was actually placed on the surface of the instrument's shaft. Hence, there is a minor offset between this marker and the center (in the z direction) of the color marker, which was not taken into account in the estimation of the depth position, and consequently the width of the tip.

During our experimentations, we have also investigated the problem of occlusion consistency (not to be confused with marker occlusion) between real and virtual objects. Addressing this issue required two pieces of information: the 3D position of the instrument tip, which was discussed earlier, and the instrument orientation with respect to the camera. The latter was estimated with a method that yielded an accuracy of about 2^{o}. In practical terms, a more precise estimation of this parameter is very difficult to achieve, considering the discretized nature of the data involved in the calculations. This data essentially arise from two discretization procedures: image digitization during acquisition and Hough space construction. Yet, the second procedure uses the output of the first one, which unavoidably increases the error. A potential remedy would be to use a higher resolution camera, or increase the resolution of the Hough-space, but at the cost of lower frame rate during tracking. Despite this limitation, our approach showed a good consistency with the gold-standard measurements. A visual experiment was performed, illustrating that the

accuracy of the estimated pose is adequate for achieving a correct visual effect, and consequently a realistic augmented reality scene.

In terms of speed performance, our implementation achieved near real-time efficiency (21 Hz) based on Hough space with moderate resolution (1 pixel × 0.5 degrees). Hough space resolution is computationally expensive and it heavily affects the algorithm's speed. Although we experimented with higher resolution values in order to achieve better accuracy, it was concluded that the gain in accuracy was disproportionally small (almost negligible) compared to the decrease of the frame rate. Although in this Chapter we present experimentations using a single laparoscopic instrument, the instrument tracking algorithm can be easily extended to track two instruments at a time, with the only requirement to attach a different color marker to the tip of the second instrument.

Overall, the presented method proved sufficient for the realization of simple AR training tasks similar to those described in this Chapter, targeting on depth perception and hand-eye coordination. However, this method does not provide information regarding important parameters such as opening/closing of the tip and rotation of the shaft around its axis. As stated earlier, our ultimate research goal included the implementation of a richer set of training scenarios that would address important surgical skills such as bimanual instrument interaction. Implementation of these tasks required complex interactions between instruments and VR components of the training scene, such as grasping and lifting. To simulation these actions, we introduced real-time physics calculations into our framework, using the module presented in Chapter 6. As also stated in this Chapter, such actions could only be simulated with an instrument tracking method that would provide information regarding all the D.O.F of the laparoscopic instruments. Based on this fact, during the third research and development stage we decided to integrate sensory devices into our surgical simulation framework. The technical details and results regarding this integration are presented in the following two Chapters.

This page has been intentionally left blank

Chapter 9

Integration of EM Sensors in our Simulation Framework

In order to implement sensor-based instrument tracking for our simulation framework, we developed a calibration method for extracting the spatial relationship between a laparoscopic instrument and an electromagnetic (EM) tracking device attached on the instrument's handles. Development of this method proved a step of instrumental significance towards the fulfillment of the final version of our framework, since it allowed efficient utilization of EM sensors for accurate instrument tracking. The outcome was a very robust instrument tracking setup, providing sufficient accuracy, stability, and frame rate efficiency, allowing the integration of real-time physics into our AR surgical simulation framework.

9.1 Introduction to sensor-based instrument tracking

Instrument tracking is a critical part for every laparoscopic training platform. In VR or AR simulators, knowledge of the laparoscopic instrument pose is essential for implementing training scenarios that provide realistic interaction between the instrument and virtual objects, such as pegs or rings [76]. In PR systems, information regarding the movements of the instrument shaft or tip is also essential for obtaining valuable kinematic information for assessment purposes. However, although the current literature abounds of computational techniques designed for assessment purposes [212], these are based on data obtained from sensors attached to an arbitrary position of the instrument, thus providing an abstract measure of the kinematics of the tool [187, 213]. Precise information about the orientation of the shaft or the position of the tip is not usually employed, mainly due to the difficulty in obtaining this information from the sensor's data. Recently some computer vision approaches have been employed for estimating the instrument pose [156, 157].

In most cases, instrument tracking is achieved with 3D tracking devices that consist of: a movable component firmly attached to the instrument and a static component placed at a known position with respect to the simulation environment (e.g. box trainer) [116]. Such devices may be based on electromagnetic (EM) [76], optical [187, 214] or mechanical sensors [45, 215]. The state of the art 3D tracking systems currently used in laparoscopic simulation training provide highly accurate information regarding the pose of the movable component with respect to the known reference frame (static component) [116]. However, to obtain the pose of the instrument with respect to this reference frame, an extra calibration step is essential: knowledge of the instrument pose with respect to the attached tracking device.

While commercially available laparoscopic simulators implement custom calibration methods and achieve accurate results, to the best of our knowledge no plug-and-play calibration technique exists that could be easily applied in experimental practice. In their proposal of a guidance system for laparoscopic surgery, Nicolau *et al.* presented a calibration method for the position and orientation of the instrument tip with respect to a pattern marker attached to the handle. However, as described by the authors, the calibration method itself was not the main focus of that work [128]. Pagador *et al.* presented a calibration method for instrument tracking using EM tracking devices [216]. Although the authors provided a detailed description, the proposed calibration protocol required a custom made wooden apparatus, hence making their method difficult to replicate.

In our effort to implement sensor-based instrument tracking for our AR surgical simulation framework, we developed a robust, accurate and easy to implement calibration protocol for estimating the pose of a laparoscopic instrument with respect to a tracking device that is attached to a random location on the instrument handle. The outcome of our research was a calibration method that can be easily adapted to various types of rigid endoscopic tools and tracking devices.

This method can provide accurate tool kinematics for use in both box-trainer platforms as well as prototypes of custom VR and AR simulators.

9.3 Method description

9.3.1 Experimental Setup

The basic components of the experimental setup include: standard laparoscopic instrument and trocar, and the trakStar™ (Ascension Tech Corp., Burlington, VT) EM tracking system. The tracking system consists of a transmitter placed at a fixed position on a planar surface, and a receiver (sensor) mounted at an arbitrary position on the instrument's handle. Additionally, a tripod that holds the trocar at a fixed position with respect to the transmitter is employed. The experimental configuration is illustrated in Fig. 9.1a.

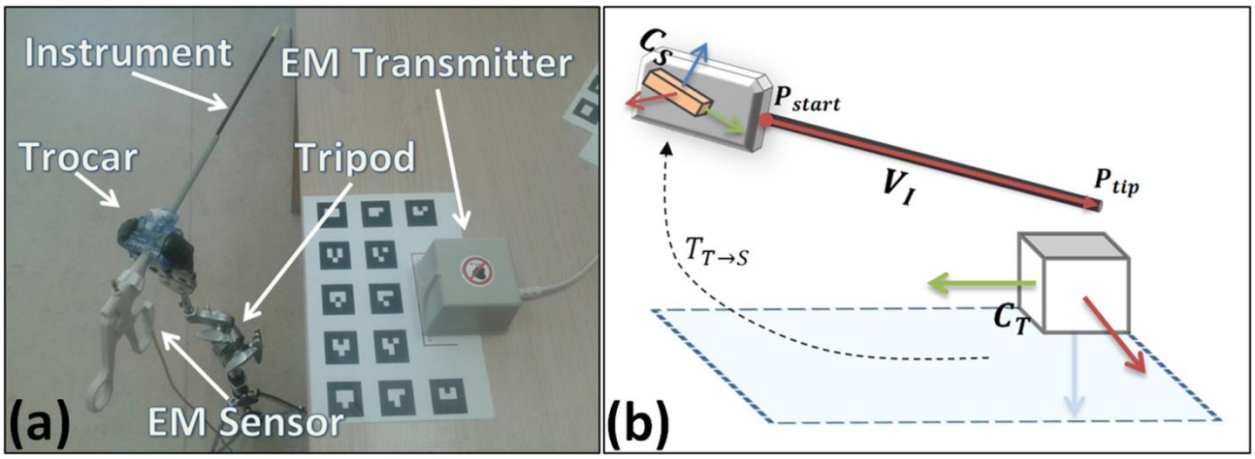


Figure 9.1 (a) Illustration of the experimental setup consisting of a laparoscopic instrument, an EM tracking device, a surgical trocar and a tripod to hold the trocar in a fixed position. (b) The coordinate systems corresponding to the experimental setup, $\boldsymbol{C_T}$ for the EM transmitter and $\boldsymbol{C_S}$ for the EM sensor, and the laparoscopic instrument expressed as a vector $\boldsymbol{V_I}$, from points $\boldsymbol{P_{start}}$ to $\boldsymbol{P_{tip}}$.

9.3.2 Theoretical Background

The aim of the proposed calibration method is to calculate the pose (position & orientation) of the instrument shaft, as well as the 3D position of the tooltip, with respect to the transmitter. Figure 1b illustrates the two coordinate systems corresponding to the experimental setup: the reference frame of the transmitter ($\boldsymbol{C_T}$), here referred to as the global coordinate system and the reference frame of the sensor ($\boldsymbol{C_S}$), here referred to as the local coordinate system. The EM tracking device provides the position and orientation of the sensor with respect to the transmitter, defined as a linear transformation $\boldsymbol{M_{T\to S}}$ from $\boldsymbol{C_T}$ to $\boldsymbol{C_S}$. A Cartesian transformation between two coordinate systems is expressed as a homogenous transformation matrix that consists of two parts: rotation and translation:

$$\boldsymbol{M} = \begin{bmatrix} & R_{3x3} & & x \\ & & & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9.1}$$

As Fig. 9.1b depicts, the instrument's shaft can be described as a vector $\boldsymbol{V_I}$ connecting the points $\boldsymbol{P_{start}}$ and $\boldsymbol{P_{tip}}$ of the local reference frame, where $\boldsymbol{P_{tip}}$ refers to the tooltip and $\boldsymbol{P_{start}}$ is an arbitrary point lying on the shaft, close to the instrument handles. Expressed in spherical coordinates any point along the shaft is defined as:

$$\boldsymbol{P} = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \begin{bmatrix} r \cdot \sin(\theta) \cdot \cos(\varphi) \\ r \cdot \sin(\theta) \cdot sin(\varphi) \\ r \cdot \cos(\theta) \end{bmatrix} \tag{9.2}$$

were (x_1, y_1, z_1) are the local coordinates of $\boldsymbol{P_{start}}$, θ and φ are the two angles defining the orientation of $\boldsymbol{V_I}$ with respect to $\boldsymbol{C_S}$, and r is the length of the vector connecting $\boldsymbol{P_{start}}$ and $\boldsymbol{P}$.

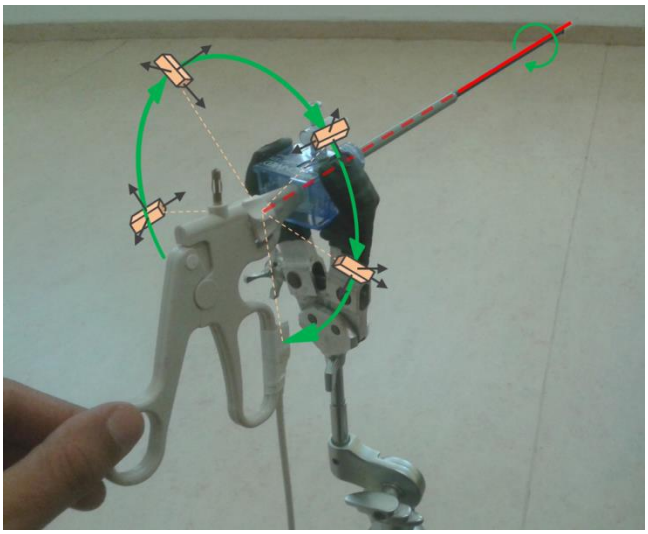


Figure 9.2 Step 1 of the calibration method: a 360° rotation of the instrument around its shaft provides a set of uniformly distributed poses of the EM sensor with respect to the EM transmitter.

Using the Cartesian transformation matrix of Eq. 1, any point with (x, y, z) coordinates belonging to the local reference frame, can be transformed to a point in the global reference frame using Eq. 3:

$$\boldsymbol{P} = \boldsymbol{M_{T \to S}} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{9.3}$$

Applying Eq. 3 for $\boldsymbol{P_{start}}$ and $\boldsymbol{P_{tip}}$ along with the transformation $\boldsymbol{M_{T \to S}}$, which is provided by the EM tracking system, the vector $\boldsymbol{V_I}$, corresponding to the pose of the shaft, as well as the tooltip

position $\boldsymbol{P_{tip}}$ can be fully defined in the global reference frame. So, essentially the calibration method aims to compute the following parameters: $\boldsymbol{P_{start}}$, r, θ and φ.

9.3.3 Calibration Protocol

The proposed calibration method consists of two steps. In the first step, the instrument is fully inserted into the trocar, which is positioned at a fixed tripod within the range of the EM transmitter (Figure 9.1a). This setup prevents the instrument from moving to a direction other than the direction of the trocar. The trocar direction and consequently the instrument shaft, defines an arbitrary axis in the global reference frame ($\boldsymbol{C_T}$). At this stage a 360° rotation of the instrument around its shaft is performed, as illustrated in Fig. 9.2. During this rotation, the EM sensor performs a circular motion with respect to $\boldsymbol{C_T}$ providing a set of uniformly distributed poses $\boldsymbol{M^i_{T\rightarrow S}}$. The barycenter of rotation for this circular path, which lies on the shaft (Fig. 9.2), is calculated using Eq. 4.

$$[x_c\ ,\ y_c\ ,\ z_c\ ] = {}^{1}/_{N} \cdot \sum_{i=1}^{N} [\ x,\ y,\ z\ ]_i \tag{9.4}$$

where $(x_c\ ,\ y_c\ ,\ z_c)$ are the coordinates of the barycenter in the global reference frame, *N* is the total number of sensor's positions, and $(x,\ y,\ z)_i$ are the coordinates of the $\boldsymbol{M^i_{T\rightarrow S}}$ origins in the global reference frame.

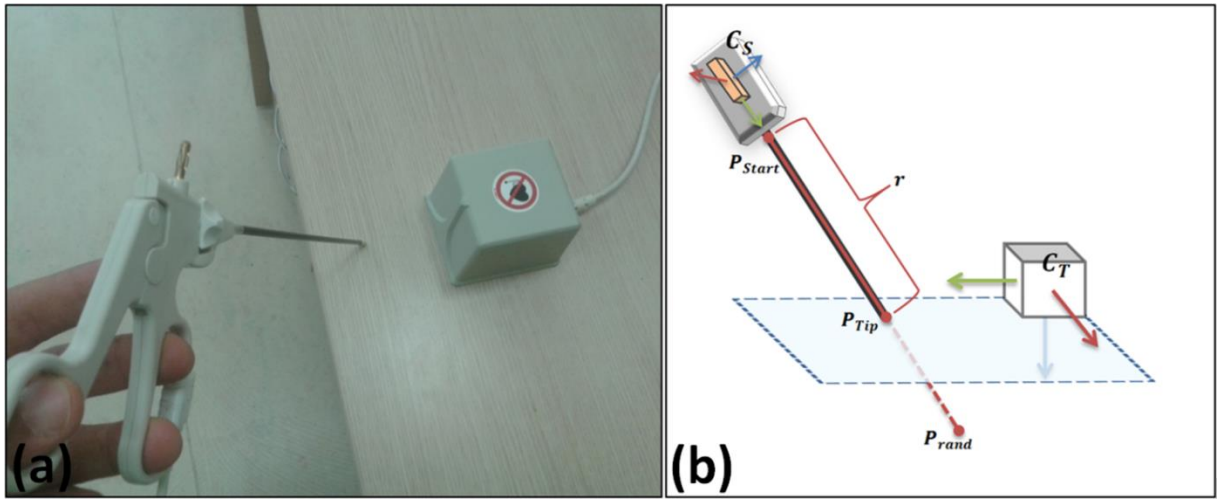


Figure 9.3 (a) Step 2 of the calibration method: The instrument tip is placed in contact with the EM transmitter's bottom surface. (b) The length (r) of the instrument shaft is calculated via the point $\boldsymbol{P_{tip}}$, at which the line $\boldsymbol{P_{start}}$-$\boldsymbol{P_{rand}}$ intersects the bottom plane of the EM transmitter coordinate system.

Since $(x_c\ ,\ y_c\ ,\ z_c)$ lie on the instrument shaft, transforming these coordinates into the local reference frame provides $\boldsymbol{P_{start}}$. This transformation is achieved using the inverse of any $\boldsymbol{M^i_{T\rightarrow S}}$:

$$\boldsymbol{P_{start}} = [\boldsymbol{T^{i}_{T\rightarrow S}}]^{-1} \cdot [x_c\ ,\ y_c\ ,\ z_c\ ]^{T} \tag{9.5}$$

Given the fact that the sensor moves along a circular path, the collected poses of the sensor lie on a plane that is perpendicular to the axis of rotation. Hence, singular value decomposition (SVD) of the collected $\boldsymbol{M^{i}_{T\rightarrow S}}$ origins provides the orientation of the instrument shaft with respect to the global reference frame in a form of a 3D vector. The remaining step is to transform this 3D vector into the local coordinate system. This is achieved by using the inverse of any $\boldsymbol{M^{i}_{T\rightarrow S}}$ (Eq. 3). This transformation results in a direction vector $[n_x\ ,\ n_y\ , n_z]$ at the local coordinate system. The angles θ and φ, which describe the orientation of $\boldsymbol{V_I}$ with respect to the EM sensor, are calculated as:

$$\theta = \arccos\left(\ n_z\ /\ \sqrt{{n_x}^2 + \ {n_y}^2 + \ {n_z}^2}\right) \tag{9.6}$$

$$\varphi = arctan\left(n_y/n_x\right) \tag{9.7}$$

The second step of the calibration protocol aims to find the length of $\boldsymbol{V_I}$, denoted as r in Eq. 2. During this step, the instrument is positioned so that that the tooltip comes in contact with the surface on which the EM transmitter is placed (see Fig. 9.3a). At this stage, $\boldsymbol{P_{start}}$ and the angles θ and φ are known. Hence, considering an arbitrary length for the instrument shaft, a random point $\boldsymbol{P_{rand}}$ that lies on the shaft is assumed (Fig. 9.3b). Solving a line-plane intersection system of equations, the point $\boldsymbol{P_{tip}}$ at which the line $\boldsymbol{P_{start}}$-$\boldsymbol{P_{rand}}$ intersects the bottom plane of $\boldsymbol{C_T}$ is derived. The distance between this point and $\boldsymbol{P_{start}}$ is the length of the instrument's shaft (r).

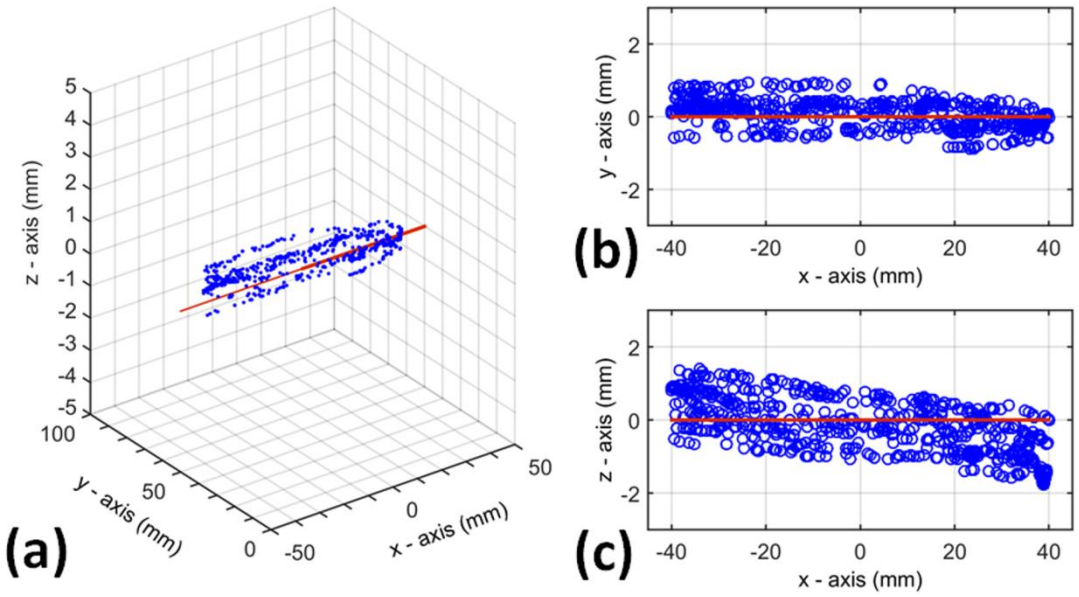


Figure 9.4 (a) 3D positions collected while the instrument tip moves across a line (indicated in red) parallel to the X axis of the transmitter's coordinate system. (b) Deviation from the ideal line in the Y direction. (c) Deviation from the ideal line in the Z direction.

9.4 Method evaluation

To evaluate the accuracy of the presented method, three evaluation experiments were performed with regard to the instrument's orientation and tip position. The first experiment aimed to measure the accuracy in the estimation of angles θ and φ that describe the instrument's shaft orientation. As a gold standard we used a second sensor connected to the same transmitter that the sensor attached to the instrument handle is connected to. In particular, a custom component was built allowing the second sensor to be positioned inside a trocar so that its axis was perfectly aligned with the direction of the trocar. Based on this configuration, we were able to obtain theoretical estimates about the angles θ and φ, which essentially provide the direction of the trocar (with respect to the EM transmitter). The trocar was always placed at a fixed position with respect to the transmitter. Then, a set of 50 calibration estimates about the trocar direction (angles θ and φ), were obtained by rotating the instrument around the trocar direction axis (Fig. 9.2). These estimates were provided by the proposed method based on the measurements obtained by the first sensor attached to the instrument handle as described in the Methods. Comparing the outcome of each of these calibrations with the theoretical values of θ and φ, the mean error and standard deviation were measured. As Table 1 illustrates, the mean errors were 0.46° ± 0.2° for angle θ and 0.6° ± 0.51° for angle φ.

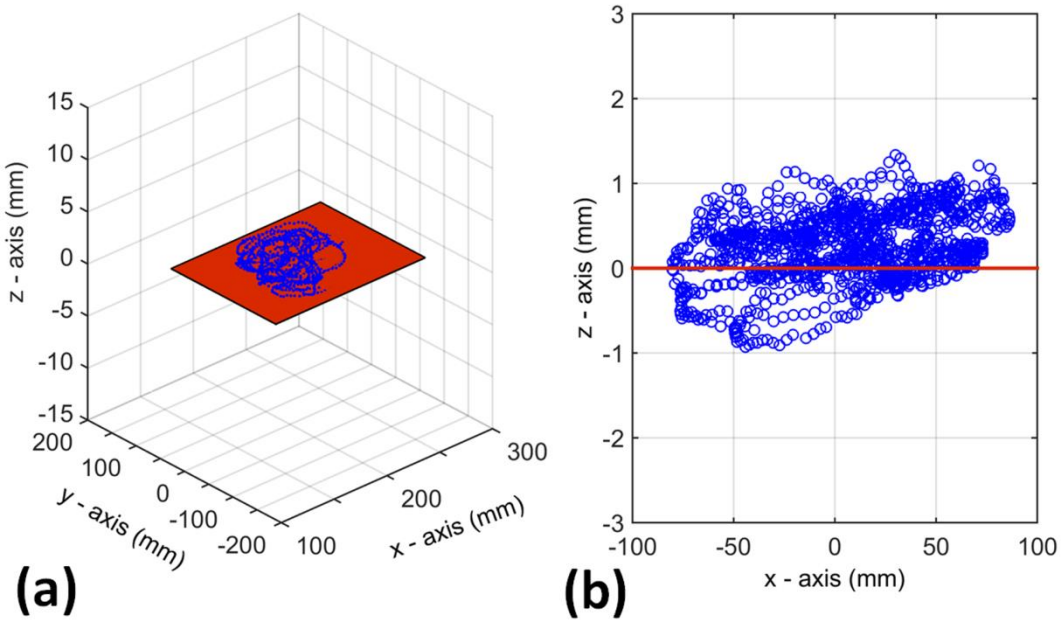


Figure 9.5 (a) 3D positions collected while the instrument tip performs continuous random movement on the x-y plane of the EM transmitter's coordinate system. (b) Deviation from the ideal plane in the Z direction.

Then, we evaluated the accuracy of the proposed method in estimating the 3D position of the tooltip. During the second experiment, a set of tooltip positions were recorded while the tooltip

moved across a line parallel to the x-axis of the transmitter's coordinate frame, as illustrated in Fig. 9.4a. The projection of these positions on the x-y and x-z planes of the transmitter coordinate frame can be seen in Fig. 9.4b and Fig. 9.4c respectively. Table 9.1 depicts the mean error regarding the deviation of the recorded positions from the theoretical line: 0.67 ± 0.4 mm in the y-axis and 0.37 ± 0.2 in the z-axis.

During the third experiment, a set of tooltip positions were collected while the tooltip performed random movements on the x-y plane of the transmitter's coordinate frame, as illustrated in Fig. 9.5a. The mean error regarding the deviation of the recorded positions from the theoretical plane, indicated as a red line in Fig. 9.5b, is: 0.39 ± 0.2 mm (Table 9.1).

Table 9.1 Mean errors and standard deviations for the three experiments that were performed to evaluate the accuracy of the proposed method.

		Error	Standard deviation
Experiment 1	Error in angle φ	0.68 °	0.51 °
	Error in angle θ	0.46 °	0.23 °
Experiment 2	Deviation from line on the y-axis	0.67 mm	0.46 mm
	Deviation from line on the z-axis	0.37 mm	0.27 mm
Experiment 3	Deviation from plane	0.39 mm	0.28 mm

Figure 9.6 illustrates qualitative results regarding the potential use of the proposed calibration method in an AR environment. A pattern-marker was employed to obtain the relationship between the camera and the EM transmitter. This setup provided the pose of the EM sensor, attached on the instrument handles, with respect to the camera coordinate system. Using this information along with the outcome of our calibration, a virtual cylinder (in red) was augmented at the camera scene, in order to visually illustrate the accuracy regarding the estimation of the position of the shaft with respect to the camera. Although tracking of the pattern-marker introduced additional errors to the final result, the visual outcome is indicative of the accuracy that the proposed calibration method provides.

9.5 Discussion

This Chapter presented a calibration method for calculating the pose of a rigid laparoscopic instrument with respect to a 3D tracking sensor that can be attached to an arbitrary position on the handle. Our method allows real-time tracking of the tip position, as well as tracking of the shaft orientation with respect to the training, or operating, environment. The goal of this study is to

provide an accurate and easy to implement calibration protocol, which can be applied to different types of instruments and a wide variety of tracking devices. Existing custom training systems may also be in benefit from this technique, such as for example for extracting the tooltip position or for generating a kinematic model of the instrument shaft.

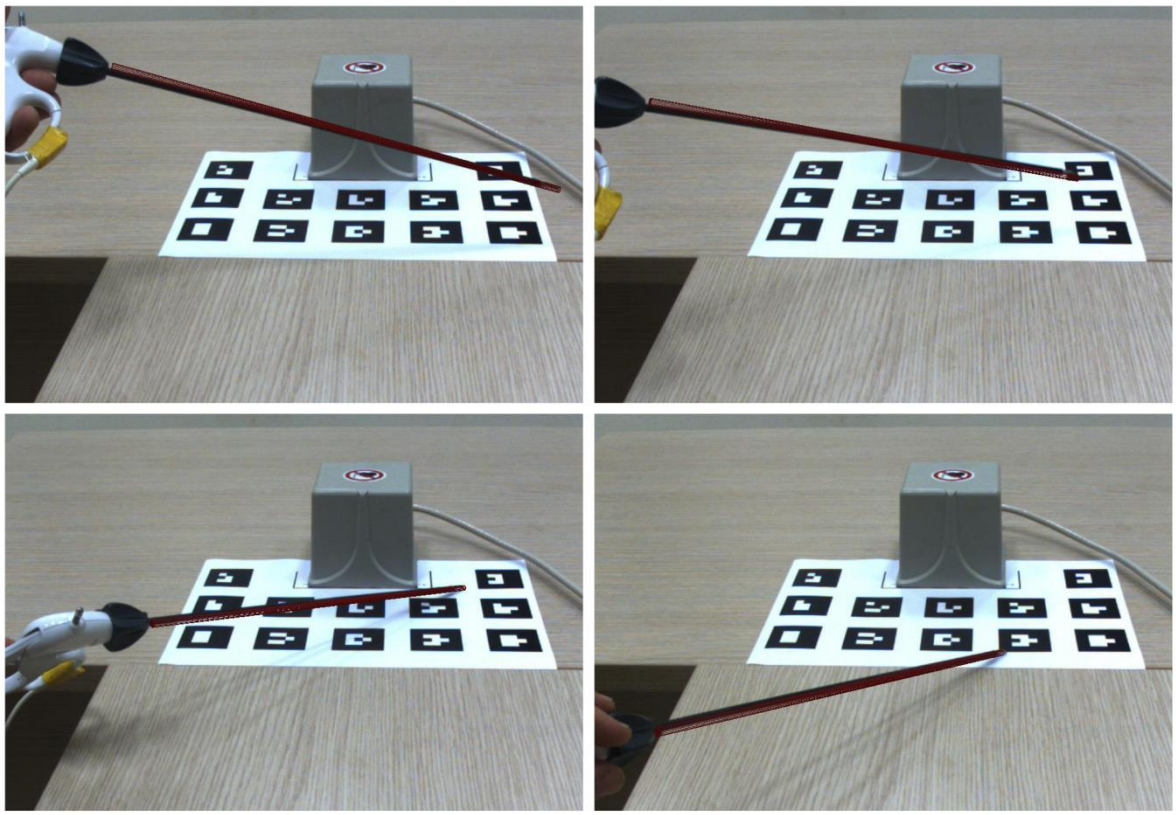


Figure 9.6 Screenshots of an AR application, where a virtual representation of the instrument shaft (in red) is rendered on top of the real shaft. Position and orientation of the virtual shaft are calculated using the output of the presented calibration method.

Evaluation results indicate sub-millimeter accuracy in the estimation of the tooltip position with respect to a known coordinate frame. This level of accuracy allows the potential use of the proposed method for objective assessment of the trainee's performance in standard box trainers, where information about the tip position cannot be extracted, although it is important [217]. Studies have showed that knowing how the operator performs a surgical maneuver is essential since a higher level of dexterity is associated with shorter operations and fewer complications [218].

Our results also indicate sub-degree accuracy in the estimation of the orientation of the instrument shaft with respect to the sensor attached to the handle. This finding allows the proposed method to be employed for example in AR applications, where standard problems such as occlusion handling require increased levels of precision [157]. To support this claim, the

presented method was recently employed by our group to obtain an accurate 3D model of the laparoscopic instrument for AR applications in simulation-based training [76].

A significant advantage of the calibration method described in this Chapter is that it is based on two simple calibration steps that are performed only once, prior to task performance. Yet, the experimental setup is based on a common EM sensor, a surgical trocar, and a static holder, all components of which can be easily available in a simulation training lab. These prerequisites allow the method to be easily applied by users with non-technical background to derive a 3D representation of the instrument shaft. An additional advantage arises from the fact that the method allows placement of the sensor at any arbitrary position on the instrument handle, offering surgeons the flexibility to decide upon an optimum placement of the sensor that will not affect or restrict hand movements. Hence, our method can also be utilized for *in vivo* applications, where freedom of motion is crucial. In such a case, one could design experiments to obtain the kinematics of the entire 3D instrument model, and also combine this information with pre-calculated (e.g. from CT or MRI) anatomical position data of critical anatomies.

A potential limitation originates from the inherent inaccuracy of EM sensors, which may be affected by various external factors. For example, EM sensors similar to the ones used in this study can demonstrate significant loss of accuracy due to EM interference caused by metallic objects present in the surrounding environment and in close proximity with the sensors employed. In a similar manner, optical-based tracking devices require a clear field of view between the camera and the sensor in order to provide accurate tracking results. These factors should also be considered and avoided in the calibration setup; otherwise they could significantly affect the accuracy of the results.

In conclusion, we have developed a calibration technique to obtain the tip position and a 3D model of the shaft of rigid endoscopic instruments utilized in MIS. In contrast to other works were EM tracking sensors are placed on the tip of surgical tools [219], our method utilizes a single EM sensor placed on the handle. Moreover, the proposed method does not require the employment of special calibration frames [216], it is simple, inexpensive, and have potential applications not only in training systems but also in the operating room.

Chapter 10

Evaluation of the Final Framework Setup

This Chapter presents the final version of our surgical simulation framework, including all its core components as described earlier; Ogre3D and ARToolkit for real-time visualizations of virtual objects within a box-trainer environment, Bullet for physics-based interactions between laparoscopic instruments and virtual objects, and electromagnetic (EM) sensors for instrument tracking. In addition, an Arduino controller equipped with rotary encoders and infrared sensors has been utilized for obtaining the additional degrees of freedom regarding the laparoscopic instruments (opening-closing angle of the tooltip, and rotation of the instrument's shaft). The presented setup includes all the essential components for being characterized a fully functional surgical simulator, introducing AR technology in MIS training.

10.1 Introduction

Although the studies presented in Chapters 7 and 8 have opened the opportunity for introducing core elements of AR technology into surgical education, demonstrating promising results regarding the potential of developing AR tasks based solidly on computer vision algorithms [156, 157]. However, the presented implementations did not allow the trainee to actually interact with virtual training models rendered on the screen, since key degrees of freedom such as grasping and tip rotation could not be captured with the instrument tracking techniques utilized in our framework, thus limiting the training value of the implemented AR tasks.

As stated in the introduction of this thesis, our goal was to bridge the gap between box-trainers and VR simulators, and demonstrate that an AR-based training system could utilize the important assets of both training modalities: the increased sense of visual realism and force-feedback provided by the endoscopic tools, combined with the flexibility of VR in the development of training scenarios and the opportunity for automated performance assessment based on real-time data collection. If such a system was to be developed, the VR elements introduced to the AR scene should had the ability to respond to collisions and other type of forces applied by the actual endoscopic tools, similarly to the training models used in box-trainers (e.g. pegs, cutting tissue, etc.). Reproducing these tasks in an AR environment though is not trivial due to a series of interrelated technical challenges such as: tracking and pose estimation of the tools, tracking and geometric modeling of the physical world, 3D rendering of the virtual objects, and physics-based simulation of the interactions occurred between the VR objects and the physical world (e.g. collisions with the box surface and endoscopic tools). Hence, implementing tasks that involve interaction between the actual endoscopic tools and the (training) VR models would clearly signify an important step towards the development of a genuine AR surgical simulator. In addition, important assets that apply in VR simulation, such as automated performance assessment and flexibility in modifying the difficulty level of the tasks, would also be applicable in AR-based training.

10.2 Experimental setup

In the final research stage of this thesis, we implemented a multisensory interface that could be easily attached to the handle of the endoscopic tool providing sufficient information about the tool kinematics. This setup allowed the implementation of training scenarios for technical skills acquisition such as perception of depth of field, hand-eye coordination and bimanual operation. Based on this system, the trainee is able to interact with various virtual elements introduced into the box-trainer, using the actual laparoscopic instrumentation (camera and tools).

10.2.1 Hardware setup

The main components of the system include: a standard PC with a monitor (Intel^{R} Core^{TM} 2 Duo 3.1 GHz), a fire-wire camera with appropriate wide-angle lenses (PtGray Flea®2), a box-trainer, a pair of laparoscopic tools (Fig. 10.1), and three different types of sensors attached to the laparoscopic tool. The sensors (Fig. 10.2) are used to provide eight degrees of freedom (DOF) information regarding the tool kinematics: 3D pose of the tool (6 DOF), shaft rotation (1 DOF), and opening angle of the tooltip (1 DOF). Moreover, a fiducial pattern marker is placed on the bottom surface of the box trainer to help us define the global coordinate system of the simulation environment.

To obtain the pose of the tool, trakStar^{TM} (Ascension Tech Corp., Burlington, VT) electromagnetic (EM) position-orientation sensors are employed. The EM transmitter is placed at a fixed position within the box-trainer, (see Fig. 10.1) whereas the sensors are attached to the tool handles as shown in Fig. 10.2. The pose of the receivers with respect to the tool as well as the pose of the transmitter with respect to the global reference frame are obtained through a calibration process.

A custom-made rotary encoder controlled via an Arduino[9] microcontroller board is employed to measure the rotation of the shaft. In particular, the encoder consists of a magnetic rotor firmly attached to the shaft, and a plastic stator attached to the handle as shown in Fig. 10.2. Inside the stator, two Hall Effect sensors (HFs), positioned with an approximate 90 degree angle separation, detect changes on the sinusoidal waveform that the magnetic rotor generates during rotation. The voltage output of the HFs is digitally converted using a microcontroller's analog to digital converter (ADC). Calculation of the shaft rotation angle is obtained with the CORDIC algorithm [220]. This setup allows us to obtain angular measurements for the full 360^{o} range of the shaft revolution. The microcontroller is equipped with a 10bit resolution ADC, corresponding to an angle resolution of 0.35^{o}.

[9] http://www.arduino.cc/

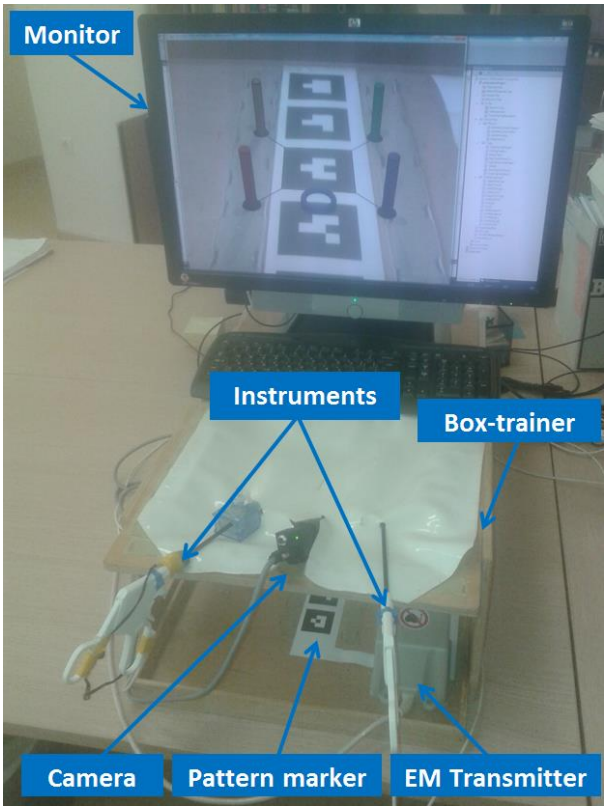


Figure 10.1 Experimental setup, consisting of a box-trainer, a pc, two laparoscopic instruments and a fire-wire camera.

The opening angle of the tooltip is acquired via a specially designed IR proximity sensor attached to the trigger of the tool as shown in Fig. 10.2. The sensor consists of an IR Led emitter–receiver pair. The receiver measures the amount of the emitted IR light reflected to the handle of the instrument. The voltage output of the IR receiver is inversely proportional to the distance between the trigger and the handle. In order to translate distance information into an angle value, a pre-calibration process is required. This process provides the maximum and minimum amount of IR reflectance corresponding to the maximum and minimum opening angle respectively. Additionally, the maximum opening angle of the tooltip has to be known. For the laparoscopic tools employed, the tooltip angle range varies from 0^o to 45^o. The output of the IR proximity sensor is fed to the microcontroller ADC, and the transformation of the measured reflectance into angle units is provided by:

$$\varphi_{curr} = R_{curr} \times \frac{R_{max} - R_{min}}{\varphi_{max}} \tag{10.1}$$

where φ_{curr} is the calculated rotation angle, R_{curr} is the reflectance measurement, φ_{max} is the maximum opening angle of the tooltip, and R_{max}, R_{min} are the maximum and minimum reflectance values respectively obtained from the pre-calibration process.

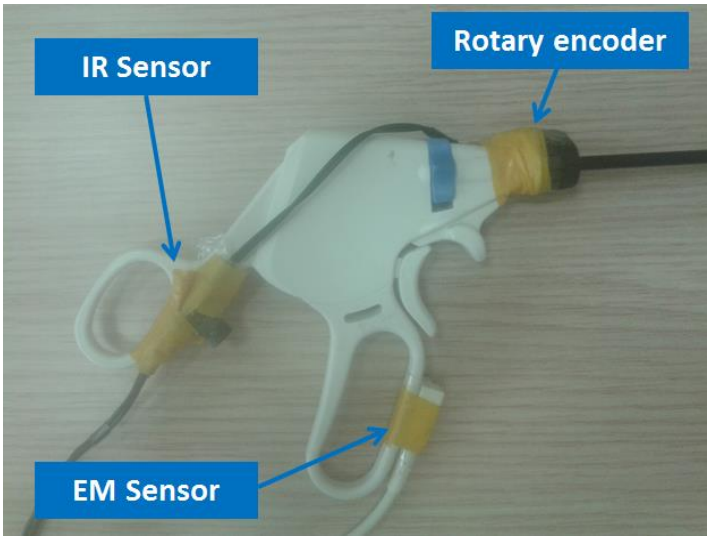


Figure 10.2 A close-up of the laparoscopic instruments sensors. The illustrated setup provides a total of eight degrees of freedom that fully describe the instrument's kinematics.

10.2.2 Simulation engine

As discussed in Chapter 5, the software engine for the VR-based laparoscopic tasks was developed using Ogre3D[10] as a framework for managing both the creation of the graphical user interface and the rendering of the mixed reality scene for the tasks. The ARToolkitPro[11] library was used to calculate the spatial relationship between the camera and the box-trainer reference frame in terms of rotation and translation, by tracking the pose of the pattern marker described earlier. Simulation of the physical behavior of the virtual objects (such as collision detection, response between tools and virtual objects, and soft body deformations), was implemented with the Bullet[12] real-time physics engine. In order to meet the specific needs of each task and achieve realistic behavior of the virtual objects, several modifications and supplementary algorithms to the Bullet source code were applied. Finally, the 3D models of the virtual objects employed in the simulation tasks, were performed in Blender3D[13].

10.3 Training scenarios

10.3.1 Task description

Based on the aforementioned simulation engine, three training tasks were developed, targeting different technical skills in laparoscopic surgery (Fig. 10.3):

[10] http://www.ogre3d.org
[11] http://www.artoolworks.com
[12] http://bulletphysics.org
[13] http://www.blender.org

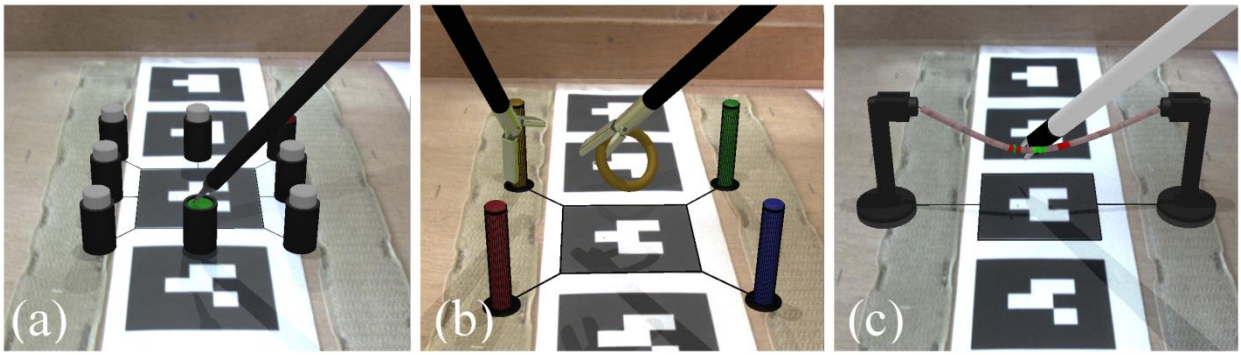


Figure 10.3 Screenshots of the three augmented reality training tasks: (A) instrument navigation, (B) peg transfer, (C) clipping.

Task 1: Instrument Navigation (IN). A total of eight rounded white buttons, each one enclosed in a black cylinder, is introduced to the center of the box-trainer (Fig. 10.3a.). The task requires the user to hit the buttons in a sequential order, when each one of them is highlighted in green. The order on which the buttons get highlighted is random and also varies randomly among training sessions. The user is given a time limit of eight seconds for each button. Two types of error were recorded during this task: the number of missed targets due to time expiration, and the number of tooltip collisions with either the base of the box-trainer or the black cylinders.

Task 2: Peg Transfer (PT). This task is based on a virtual peg-board consisted of four cylindrical targets and an equal number of torus-shaped pegs introduced sequentially into the scene (Fig. 10.3B.). Each target has a distinctive color. The trainee has to lift and transfer each peg to the target with the same color. When a peg has been transferred, another peg of different color is introduced. Two types of error were recorded during this task: unsuccessful transfer attempts and peg drops. Unsuccessful attempts occur when a peg is dropped away from the center of the box-trainer or when it is transferred to a target with a different color, while peg drops occur each time a peg is dropped accidentally during transfering.

Task 3: Clipping (CL): A virtual vein is introduced into the scene as shown in Fig. 10.3C. The center of the vein is highlighted in green and two different locations on either side of its center are highlighted in red. The width of each highlighted region is 5 mm. The goal is to apply a clip at the center of each of the red regions using a virtual clip applicator. Once clipped, then the user has to use virtual scissors to cut at the center of the green region. Two different metrics were recorded during this task: The distance error during clipping/cutting with respect to the center of the corresponding region, and the number of unsuccessful clipping/cutting attempts, which are recorded when the user clips/cuts outside the highlighted area.

10.4 Study design and statistical analysis

We collected data from subjects with two different levels of expertise: ten experienced surgeons (experts), and ten individuals with no experience in laparoscopic surgery (novices). Each participant performed two trials of each task. Prior to performance, each subject performed a trial session of each task to enhance familiarization.

The collected data regarding the performance of the two groups were statistically analyzed using the MATLAB® Statistics toolbox (MathWorks, Natick, MA, USA). Between-group comparison of performance metrics was undertaken with the Mann-Whitney U test (5 % level of significance).

10.4.1 Questionnaire

To further investigate the training value of the simulator, the subjects from both experience groups were asked to complete a questionnaire after the completion of the study protocol. Using a 5-point Likert scale scoring system, the provided choices were “none”, “low”, “medium”, ”high” and “very high”. Each of the three training tasks was assessed based on the following criteria:

1. How do you rate the realism of the graphical representation of the VR objects?

2. How do you rate the realism of the interaction between the instruments and the VR objects?

3. How do you rate the difficulty of the task?

4. How important was the lack of force feedback during tool-object interaction?

5. How restrictive in performing a task was the attachment of sensors on the laparoscopic tool?

10.5 Results

To evaluate the construct validity of the proposed system a statistical comparison of the two experience groups was performed with regard to the three tasks described previously. The performance metrics included the two types of errors for each task, the task completion time, and the total pathlength of the laparoscopic tools. Figures 10.4 – 10.6 illustrate bar charts of the four performance metrics for the three training tasks respectively. It is clear that experts outperform novices in all tasks and metrics. Table 10.1 depicts the median values of each group and the p-values obtained from the between-group comparison test. It can be noticed that in almost all performance metrics, the p-values denote a highly significant performance difference between the groups (< 0.01).

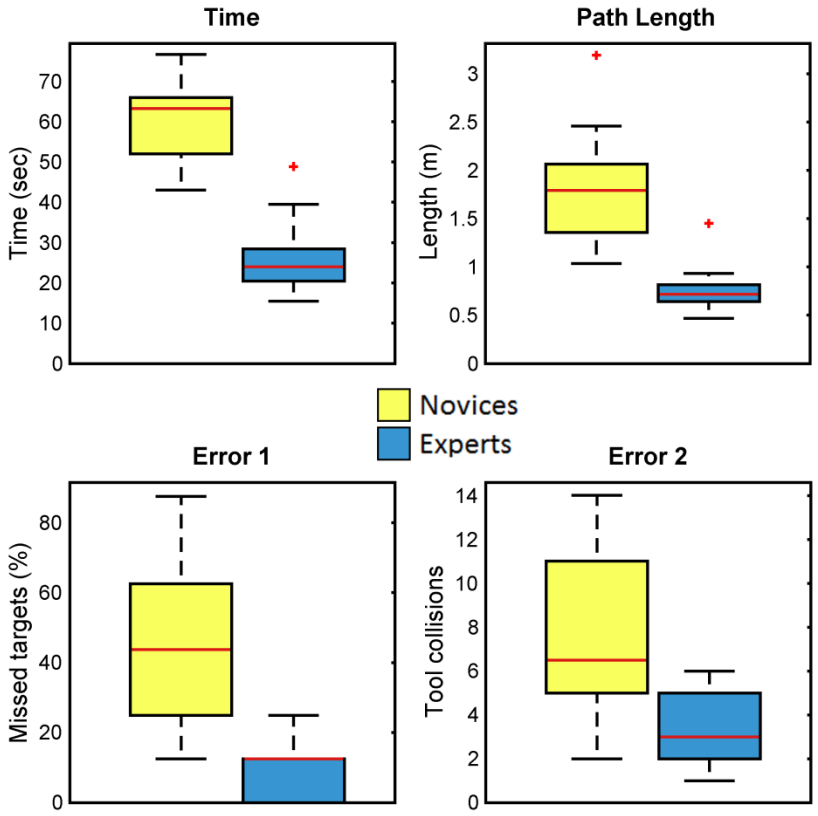


Figure 10.4 Instrument navigation: Box plot comparison for time, pathlength, error 1 and error 2 between the two experience groups.

The only metric that demonstrates a slightly reduced difference between the groups is error 1 for peg transfer, although the measured p-value is also significant (<0.05). From the same figures, it can also be noticed that the interquartile difference of experts is clearly smaller than that of the novices, indicating a robust performance. This is especially noticeable for time and pathlength, where the results for the experts group demonstrate a very small interquartile difference across all tasks. Additionally, for the PT task one can notice that error 1 for experts is 0%, which indicates that none of them missed a peg across the attempted trials. Similarly, for the clipping task the error 2 of the experts group is 0%, indicating that none of them missed a target during the trials. It is also important to notice that for the novices group the completion time and instrument pathlength are two to three times higher than that of the experts group for all training tasks, indicating the potential of these metrics to capture the difference in experience between the two groups. The actual numerical results for this comparison are provided in Table 10.1.

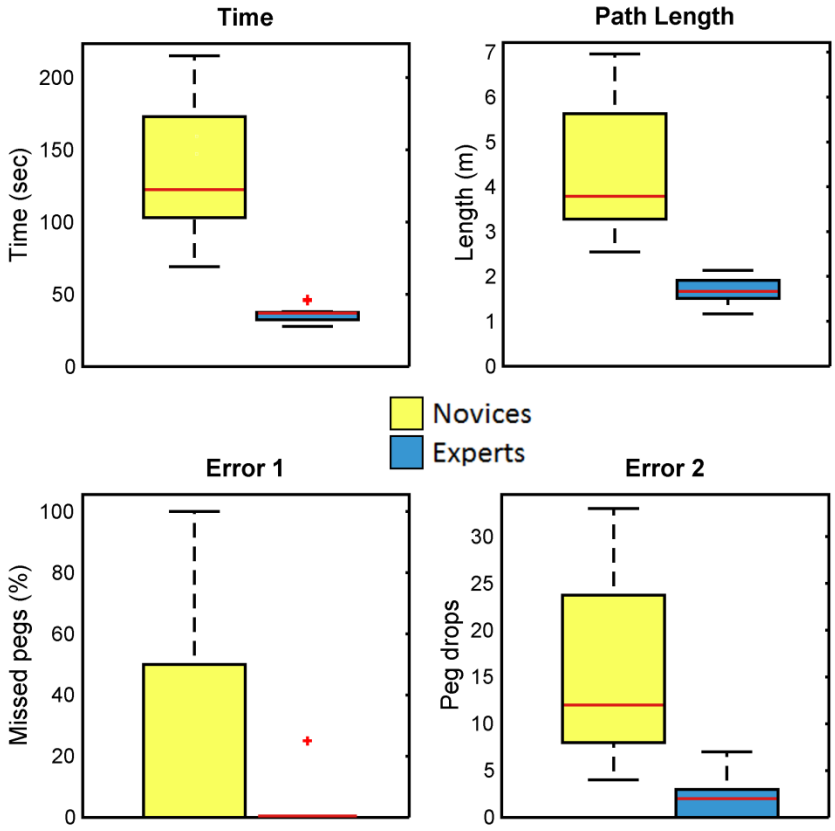


Figure 10.5 Peg transfer: Box plot comparison for time, pathlength, error 1 and error 2 between the two experience groups.

Figure 10.7 illustrates a plot of the trajectory of the instrument controlled by the dominant hand of the subject that is closest to the median of the total pathlength. It is clear that the expert's trajectory is smoother and more confined compared to that of the novice. Especially for the clipping task, the expert demonstrates a fine pattern of movements, whereas the novice performs multiple retractions of the tool to locate the targets. Moreover, the tool trajectory of the novice is less targeted compared to that of the expert subject, and it is also accompanied by a significant amount of jitter.

Table 10.1. Performance results (medians) and between-group comparison of the three training tasks and the four observed metrics.				
Task	Metric	Experts	Novices	p-value
Instrument navigation	Path (m)	0.71	1.79	<0.01
	Time (s)	24.04	63.26	<0.01
	Error1	12.50	43.75	<0.01
	Error2	3	6.50	<0.01
Peg transfer	Path (m)	1.66	3.78	<0.01
	Time (s)	36.88	122.43	<0.01
	Error1	0	0	<0.05
	Error2	2	12	<0.01
Clipping	Path (m)	1.81	5.33	<0.01
	Time (s)	26.24	101.15	<0.01
	Error1	1.07	2.55	<0.01
	Error2 (%)	0	33.3	<0.01

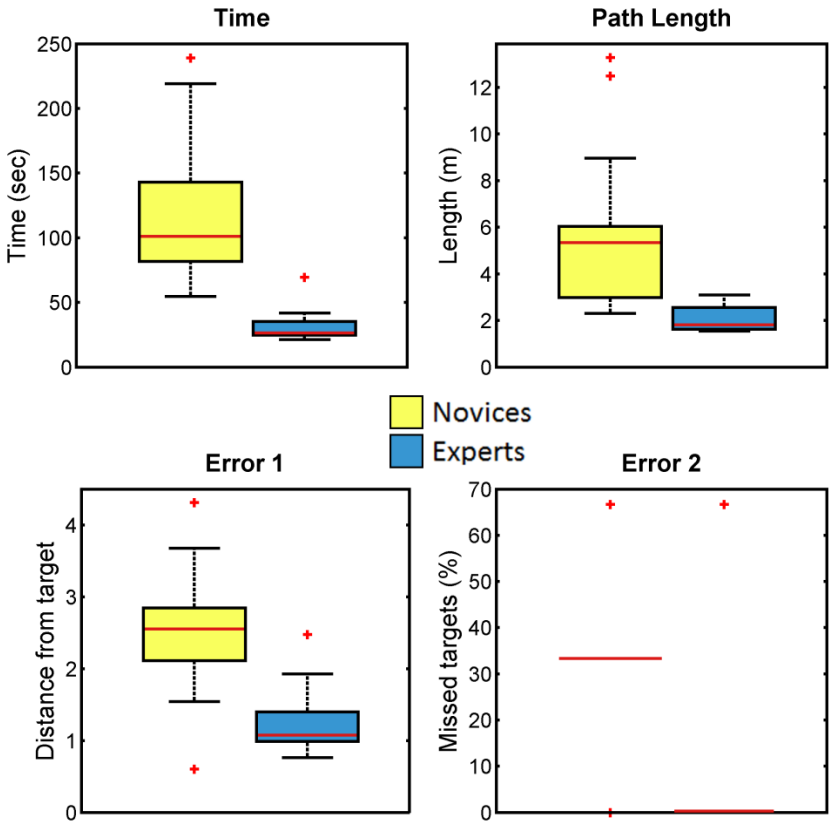


Figure 10.6 Clipping: Box plot comparison for time, pathlength, error 1 and error 2 between the two experience groups.

Table 10.2 illustrates the subjects' ratings for the face validity of the proposed training system. With regard to the graphical representation and the physics-based behavior of the augmented reality tasks, the subjects from both groups agreed that the attributed realism was more than sufficient (high–very high) to provide the expected training qualities. The subjects also found the difficulty of the instrument navigation task to be lower compared to peg transfer and clipping. For these two tasks, the novices group encountered greater difficulty in performing them compared to experts. Regarding the sense of force feedback, both groups seem to agree that its absence does not play a significant role for the tasks that do not involve soft-tissue deformations (instrument navigation and peg transfer). For the clipping task however, force feedback seems to be important, based on the subjects' ratings. The subjects also concluded that the sensors attached to the laparoscopic instruments do not seem to restrict the maneuvers performed by the user during task performance.

Table 10.2. The feedback questionnaire statements and mean ratings for each task and experience group.

Statements	Instrument Navigation		Peg Transfer		Clipping	
	Novices	*Experts*	*Novices*	*Experts*	*Novices*	*Experts*
1. How do you rate the realism of the graphical representation of the VR objects?	4.5	4	4.5	4.5	4	4
2. How do you rate the realism of the interaction between the instruments and the VR objects?	4.4	4.2	4.2	4	4	3.8
3. How do you rate the difficulty of the task?	2.5	2.5	4.0	3	4.5	3.5
4. How important was the lack of force feedback during tool-object interaction?	3	3.5	2.5	3	4	4
5. How restrictive in performing a task was the attachment of sensors on the laparoscopic tool?	1	1.5	1	1.5	1	1.5

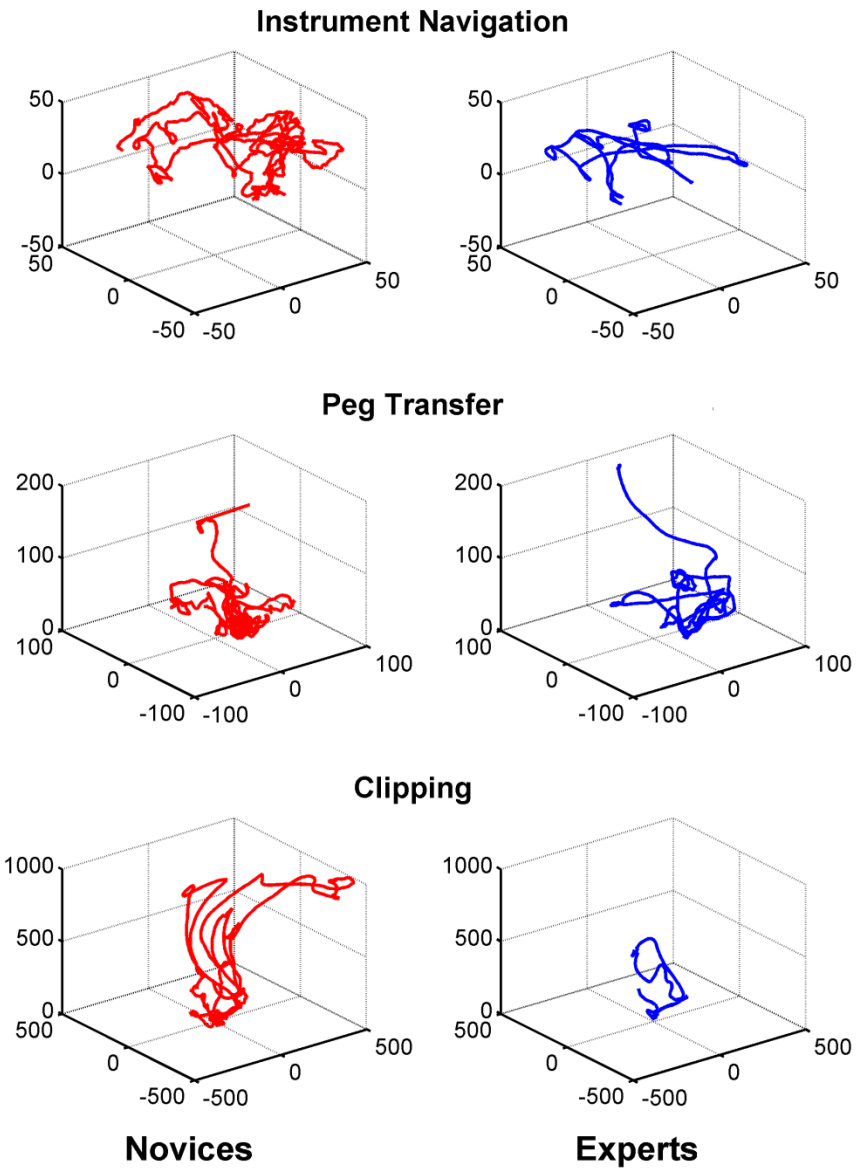


Figure 10.7 Instrument trajectories obtained from the subjects that were closer to the pathlength median in the three training tasks.

10.6 Discussion

In this Chapter we presented the evaluation study of the final version of our surgical simulation framework. In contrary to the existing AR-based training platforms (ProMIS [129]), the system presented in this Chapter is a genuine AR-based training system that allows the user to interact in real-time with rigid and deformable VR models. An important advantage of the proposed system is the increased sense of visual realism, emerging from the realistic mixture of real world and VR models that AR technology provides. The improved visual feedback provides better understanding

about the position of the VR elements, which in turn provides the trainee enhanced perception of depth compared to purely virtual environments. In addition to the visual realism however, the proposed system allows trainees to gain better familiarization with the actual operating conditions due to the actual laparoscopic instrumentation employed to perform the tasks. Another advantage of the proposed simulator is that its cost is significantly less than that of a commercial VR laparoscopic simulator, since it employs custom-made low-cost sensors and open-source graphics libraries. Hence the proposed system seems a low-cost alternative to the commercial VR simulators for recognizing users with different experience in laparoscopic surgery.

A potential drawback of the proposed simulator is the lack of force feedback during interaction. Although there are a few options for commercially available force feedback devices, it would be problematic to integrate them into the proposed AR setup since they are bulky and thus they would obscure the visual field of view if they were placed inside the box trainer. To overcome this limitation, a device that would provide force feedback without affecting the camera field of view could be employed, but to the best of our knowledge a device with such characteristics is not currently available. Nevertheless, the importance of force feedback in the acquisition of basic surgical skills is subject to controversy [156]. Studies indicate that for basic skills, force feedback does not play a critical role in the efficiency of a training platform [62, 221]. In addition, some studies claim that an inaccurate implementation of force feedback results in poor quality of simulation that could lead to adverse effects and bad habits [222]. However, there are studies that reporting that even a simulator without force feedback can provide effective transfer of training [55]. To assess its significance in the proposed setup, we asked the participants to rate the impact of the lack of force feedback for each task practiced. Based on the replies received, it was concluded that although feedback would be and important asset, it is not crucial for training tasks that do not involve interaction with soft tissues. This may be due to the fact that interaction with soft tissue requires gentle hand movements, and hence force-feedback would provide better realization of the applied forces, helping the surgeon not to damage the tissue.

Besides the importance of force feedback, the questionnaire statements aimed to rate other key aspects of our simulator such as its realism, difficulty of the training tasks and the potential motion-restrictive effects that the attachment of sensors to the tool may produce. First, we asked the participants to evaluate the visual representation and physics-based behavior of the virtual models. The two experience groups seemed to agree that the proposed simulator provides in overall more than a sufficient sense of realism. Additionally, novices faced greater difficulties in achieving the goals of the proposed tasks compared to the experts group. Both experience groups also agreed that the attachment of the sensory equipment to the laparoscopic tools did not reduce their freedom to perform the required hand maneuvers, and consequently did not impeded task performance. In overall, the questionnaire replies indicate that the AR simulator was well accepted both by novices and experts.

With regard to the training tasks, they focus on basic surgical skills such as perception of depth of field and hand eye coordination. These tasks were designed so as to be similar to those offered by commercial VR simulators and box trainers. Although one could certainly develop more complex tasks, our main purpose here was to provide a proof of concept about the educational potential of these tasks in an AR setting. Similarly, the observed metrics are to some extent similar to the metrics employed for performance evaluation in the current VR simulators.

The between-group comparisons of the two experience groups show that the proposed tasks exhibit construct validity. For all metrics, and especially for time and pathlength, our results show significant performance differences between experts and novices. These results can be interpreted as indicative of the simulator's potential to discriminate groups with different level of surgical experience.

Since AR and VR have in essence the same capabilities, the presented system allows the same flexibility in task prototyping that a VR platform would offer. Consequently, besides the three training tasks demonstrated in this Chapter, our future work includes the implementation of additional scenarios that will allow further investigation of the simulator's training efficiency as well as the transferability of the skills acquired with the proposed system to the operating environment. In the same context, we aim to conduct a comparison study that will compare the training value of our simulator with regard to that achieved by a commercial VR simulator. Finally, to further enhance the robustness of our simulator, we aim to improve the stability of the sensors employed, and also improve their design in a modular setup with portable characteristics.

Chapter 11

Research Summary, Conclusions and Future Work

As mentioned in the introductory Chapter, the scope of this thesis was the introduction of AR technology in the field of laparoscopic simulation training. The fact that the existing literature lacks relative works motivated us to follow a rather ambitious goal; the development of the first "Augmented Reality Simulator for Minimally Invasive Surgery". This Chapter summarizes the research process of this thesis including realizations and results, and discusses possible extensions as well as future research directions.

11.1 Research summary, realizations and results

Creating a computer-based surgical simulator from scratch is a challenging and demanding task, requiring involvement and knowledge of a wide range of topics. In general, the main design requirements for developing such simulator can be generally categorized in five fields; computer graphics, physics-based modeling, sensory devices, and finally hardware/software design. In addition to the technical challenges involved, building and validating a surgical simulator for laparoscopic training and assessment also requires background knowledge of the medical and surgical aspects; the psychomotor and cognitive skills that a simulator has to support, and how these skills will be trained and assessed via appropriate training scenarios.

As Chapter 2 discussed, VR-based platforms exist for almost two decades, while current state-of-the art platforms are truly high-end simulator devices reaching levels of sophistication equivalent to those of flight simulators used in the aviation industry. It is rather obvious that efficient solutions regarding each of the essential elements required building a computer-based surgical simulator have already been successfully implemented. The majority however, if not all, of these systems has been developed for commercial purposes and thereby, the amount of information available regarding their practical implementation is very limited. Consequently, in order to investigate the potentials of AR in laparoscopic simulation training, we should improvise on technical solutions that would make the utilization of AR possible. Adding to the difficulty of our thesis primary goal, employment of AR in real-life applications dates back only a few years and critical issues have yet being solved. Based on these considerations, our strategy during this thesis was to identify the essential requirements for the implementation of a basic surgical training framework, and target our research specifically on those vital parts that would allow an efficient prototype implementation. During this process, we aimed to investigate alternative solutions regarding critical simulator components, such as instrument tracking, and perform studies similar to those performed in commercial training platforms: evaluation of construct, content[14] and face validity[15].

Initially, we focused on obtaining some preliminary findings regarding the use of AR in a laparoscopic skills training application. In this context we developed a basic prototype of an AR simulator, including those minimum features that would allow the creation of training scenarios. Incorporating simple AR visualizations within a real box-trainer environment combined with real-time laparoscopic instrument tracking, this prototype allowed the creation of an elementary training scenario, which required surgical trainees to identify and touch virtual spheres superimposed on top of an inanimate stomach model using real laparoscopic instruments. AR visualizations were achieved using the OpenGL library while pattern-marker tracking using

[14] Content validity is the assessment of the appropriateness of a simulator as a teaching modality and involves formal evaluation by experts [33].

[15] Face validity is assessed informally by non-experts and is used to determine the realism of a simulator, or whether the simulator represents what it is supposed to represent [33].

ARToolkit was utilized both for obtaining the spatial relationship between the endoscopic camera and the box-trainer environment, as well as the pose of laparoscopic instruments with respect to this environment. The latter was calculated by a small pattern marker attached on the instruments. During the design stage of this prototype, we also implemented techniques for solving the occlusion problem, a common challenge of any AR application, achieving efficient real-time occlusion handling that resulted in a realistic mixture of virtual objects and real laparoscopic instruments into the same environment (see Chapter 5).

Having developed the first AR training scenario allowed us to perform a comparison study between two groups of novice and experienced laparoscopic surgeons. During this study, we recorded the trajectories of laparoscopic instruments, and analyzed the results using Hidden Markov Models (HMMs) and Dynamic Type Wrapping (DTW). The comparison results demonstrated a clear distinction between novices' and experienced surgeons', providing a first solid indication that AR-based training can be used for classifying expertise in laparoscopic surgery (see Chapter 7). Although preliminary, these findings were important since it was the first study involving a laparoscopic training scenario in which trainees actually interacted with AR graphics in real-time. With this study, we showed that AR can indeed constitute a valid tool for automated objective performance assessment in laparoscopic simulation [157].

The software/hardware development process that preceded the aforementioned study also provided a roadmap for our research, indicating potential weaknesses and possible improvements that should apply towards the fulfillment of our ultimate goal, which was the development of a fully functional AR laparoscopic simulation framework. In this direction, we investigated an alternative instrument tracking solution. Attaching pattern markers on the instruments provided sufficient results in terms of tracking accuracy and frame rate efficiency, but in somehow impractical way; markers obscured the field of view, while in order for tracking to be robust these markers should always be visible to the camera, thus restricting the instruments' available d.o.f. To overcome these practical issues, we developed an instrument tracking technique that employed image processing algorithms for calculating the 3D pose of laparoscopic instruments with respect to the endoscopic camera (see Chapter 8).

Our goal at this research stage was not to present fully-featured scenarios or evaluate the validity of our framework in training or assessing surgical skills training, but to develop proof of concept training tasks for evaluating the accuracy of the instrument tracking method itself, thus illustrating its feasibility for use in our system. In this context, two different AR-based training tasks for basic laparoscopic skills were designed. The first required trainees to touch virtual spheres introduced into a real box-trainer, while the second asked trainees to touch and carry such spheres on a predefined location of the box-trainer. The results of our experiments indicated that vision-based instrument tracking could be applied in AR training scenarios focusing on basic laparoscopic skills such as depth-perception and hand-eye coordination. In terms of accuracy, our method achieved an absolute error of less than 2.5 mm within the area of interest (less than 20cm from the

endoscopic camera). This level of accuracy is more than adequate for implementing simple training scenarios, as well as for consistently solving the AR occlusion issue. In terms of frame rate performance, near real-time efficiency was achieved (21 Hz) with large margins of improvement since the PC utilized in our experiments was slow based on the current standards. The results made clear that an image-based approach for instrument tracking can be used for the creation of simple AR-based training scenarios allowing practice of basic skills such as 2D to 3D perception and hand-eye coordination. Hence, these results illustrated the potential use of AR as a low-cost alternative to VR for basic laparoscopic skills training and assessment [156].

At that stage of our research process, we have created a primitive version of an AR laparoscopic trainer using a relatively simple software/hardware setup. Furthermore, we have demonstrated the potential benefits of introducing AR into laparoscopic simulation training, since our studies have indicated construct validity in the assessment of fundamental laparoscopic skills. The next steps towards our final goal would involve implementation of complex training scenarios similar to the FLS® tasks (see Chapter 2). Achieving this goal required integration of physics-based interactions between instruments and virtual objects into our simulation framework. Physics-based modeling is a perquisite for the implementation of training tasks involving grabbing and lifting virtual objects, or interactions with deformable structures. The latter is essential for the implementation of tasks focusing on procedural skills, such as clipping, cutting etc.

The integration of real-time physics into our framework required further improvements on the instrument tracking methodology. Although experiments both with pattern-marker tracking and image-based tracking showed that these two methods can constitute efficient solutions in simple scenarios, none of them provided sufficient accuracy and stability for utilization in an AR environment where behavior of virtual objects needed to be dynamically simulated. Hence we turned our focus on advanced solutions similar to those used in commercial surgical simulators. Specifically, we concluded on the integration of electromagnetic (EM) sensors, making a compromise between cost and efficiency. The final version of our simulation framework, utilized Ogre3D for advanced graphics quality, Bullet Physics Engine for real-time simulation of virtual objects physical behavior and EM sensors for real-time tracking of laparoscopic instruments.

As described extensively in Chapters 4-6, the final result was a fully-featured laparoscopic simulation platform, providing equivalent capabilities and flexibility in training task prototyping ~~as~~ similar to that of commercial simulators. To prove this claim, we implemented two tasks aiming on basic psychomotor skills' training (instrument navigation, depth perception and hand-eye coordination) and one task focusing on procedural skills (artery clip application and cutting) and evaluated the construct validity of our framework through a statistical comparison between two experience groups. The first consisted of ten experienced surgeons and the second included ten individuals with no experience in laparoscopic surgery. The performance metrics used for this evaluation study included the two task-specific errors for each training scenario, as well as the task completion time and the total pathlength of the laparoscopic instruments. Results indicated high

construct validity since in all tasks and all performance metrics, experts significantly outperformed novices. In addition to performance comparison, in this study we included a questionnaire both for experts and novices, which provided significant feedback regarding our simulators content and face validity (see Chapter 9).

During the development project of our surgical simulation framework, we also invented and implemented a calibration method for obtaining the pose of laparoscopic instruments with respect to EM sensors attached on the instruments' handles. This calibration method illustrated sub-millimeter accuracy in estimating the instruments' tooltip 3D position, and sub-degree accuracy across the three cardinal axes in the estimation of the instruments' orientation. The achieved levels of accuracy indicate that the proposed method, except for integration into our framework, has also potential use for objective assessment of the trainee's performance in standard box trainers, where essential information about the tip position cannot be extracted.

It is important to note that in the final setup of our simulator, the occlusion handling technique utilized in the previous versions did not provide satisfactory results in terms of visual realism. The reason was the increased levels of virtual objects details (as compared to the simple virtual spheres used in previous versions) that created the need of higher levels of accuracy. Despite the fact that we utilized EM sensors, registration errors arising from small inaccuracies in instrument tracking, coupled with inaccuracies from camera calibration and pattern-marker tracking (see Chapter 5) resulted in an unrealistic occlusion handling outcome. A solution regarding this issue is proposed in the section discussing future research directions.

11.2 Conclusions

In this thesis, we have designed, developed and evaluated a prototype of a surgical simulation platform for training and assessment of fundamental laparoscopic skills. To the best of our knowledge, the outcome of this thesis is the first genuine AR-based training system that allows the user to interact in real-time with rigid and deformable VR models. Our platform provides a solid basis for the development of training scenarios for psychomotor as well as procedural skills, utilizing a standard laparoscopic box-trainer, standard laparoscopic instruments, and relatively minimal sensory equipment.

Based on comparison studies performed in the context of this thesis, and the user feedback that we obtained during these studies, allow as to safely reach the conclusion that AR has strong educational potentials and could be used as a counterpart of PR and VR in laparoscopic simulation training, since it demonstrates important advantages compared to these modalities. Compared to VR, an important advantage of AR is the increased sense of visual realism, emerging from the realistic mixture of real world and VR models that this technology provides. The improved visual feedback translates into a better understanding regarding the position of the VR objects, which in

turn provides the trainee enhanced perception of depth compared to purely virtual environments. Additionally, an AR-based training platform based on real laparoscopic instruments, allows trainees to gain better familiarization with the actual operating conditions due to the actual laparoscopic instrumentation employed to perform the tasks. Finally, in this thesis we have proved that building an AR surgical simulator can be achieved with a significantly lowered cost of hardware compared to commercially available simulation platforms. Compared to PR box-trainers, an AR laparoscopic simulation platform demonstrates the same advantages as those demonstrated by VR. Namely, it provides greater flexibility in the design of training scenarios, it allows implementation of scenarios focusing on procedural skills, it is a more practical training environment since it does not require the use (and substitution) of physical training models, and finally it allows the employment of automated performance assessment methods.

However, along with its advantages, AR demonstrates some drawbacks and limitations compared to PR and VR. In both these modalities, haptic feedback during interaction with training models is provided, in PR as a natural effect and in VR using force feedback mechanism. In AR on the other hand, natural force feedback does not exist since trainees interact mostly with virtual objects, while the existing commercial solution for mechanical force feedback cannot be integrated into an AR setup. The reason is that the existing force feedback devices need to be positioned in front of the laparoscopic instruments. Hence, the existence of such devices within the training environment severely obscures the trainees' visual field of view. Overcoming this limitation requires a mechanism that will provide force feedback, designed however in such way that it could be placed outside the training environment or alternatively, a device small enough not to be visible from the endoscope when placed inside the box-trainer. To the best of our knowledge a device with such characteristics is not currently available.

Another potential drawback of AR as compared to VR (which also stands for PR), is the practical difficulty of implementing training scenarios that would focus on simulating actual laparoscopic operations. In Chapter 2, we described how surgical operations such as laparoscopic cholecystectomy etc. can be performed in the purely virtual environment of commercial VR trainers. Creating an equivalent training scenario in an AR environment would require the introduction of virtual abdominal organs within a real surrounding environment (abdomen). It is evident however that implementing such a setup is practically impossible without using real human cadavers.

11.3 Future Research Goals and Directions

Future work should focus on two directions. First, technical improvements should be implemented, aiming to improve the overall simulation quality and provide the opportunity of creating additional training scenarios. Second, further studies regarding the educational efficiency of the proposed framework must be performed.

The main technical challenge that should be addressed is the problem of AR occlusion. As stated earlier, during this thesis we have achieved efficient occlusion handling in training tasks involving simple-shaped virtual models and little interaction between them and the laparoscopic instruments. However, in the final version of our platform where trainees could perform complex actions such as grasping, clipping and cutting, the same occlusion handling technique did not provide a satisfactory visual outcome. This has mainly to do with the way the occlusion problem is addressed; the virtual colorless representations of laparoscopic instruments (see Chapter 5) must be perfectly aligned in terms of position and orientation with their real counterparts. Although our calibration method achieved spatial registration accurate enough for implementing training scenarios, even the smallest inaccuracies resulted in slight miss-registrations, which are visible by the human eye. To overcome this problem, perfect 3D modeling of laparoscopic instruments is required along with a zero-error calibration of the spatial relationship between instruments and tracking devices (EM sensors). This can only be achieved using a precisely engineered system of instruments and sensors, with manufacturing tolerances similar to those employed in VR platforms. In addition, an errorless camera calibration technique must be utilized (see Chapter 5).

In the context of this thesis, we focused our research on the development of simple tasks for fundamental skills training, designing tasks for instrument navigation and object transferring. Except for basic skills however, modern VR laparoscopic trainers also provide tasks for procedural skills training as well as the opportunity to perform a complete surgical operation within a virtual environment. And while the latter might be impractical in AR (see previous section), a true counterpart of VR trainers based on the AR technology should provide equivalent opportunities for trainees for basic and procedural skills. In the final version of the presented simulation framework, we performed an important step towards this direction by creating a simple procedural task that involved clipping and cutting of a virtual artery, introduced into the real environment of the box-trainer. This training scenario was a proof-of-concept regarding the potentials of our platform and the capabilities of AR in general, however lacking in terms of both visual realism as well as behavioral realism in comparison to equivalent tasks of commercial VR trainers.

Thus, a future improvement should be the integration of a more efficient method for the simulation of deformable virtual objects. This is of crucial importance for the implementation of training scenarios involving complex deformable geometries, similar to the procedural tasks of state-of-the art VR trainers. The current version of the Bullet library utilized in our framework does not provide adequate frame rate efficiency when shapes of high levels of details are simulated. Consequently, real-time interaction between such shapes and the laparoscopic instruments is problematic, since unrealistic effects occur. According to Bullet developers, the next version (v.3.0) of the library will provide an optimized soft-body implementation, utilizing a better exploitation of the capabilities of new generation GPUs. As a first step, our framework should be modified to integrate the next Bullet version, and tests must be performed to evaluate its frame-rate efficiency

in simulating complex structures such as models of human anatomy. This is a perquisite for the creation of a wider range of training scenarios within our framework, focusing on procedural skills such as suturing, knot tying etc. In case this modification proves inefficient, alternative solutions must be examined.

Finally, in addition to the technical improvements, effort must be put on the validation of the educational value of the platform. In the context of this thesis, the proposed platform has been mostly evaluated in terms of assessment efficiency. Following standard study protocols, we concluded that our framework demonstrates construct validity. A complete investigation regarding its educational value and potentials however should include studies that will compare the training value of our platform with that achieved by PR and VR modalities. The protocols for performing such comparisons are common in the current literature. They are used extensively for the evaluation of VR trainers against PR box-trainers. Hence, similar protocols and studies must be performed to assess the educational value of our AR surgical simulation platform.

List of figures

This page has been intentionally left blank

References

1. Lau WY, Leow CK, Li AK (1997) History of endoscopic and laparoscopic surgery. World J Surg 21:444–53.

2. Feuerstein M (2007) Augmented Reality in Laparoscopic Surgery: New Concepts for Intraoperative Multimodal Imaging.

3. Radojcić B, Jokić R, Grebeldinger S, Meljnikov I, Radojić N [History of minimally invasive surgery]. Med Pregl 62:597–602.

4. Blum CA, Adams DB (2011) Who did the first laparoscopic cholecystectomy? J Minim Access Surg 7:165–8. doi: 10.4103/0972-9941.83506

5. Bhattacharya K (2007) Kurt Semm: A laparoscopic crusader. J Minim Access Surg 3:35–6. doi: 10.4103/0972-9941.30686

6. Litynski GS Erich Mühe and the rejection of laparoscopic cholecystectomy (1985): a surgeon ahead of his time. JSLS 2:341–6.

7. Park AE, Lee TH (2011) Minimally Invasive Surgical Oncology. doi: 10.1007/978-3-540-45021-4

8. Aggarwal R, Moorthy K, Darzi A (2004) Laparoscopic skills training and assessment. Br J Surg 91:1549–1558. doi: 10.1002/bjs.4816

9. TANG B, CUSCHIERI A (2006) Conversions During Laparoscopic Cholecystectomy: Risk Factors and Effects on Patient Outcome. J Gastrointest Surg 10:1081–1091. doi: 10.1016/j.gassur.2005.12.001

10. Tanaka S, Matsuo K, Sasaki T, Nakano M, Shimura H, Yamashita Y Clinical outcomes and advantages of laparoscopic surgery for primary Crohn's disease: are they significant? Hepatogastroenterology 56:416–20.

11. Braga M, Vignali A, Zuliani W, Frasson M, Di Serio C, Di Carlo V (2005) Laparoscopic versus open colorectal surgery: cost-benefit analysis in a single-center randomized trial. Ann Surg 242:890–5, discussion 895–6.

12. Perugini RA, Callery MP (2001) Complications of laparoscopic surgery.

13. Jönsson B, Zethraeus N (2000) Costs and benefits of laparoscopic surgery--a review of the literature. Eur J Surg Suppl 48–56.

14. Agha R, Muir G (2003) Does laparoscopic surgery spell the end of the open surgeon? J R Soc Med 96:544–6.

15. Derossis AM, Fried GM, Abrahamowicz M, Sigman HH, Barkun JS, Meakins JL (1998) Development of a model for training and evaluation of laparoscopic skills. Am J Surg 175:482–7. doi: S0002961098000804 [pii]

16. Tendick F, Downes M, Goktekin T, Cavusoglu MC, Feygin D, Wu X, Eyal R, Hegarty M, Way LW (2000) A Virtual Environment Testbed for Training Laparoscopic Surgical Skills. Presence Teleoperators Virtual Environ 9:236–255. doi: 10.1162/105474600566772

17. Liu A, Tendick F, Cleary K, Kaufmann C (2003) A Survey of Surgical Simulation: Applications, Technology, and Education. Presence 12:599–614.

18. King RC, Atallah L, Lo BPL, Yang G (2009) Surgical Skills Assessment. 13:673–679.

19. Crochet P, Aggarwal R, Dubb SS, Ziprin P, Rajaretnam N, Grantcharov T, Ericsson KA, Darzi A (2011) Deliberate practice on a virtual reality laparoscopic simulator enhances the quality of surgical technical skills. Ann Surg 253:1216–22. doi: 10.1097/SLA.0b013e3182197016

20. Soler L, Marescaux J (2008) Patient-specific surgical simulation. World J Surg 32:208–212. doi: 10.1007/s00268-007-9329-3

21. Debes AJ, Aggarwal R, Balasundaram I, Jacobsen MB (2010) A tale of two trainers: virtual reality versus a video trainer for acquisition of basic laparoscopic skills. Am J Surg 199:840–5. doi: 10.1016/j.amjsurg.2009.05.016

22. Aggarwal R, Leong J, Leff D, Warren O, Yang G-Z, Darzi A (2008) New technologies for the surgical curriculum. World J Surg 32:213–6. doi: 10.1007/s00268-007-9338-2

23. Figert PL, Park AE, Witzke DB, Schwartz RW (2001) Transfer of training in acquiring

laparoscopic skills. J Am Coll Surg 193:533–7.

24. Aggarwal R, Darzi A (2005) Organising a surgical skills centre. Minim Invasive Ther Allied Technol 14:275–9. doi: 10.1080/13645700500236822

25. Cuschieri A, Shapiro S (1995) Extracorporeal pneumoperitoneum access bubble for endoscopic surgery. Am J Surg 170:391–4.

26. Jansen FW, Kolkman W (2008) Implementation difficulties of advanced techniques in gynecological laparoscopy. Gynecol Surg 5:261–264. doi: 10.1007/s10397-008-0390-1

27. Ashraf a., Collins D, Whelan M, O'Sullivan R, Balfe P (2015) Three-dimensional (3D) simulation versus two-dimensional (2D) enhances surgical skills acquisition in standardised laparoscopic tasks: A before and after study. Int J Surg 14:12–16. doi: 10.1016/j.ijsu.2014.12.020

28. Munz Y, Almoudaris AM, Moorthy K, Dosis A, Liddle AD, Darzi AW (2007) Curriculum-based solo virtual reality training for laparoscopic intracorporeal knot tying: objective assessment of the transfer of skill from virtual reality to reality. Am J Surg 193:774–83. doi: 10.1016/j.amjsurg.2007.01.022

29. Sarker SK, Chang a, Vincent C, Darzi a W (2005) Technical skills errors in laparoscopic cholecystectomy by expert surgeons. Surg Endosc 19:832–5. doi: 10.1007/s00464-004-9174-5

30. Stefanidis D, Heniford BT (2009) The formula for a successful laparoscopic skills curriculum. Arch Surg 144:77–82; discussion 82. doi: 10.1001/archsurg.2008.528

31. Halvorsen FH, Elle OJ, Fosse E (2005) Simulators in surgery. Minim Invasive Ther Allied Technol 14:214–23. doi: 10.1080/13645700500243869

32. Halvorsen FH Virtual Reality Simulation in Laparoscopic Surgical Education Department of Cardiothoracic Surgery Oslo University Hospital , Rikshospitalet Department of Surgery Kristiansand Hospital Faculty of Medicine , University of Oslo.

33. McGreevy JM (2005) The aviation paradigm and surgical education. J Am Coll Surg 201:110–7. doi: 10.1016/j.jamcollsurg.2005.02.024

34. C.E. Buckley ENDR and PCN (2012) Virtual Reality in Psychological, Medical and Pedagogical Applications. doi: 10.5772/2607

35. Fried GM, Feldman LS, Vassiliou MC, Fraser SA, Stanbridge D, Ghitulescu G, Andrew CG (2004) Proving the value of simulation in laparoscopic surgery. Ann Surg 240:518. doi: 00000658-200409000-00013 [pii]

36. SAGES - Society of American Gastrointestinal and Endoscopic Surgeons. http://www.sages.org/. Accessed 1 Oct 2015

37. EAES - Home. http://www.eaes-eur.org/Home.aspx. Accessed 1 Oct 2015

38. Palter VN, Grantcharov TP (2010) Simulation in surgical education. CMAJ 182:1191–6. doi: 10.1503/cmaj.091743

39. Jakimowicz J, Fingerhut a (2009) Simulation in surgery. Br J Surg 96:563–4. doi: 10.1002/bjs.6616

40. (2006) Haptics and Physical Simulation. PHD thesis

41. Pellen M, Horgan L, Roger Barton J, Attwood S (2009) Laparoscopic surgical skills assessment: Can simulators replace experts? World J Surg 33:440–447. doi: 10.1007/s00268-008-9866-4

42. Satava RM (2007) Historical Review of Surgical Simulation—A Personal Perspective. World J Surg 32:141–148. doi: 10.1007/s00268-007-9374-y

43. Reznick RK (1993) Teaching and testing technical skills. Am J Surg 165:358–61.

44. Martin J a, Regehr G, Reznick R, MacRae H, Murnaghan J, Hutchison C, Brown M (1997) Objective structured assessment of technical skill (OSATS) for surgical residents. Br J Surg 84:273–8.

45. Arikatla V (2013) Towards virtual FLS: development of a peg transfer simulator. … Int J …. doi: 10.1002/rcs

46. Ragle CA (2012) Advances in Equine Laparoscopy. John Wiley & Sons, Inc., West Sussex, UK

47. Fundamentals of Laparoscopic Surgery - ...the definitive laparoscopic skills enhancement and assessment module. http://www.flsprogram.org/. Accessed 1 Oct 2015

48. de Montbrun S, MacRae H (2012) Simulation in Surgical Education. Clin Colon Rectal Surg 25:156–165. doi: 10.1055/s-0032-1322553

49. Hanna L (2010) Simulated surgery: the virtual reality of surgical training. Surg 28:463–468. doi: 10.1016/j.mpsur.2010.05.004

50. Arikatla VS, Ahn W, Sankaranarayanan G, De S (2013) Towards virtual FLS: development of a peg transfer simulator. Int J Med Robot Comput Assist Surg 10:344–355.

51. Delp SL, Loan JP, Hoy MG, Zajac FE, Topp EL, Rosen JM (1990) An interactive graphics-based model of the lower extremity to study orthopaedic surgical procedures. IEEE Trans Biomed Eng 37:757–767. doi: 10.1109/10.102791

52. Satava RM (1993) Virtual reality surgical simulator. Surg Endosc 7:203–205.

53. Seymour NE (2008) VR to OR: a review of the evidence that virtual reality simulation improves operating room performance. World J Surg 32:182–188. doi: 10.1007/s00268-007-9307-9

54. Wilson MS, Middlebrook A, Sutton C, Stone R, McCloy RF (1997) MIST VR: a virtual reality trainer for laparoscopic surgery assesses performance. Ann R Coll Surg Engl 79:403–4.

55. Seymour NE, Gallagher AG, Roman SA, O'Brien MK, Bansal VK, Andersen DK, Satava RM, O'Brien MK, Bansal VK, Andersen DK, Satava RM (2002) Virtual reality training improves operating room performance: results of a randomized, double-blinded study. Ann Surg 236:458. doi: 10.1097/01.SLA.0000028969.51489.B4

56. Loukas C, Nikiteas N, Schizas D, Lahanas V, Georgiou E (2012) A head-to-head comparison between virtual reality and physical reality simulation training for basic skills acquisition. Surg Endosc 26:2550–8. doi: 10.1007/s00464-012-2230-7

57. Loukas C, Lahanas V, Kanakis M, Georgiou E (2014) The Effect of Mixed-Task Basic Training

in the Acquisition of Advanced Laparoscopic Skills. Surg Innov. doi: 10.1177/1553350614556365

58. Arikatla VS, Sankaranarayanan G, Ahn W, Chellali A, De S, Caroline GL, Hwabejire J, DeMoya M, Schwaitzberg S, Jones DB (2013) Face and construct validation of a virtual peg transfer simulator. Surg Endosc 27:1721–9. doi: 10.1007/s00464-012-2664-y

59. Barsuk JH, McGaghie WC, Cohen ER, Balachandran JS, Wayne DB (2009) Use of simulation-based mastery learning to improve the quality of central venous catheter placement in a medical intensive care unit. J Hosp Med 4:397–403. doi: 10.1002/jhm.468

60. van Dongen K (2010) Virtual reality training for endoscopic surgery : composing a validated training program for basic skills. doi: 10.3990/1.9789036530620

61. Munz Y, Kumar BD, Moorthy K, Bann S, Darzi a (2004) Laparoscopic virtual reality and box trainers: is one superior to the other? Surg Endosc 18:485–94. doi: 10.1007/s00464-003-9043-7

62. Woodrum DT, Andreatta PB, Yellamanchilli RK, Feryus L, Gauger PG, Minter RM (2006) Construct validity of the LapSim laparoscopic surgical simulator. Am J Surg 191:28–32.

63. Zhou F, Dun HBL, Billinghurst M (2008) Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR. Proc - 7th IEEE Int Symp Mix Augment Real 2008, ISMAR 2008 193–202. doi: 10.1109/ISMAR.2008.4637362

64. Höllerer TH, Feiner SK (2004) Mobile Augmented Reality. 1–39.

65. van Krevelen DWF, Poelman R, Krevelen DWF Van, Poelman R (2010) A Survey of Augmented Reality Technologies, Applications and Limitations. Int J Virtual Real 9:1–20. doi: citeulike-article-id:8213624

66. Sutherland IE (1968) A head-mounted three dimensional display. In: Proc. December 9-11, 1968, fall Jt. Comput. Conf. part I - AFIPS '68 (Fall, part I). ACM Press, New York, New York, USA, p 757

67. Gordon I (2004) Augmenting Reality, Naturally.

68. Domhan T (2010) Augmented Reality on Android Smartphones. 47.

69. Azuma RR, Azuma RR (1997) A survey of augmented reality. Presence 4:355–385. doi: 10.1.1.30.4999

70. Hirzer M (2008) Marker Detection for Augmented Reality Applications. 27.

71. Kalkofen D, Sandor C, White S, Schmalstieg D (2011) Handbook of Augmented Reality. doi: 10.1007/978-1-4614-0064-6

72. Azuma R, Baillot Y (2001) Recent advances in augmented reality. Comput. Graph. (ACM).

73. Kuusisto R, Kuusisto R (2015) CREATING AUGMENTED.

74. Paul Milgram HTAUFK Augmented Reality: A Class of Displays on the Reality-Virtuality Continuum.

75. Azuma R, Baillot Y, Behringer R, Feiner S, Julier S, MacIntyre B (2001) Recent advances in augmented reality. IEEE Comput Graph Appl. doi: 10.1109/38.963459

76. Lahanas V, Loukas C, Smailis N, Georgiou E (2014) A novel augmented reality simulator for skills assessment in minimal invasive surgery. Surg Endosc. doi: 10.1007/s00464-014-3930-y

77. Samset E, Schmalstieg D, Vander Sloten J, Freudenthal A, Declerck J, Casciaro S, Rideng Ø, Gersak B (2008) Augmented Reality in Surgical Procedures. Proc. SPIE Hum. Vis. Electron. Imaging XIII 6806:

78. Nolle S, Klinker G (2006) Augmented reality as a comparison tool in automotive industry. … Augment Reality, 2006 ISMAR 2006 … 249–250. doi: 10.1109/ismar.2006.297829

79. Kato H, Billinghurst M (1999) Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In: Proc. 2nd IEEE ACM Int. Work. Augment. Real. IEEE Comput. Soc, pp 85–94

80. Fiala M (2004) ARTag, An Improved Marker Based System Based on ARToolkit. System 47166:36.

81. Wang HL, Sengupta K, Kumar P, Sharma R (2005) Occlusion Handling in Augmented Reality Using Background-Foreground Segmentation and Projective Geometry. Presence Teleoperators Virtual Environ 14:264–277. doi: 10.1162/105474605323384636

82. Nicolau S a., Pennec X, Soler L, Buy X, Gangi a., Ayache N, Marescaux J (2009) An augmented reality system for liver thermal ablation: Design and evaluation on clinical cases. Med Image Anal 13:494–506. doi: 10.1016/j.media.2009.02.003

83. Vallino JR (1998) Interactive Augmented Reality. Computer (Long Beach Calif) 76:22,24. doi: 10.1111/j.1540-5834.2011.00603.x

84. Bell B, Feiner S, Hollerer T (2002) Information at a glance [augmented reality user interfaces]. IEEE Comput Graph Appl 22:6–9. doi: 10.1109/MCG.2002.1016691

85. Vlahakis V, Karigiannis J, Tsotros M, Gounaris M, Almeida L, Stricker D, Gleue T, Christou IT, Carlucci R, Ioannidis N (2001) Archeoguide: first results of an augmented reality, mobile computing system in cultural heritage sites. … Cult Herit 131–140. doi: 10.1145/584993.585015

86. Nicolau S, Soler L, Mutter D, Marescaux J (2011) Augmented reality in laparoscopic surgical oncology. Surg Oncol 20:189–201. doi: 10.1016/j.suronc.2011.07.002

87. Shuhaiber JH (2004) Augmented reality in surgery. Arch Surg 139:170–174. doi: 10.1001/archsurg.139.2.170139/2/170 [pii]

88. Friets EM, Strohbehn JW, Hatch JF, Roberts DW (1989) A frameless stereotaxic operating microscope for neurosurgery. IEEE Trans Biomed Eng 36:608–17. doi: 10.1109/10.29455

89. Bajura M, Fuchs H, Ohbuchi R (1992) Merging virtual objects with the real world. ACM SIGGRAPH Comput Graph 26:203–210. doi: 10.1145/142920.134061

90. Coles T, John N, Caldwell D (2011) Investigating Augmented Reality Visio-Haptic Techniques for Medical Training.

91. Mischkowski R a, Zinser MJ, Kübler AC, Krug B, Seifert U, Zöller JE, Kubler AC, Krug B, Seifert

U, Zoller JE (2006) Application of an augmented reality tool for maxillary positioning in orthognathic surgery - a feasibility study. J Craniomaxillofac Surg 34:478–83. doi: 10.1016/j.jcms.2006.07.862

92. Nicolau S, Pennec X, Soler L, Ayache N (2005) A complete augmented reality guidance system for liver punctures: First clinical evaluation. Med Image Comput … 539–547.

93. Kessel PGÆD, Magee D, Zhu Y, Ratnalingam R, Gardner P, Kessel D (2007) An augmented reality simulator for ultrasound guided needle placement training. Med Biol Eng Comput 45:957–967. doi: 10.1007/s11517-007-0231-9

94. Khamene A, Wacker F, Vogt S, Azar F, Wendt M, Sauer F, Lewin J (2003) An Augmented Reality system for MRI-guided needle biopsies. Stud Health Technol Inform 94:151–7.

95. Soler L, Nicolau S, Schmid J, Koehl C, Marescaux J, Civil I, Hopital P De, Cedex S, Pennec X, Ayache N, Sophia I, Lucioles R (2004) Virtual reality and augmented reality in digestive surgery. In: Third IEEE ACM Int. Symp. Mix. Augment. Real. (ISMAR 2004). Washington, DC, USA , pp 278–279

96. Doyle WK (1996) Low end interactive image-directed neurosurgery. Update on rudimentary augmented reality used in epilepsy surgery. Stud Health Technol Inform 29:1–11.

97. Lavallée S, Cinquin P, Szeliski R, Peria O, Hamadeh A, Champleboux G, Troccaz J (1995) Building a hybrid patient's model for augmented reality in surgery: a registration problem. Comput Biol Med 25:149–64.

98. Ponce BA, Jennings JK, Clay TB, May MB, Huisingh C, Sheppard ED (2014) Telementoring: use of augmented reality in orthopaedic education: AAOS exhibit selection. J Bone Joint Surg Am 96:e84. doi: 10.2106/JBJS.M.00928

99. Wang A, Mirsattari SM, Parrent AG, Peters TM (2011) Fusion and visualization of intraoperative cortical images with preoperative models for epilepsy surgical planning and guidance. Comput Aided Surg 16:149–160. doi: 10.3109/10929088.2011.585805

100. Schneider A, Pezold S, Saner A, Ebbing J, Wyler S, Rosenthal R, Cattin PC (2014) Augmented reality assisted laparoscopic partial nephrectomy. Med Image Comput Comput Assist Interv 17:357–64.

101. Bichlmeier C, Ben O, Heining SM, Ahmadi A, Navab N (2008) Stepping into the operating theater: ARAV - Augmented Reality Aided Vertebroplasty. Proc - 7th IEEE Int Symp Mix Augment Real 2008, ISMAR 2008 1:165–166. doi: 10.1109/ISMAR.2008.4637348

102. Talbot J, Meyer J, Watts R, Grasset R (2009) An augmented reality application for patient positioning and monitoring in radiotherapy. IFMBE Proc 25:620–623. doi: 10.1007/978-3-642-03474-9-174

103. Freysinger W, Gunkel AR, Thumfart WF (1997) Image-guided endoscopic ENT surgery. Eur Arch Oto-rhino-laryngology 254:343–346. doi: 10.1007/BF02630726

104. Fuchs H, Livingston MA, Raskar R, Colucci D, Keller K, State A, Crawford JR, Rademacher P, Drake SH, Meyer AA (1998) Augmented reality visualization for laparoscopic surgery. … Image Comput … 934–943.

105. Traub J, Feuerstein M, Bauer M, Schirmbeck E., Najafi H, Bauernschmitt R, Klinker G (2004) Augmented reality for port placement and navigation in robotically assisted minimally invasive cardiovascular surgery. Int Congr Ser 1268:735–740. doi: 10.1016/j.ics.2004.03.049

106. Feuerstein M, Mussack T, Heining SM, Navab N (2008) Intraoperative laparoscope augmentation for port placement and resection planning in minimally invasive liver resection. IEEE Trans Med Imaging 27:355–69. doi: 10.1109/TMI.2007.907327

107. Katic D, Wekerle A-LL, Gortler J, Spengler P, Bodenstedt S, Rohl S, Suwelack S, Kenngott HG, Wagner M, Muller-Stich BP, Dillmann R, Speidel S, Katić D, Wekerle A-LL, Görtler J, Spengler P, Bodenstedt S, Röhl S, Suwelack S, Kenngott HG, Wagner M, Müller-Stich BP, Dillmann R, Speidel S (2013) Context-aware Augmented Reality in laparoscopic surgery. Comput Med Imaging Graph 37:174–82. doi: 10.1016/j.compmedimag.2013.03.003

108. Dohi T, Kikinis R (2002) Medical Image Computing and Computer-Assisted Intervention — MICCAI 2002. doi: 10.1007/3-540-45787-9

109. Ferrari V, Megali G, Troia E, Pietrabissa A, Mosca F (2009) A 3-D Mixed-Reality System for Stereoscopic Visualization of Medical Dataset. IEEE Trans Biomed Eng 56:2627–2633. doi: 10.1109/TBME.2009.2028013

110. Masamune K, Sato I, Liao H, Dohi T (2008) Medical Imaging and Augmented Reality. doi: 10.1007/978-3-540-79982-5

111. Murgante B, Gervasi O, Iglesias A, Taniar D, Apduhan BO (2011) Computational Science and Its Applications - ICCSA 2011. doi: 10.1007/978-3-642-21898-9

112. Marescaux J, Rubino F, Arenas M, Mutter D, Soler L (2004) Augmented-reality-assisted laparoscopic adrenalectomy. JAMA 292:2214–5. doi: 10.1001/jama.292.18.2214-c

113. Cano AM, Gayá F, Lamata P, Sánchez-González P, Gómez EJ (2008) Laparoscopic tool tracking method for augmented reality surgical applications. In: Bello F, Edwards E (eds) Lect. Notes Comput. Sci. Springer-Verlag, Berlin Heidelberg, pp 191–196

114. Baumhauer M, Feuerstein M, Meinzer H-P, Rassweiler J (2008) Navigation in endoscopic soft tissue surgery: perspectives and limitations. J Endourol 22:751–766. doi: 10.1089/end.2007.9827

115. Mirota DJ, Ishii M, Hager GD (2011) Vision-based navigation in image-guided interventions. Annu Rev Biomed Eng 13:297–319. doi: 10.1146/annurev-bioeng-071910-124757

116. Chmarra MK, Grimbergen C a, Dankelman J (2007) Systems for tracking minimally invasive surgical instruments. Minim Invasive Ther Allied Technol 16:328–40. doi: 10.1080/13645700701702135

117. Cano AMA, Gayá F, Lamata P, Sánchez-González P, Gómez EJ (2008) Laparoscopic tool tracking method for augmented reality surgical applications. Biomed … 5104 LNCS:191–196. doi: 10.1007/978-3-540-70521-5_21

118. Allen BF, Kasper F, Nataneli G, Dutson E, Faloutsos P (2011) Visual tracking of laparoscopic instruments in standard training environments. Stud Heal Technol Inf 163:11–17. doi: 10.3233/978-1-60750-706-2-11

119. Cano AM, Lamata P, Gayá F, Gómez EJ (2006) New methods for video-based tracking of laparoscopic tools. In: Harders M, Székely G (eds) Lect. Notes Comput. Sci. Springer-Verlag, Berlin Heidelberg, pp 142–149

120. Oropesa I, Sánchez-González P, Chmarra MK, Lamata P, Fernández Á, Sánchez-Margallo JA, Jansen FW, Dankelman J, Sánchez-Margallo FM, Gómez EJ (2013) EVA: Laparoscopic instrument tracking based on endoscopic video analysis for psychomotor skills assessment. Surg Endosc Other Interv Tech 27:1029–1039. doi: 10.1007/s00464-012-2513-z

121. Pérez F, Sossa H, Martínez R, Lorias D, Minor A (2013) Video-Based Tracking of Laparoscopic Instruments Using an Orthogonal Webcams System. 129–132.

122. Shin S, Kim Y, Cho H, Lee D, Park S, Kim GJ, Kim L (2014) A single camera tracking system for 3D position, grasper angle, and rolling angle of laparoscopic instruments. Int J Precis Eng Manuf 15:2155–2160. doi: 10.1007/s12541-014-0576-6

123. Voros S, Long J-A, Cinquin P (2007) Automatic detection of instruments in laparoscopic images. Int J Rob Res 26:1173–1190.

124. Chmarra MK, Bakker NH, Grimbergen C a., Dankelman J (2006) TrEndo, a device for tracking minimally invasive surgical instruments in training setups. Sensors Actuators A Phys 126:328–334. doi: 10.1016/j.sna.2005.10.040

125. Groeger M, Arbter K, Hirzinger G (2008) Motion tracking for minimally invasive robotic surgery. InTech-Open Access Publisher, Rijeka, Croatia

126. Birkfellner W, Hummel J, Wilson E (2008) Chapter 2 Tracking Devices. 23–45.

127. Shin S, Kim Y, Kwak H, Lee D, Park S (2011) 3D tracking of surgical instruments using a single camera for laparoscopic surgery simulation. Stud Health Technol Inform 163:581–7.

128. Nicolau SA, Goffin L, Soler L (2005) A Low Cost and Accurate Guidance System for Laparoscopic Surgery : Validation on an Abdominal Phantom. Proc ACM Symp Virtual Real Softw Technol 124–133. doi: 10.1145/1101616.1101642

129. Van Sickle KR, McClusky D a, Gallagher a G, Smith CD, McClusky 3rd DA, Gallagher a G, Smith CD (2005) Construct validation of the ProMIS simulator using a novel laparoscopic suturing task. Surg Endosc 19:1227–31. doi: 10.1007/s00464-004-8274-6

130. CAE. http://www.cae.com/. Accessed 22 Sep 2015

131. Botden SMBI, Buzink SN, Schijven MP, Jakimowicz JJ (2007) Augmented versus virtual reality laparoscopic simulation: what is the difference? A comparison of the ProMIS augmented reality laparoscopic simulator versus LapSim virtual reality laparoscopic simulator. World J Surg 31:764–72. doi: 10.1007/s00268-006-0724-y

132. Botden SMBI, De Hingh IHJT, Jakimowicz JJ (2009) Meaningful assessment method for laparoscopic suturing training in augmented reality. Surg Endosc Other Interv Tech 23:2221–2228. doi: 10.1007/s00464-008-0276-3

133. Botden SMBI, Buzink SN, Schijven MP, Jakimowicz JJ (2008) ProMIS augmented reality training of laparoscopic procedures face validity. Simul Healthc 3:97–102. doi: 10.1097/SIH.0b013e3181659e91

134. Soler L, Nicolau SS, Fasquel J, Agnus V, Charnoz A, Hostettler A, Moreau J, Forest CCC, Mutter D, Marescaux J (2008) Virtual reality and augmented reality applied to laparoscopic and notes procedures. 2008 5th IEEE Int Symp Biomed Imaging From Nano to Macro 1399–1402. doi: 10.1109/ISBI.2008.4541267

135. Botden SMBI, Jakimowicz JJ (2009) What is going on in augmented reality simulation in laparoscopic surgery? Surg Endosc 23:1693–700. doi: 10.1007/s00464-008-0144-1

136. Surgical Science - The global leader in medical simulation training. http://www.surgical-science.com/. Accessed 22 Sep 2015

137. Rosen J, Brown JD, Barreca M, Chang L, Hannaford B, Sinanan M (2002) The Blue DRAGON--a system for monitoring the kinematics and the dynamics of endoscopic tools in minimally invasive surgery for objective laparoscopic skill assessment. Stud Heal Technol Inf 85:412–418. doi: 10.3233/978-1-60750-929-5-412

138. Soyinka AS, Schollmeyer T, Meinhold-Heerlein I, Gopalghare D V, Hasson H, Mettler L (2008) Enhancing laparoscopic performance with the LTS3E: a computerized hybrid physical reality simulator. Fertil Steril 90:1988–94. doi: 10.1016/j.fertnstert.2007.08.077

139. Stylopoulos N, Cotin S, Maithel SKK, Ottensmeye M, Jackson PGG, Bardsley RSS, Neumann PFF, Rattner DWW, Dawson SLL, Ottensmeyer M, Jackson PGG, Bardsley RSS, Neumann PFF, Rattner DWW, Dawson SLL (2004) Computer-enhanced laparoscopic training system (CELTS): bridging the gap. Surg Endosc 18:782–789. doi: 10.1007/s00464-003-8932-0

140. Rahim MH (2009) 3D Computer Graphics: A Mathematical Introduction with OpenGL. Sch Sci Math 109:183–184. doi: 10.1111/j.1949-8594.2009.tb17955.x

141. Finney KC (2004) 3D game programming all in one.

142. Verth J, Bishop L (2004) Essential Mathematics for Games and Interactive Applications: A Programmers Guide.

143. Blandford E (1987) Computer graphics. Comput Educ 11:313–315. doi: 10.1016/0360-1315(87)90034-0

144. Fletch Dunn LP (2002) 3D Math Primer for Graphics And Game Development.

145. Waldron K, Schmiedeler J (2008) Kinematics. Springer Handb Robot 9–33. doi: 10.1007/978-3-540-30301-5_2

146. Diebel J (2006) Representing attitude: Euler angles, unit quaternions, and rotation vectors. Matrix 58:1–35. doi: 10.1093/jxb/erm298

147. Roberts LG (1966) Homogeneous matrix representation and manipulation of n-dimensional constructs.

148. Stolfi J (2014) Oriented Projective Geometry: A Framework for Geometric Computations.

149. Bloomenthal J, Rokne J (1968) Homogeneous Coordinates Homogeneous Coordinates for the Projective Plane. In Pract 11:1–27. doi: 10.1007/BF01900696

150. Bailey M, Cunningham S (2010) Introduction to computer graphics. ACM SIGGRAPH ASIA 2010 Courses - SA '10. doi: 10.1145/1900520.1900526

151. Laval PB (2003) Mathematics for Computer Graphics - Barycentric Coordinates. 1–9. doi: 10.1007/1-84628-283-7

152. Foley JD (1996) Computer Graphics: Principles and Practice. Addison-Wesley Professional

153. Tremblay C (2004) Mathematics for Game Developers.

154. Alter TD (1992) 3D Pose from 3 Corresponding Points under Weak-Perspective Projection. A.I. Memo 1378:

155. Lahanas V, Loukas C, Georgiou E (2015) A simple sensor calibration technique for estimating the 3D pose of endoscopic instruments. Surg Endosc. doi: 10.1007/s00464-015-4330-7

156. Loukas C, Lahanas V, Georgiou E (2013) An integrated approach to endoscopic instrument tracking for augmented reality applications in surgical simulation training. Int J Med Robot 9:1–28. doi: 10.1002/rcs.1485

157. Lahanas V, Loukas C, Nikiteas N, Dimitroulis D, Georgiou E (2011) Psychomotor skills assessment in laparoscopic surgery using augmented reality scenarios. In: 2011 17th Int. Conf. Digit. Signal Process. IEEE, pp 1–6

158. OpenGL - The Industry Standard for High Performance Graphics. https://www.opengl.org/. Accessed 17 Jul 2015

159. OGRE – Open Source 3D Graphics Engine. http://www.ogre3d.org/. Accessed 20 Jun 2015

160. BulletPhysics. http://bulletphysics.org. Accessed 20 Jun 2015

161. Clarke T a, Fryer JF, Wang X (1998) The Principal Point And CCD Cameras. Photogramm Rec 16:293–312. doi: 10.1111/0031-868X.00127

162. Trucco E, Verri A (1998) Introductory Techniques for 3-D Computer Vision.

163. Pressigout M, Marchand É (2004) Model-free augmented reality by virtual visual servoing. Proc - Int Conf Pattern Recognit 2:887–890. doi: 10.1109/ICPR.2004.1334401

164. Fiala M (2004) ARTag, An Improved Marker System Based on ARToolkit.

165. Lee SW, Kim DC, Kim DY, Han TD (2007) Tag detection algorithm for improving the instability problem of an augmented reality. Proc - ISMAR 2006 Fifth IEEE ACM Int Symp Mix Augment Real 257–258. doi: 10.1109/ISMAR.2006.297828

166. Wagner D, Schmalstieg D (2007) ARToolKitPlus for Pose Tracking on Mobile Devices. Proc 12th Comput Vis Winter Work CVWW07 139–146. doi: 10.1.1.157.1879

167. ARToolKit Library. http://www.hitl.washington.edu/artoolkit/. Accessed 25 Jun 2015

168. Augmented Reality SDK | ARToolKitPro. http://artoolkit.org/. Accessed 24 Jun 2015

169. Wloka MM, Anderson BG (1995) Resolving occlusion in augmented reality. In: Proc. 1995 Symp. Interact. 3D Graph. - SI3D '95. ACM Press, New York, New York, USA, pp 5–12

170. Breen D, Regli W, Peysakhov M (1994) 3D Transformations Window-to-Viewport Coordinates to Viewports Clipping to the Viewport 3D Transformations Representation of 3D Transformations 3D Transformations : 1–6.

171. Open Dynamics Engine. http://www.ode.org/. Accessed 20 Jun 2015

172. Allard, Cotin (2007) SOFA--an open source framework for medical simulation. … Reality, MMVR 15 125:13–8.

173. Marks S, Windsor J, Wünsche B (2008) Evaluation of game engines for simulated clinical training. Proc. …

174. Boeing A, Bräunl T (2007) Evaluation of real-time physics simulation systems. 1:281–288.

175. Hummel J, Wolff R, Stein T, Gerndt A, Kuhlen T (2012) An Evaluation of Open Source Physics Engines for Use in Virtual Reality Assembly Simulations. 346–357.

176. NVIDIA PhysX. https://developer.nvidia.com/physx-sdk. Accessed 20 Jun 2015

177. Havok. http://www.havok.com. Accessed 20 Jun 2015

178. Newton Dynamics. http://newtondynamics.com. Accessed 20 Jun 2015

179. Wittmeier S, Jäntsch M, Dalamagkidis K, Rickert M, Marques HG, Knoll A (2011) CALIPER: A universal robot simulation framework for tendon-driven robots. IEEE Int Conf Intell Robot Syst 1063–1068. doi: 10.1109/IROS.2011.6048111

180. Coumans E (2012) Bullet 2 . 80 Physics SDK Manual.

181. Zilles CB, Salisbury JK (1995) A constraint-based god-object method for haptic display. Proc 1995 IEEE/RSJ Int Conf Intell Robot Syst Hum Robot Interact Coop Robot 3:146–151. doi: 10.1109/IROS.1995.525876

182. LAP Mentor | Simbionix. http://simbionix.com/simulators/lap-mentor/. Accessed 29 Jul 2015

183. Epona » Medical. http://www.epona.com/site/medical/. Accessed 29 Jul 2015

184. Sakoe H, Chiba S (1978) Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans Acoust 26:43–49. doi: 10.1109/TASSP.1978.1163055

185. Padoy N, Blum T, Essa I, Feussner H, Berger M, Navab N, Lorraine L (2007) A Boosted Segmentation Method for Surgical. 10:102–109.

186. Bashir FI, Khokhar AA, Schonfeld D (2006) View-invariant motion trajectory-based activity classification and recognition. Multimed Syst 12:45–54. doi: 10.1007/s00530-006-0024-2

187. Leong JJH, Nicolaou M, Atallah L, Mylonas GP, Darzi AW, Yang G-Z (2007) HMM assessment of quality of movement trajectory in laparoscopic surgery. Comput Aided Surg 12:335–346. doi: 10.3109/10929080701730979

188. Rabiner LR (1989) A tutorial on hidden Markov models and selected applications in speech recognition. Proc IEEE 77:257–286. doi: 10.1109/5.18626

189. Rosen J, Solazzo M, Hannaford B, Sinanan M (2001) Objective laparoscopic skills assessments of surgical residents using Hidden Markov Models based on haptic information and tool/tissue interactions. Stud Health Technol Inform 81:417–23.

190. Schwarz G (1978) Estimating the Dimension of a Model. Ann Stat 6:461 – 464. doi: 10.2307/2958889

191. Mendez E, Kalkofen D, Schmalstieg D (2004) Interactive Context-Driven Visualization Tools for Augmented Reality. In: Third IEEE ACM Int. Symp. Mix. Augment. Real. (ISMAR 2006). Santa Barbara, USA , pp 209–218

192. Volonte F, Pugin F, Bucher P, Sugimoto M, Ratib O, Morel P (2011) Augmented reality and image overlay navigation with OsiriX in laparoscopic and robotic surgery: not only a matter of fashion. J Hepatobiliary Pancreat Sci 18:506–509. doi: 10.1007/s00534-011-0385-6

193. Pellen MG, Horgan LF, Barton JR, Attwood SE (2009) Construct validity of the ProMIS laparoscopic simulator. Surg Endosc 23:130–139. doi: 10.1007/s00464-008-0066-y

194. Martinez H, Mucha D, Kosmecki B, Kruger T (2008) Calibration method for electromagnetic tracked instruments in clinical applications. Int J Comput Assist Radiol Surg 3:111–113.

195. Pagador JB, Sanchez LF, Sanchez JA, Bustos P, Moreno J, Sanchez-Margallo FM, Sánchez LF, Sánchez J a, Bustos P, Moreno J, Sánchez-Margallo FM, Sanchez LF, Sanchez JA, Bustos P, Moreno J, Sanchez-Margallo FM, Sánchez LF, Sánchez J a, Bustos P, Moreno J, Sánchez-Margallo FM (2011) Augmented reality haptic (ARH): an approach of electromagnetic tracking in minimally invasive surgery. Int J Comput Assist Radiol Surg 6:257–63. doi: 10.1007/s11548-010-0501-0

196. Loukas C, Georgiou E (2011) Multivariate autoregressive modeling of hand kinematics for laparoscopic skills assessment of surgical trainees. IEEE Trans Biomed Eng 58:3289–3297. doi: 10.1109/TBME.2011.2167324

197. Dosis A, Aggarwal R, Bello F, Moorthy K, Munz Y, Gillies D, Darzi A (2005) Synchronized video and motion analysis for the assessment of procedures in the operating theater. Arch Surg 140:293–299. doi: 140/3/293 [pii]10.1001/archsurg.140.3.293

198. Wei G, Hirzinger G (1997) Real-Time Visual Servoing For Laparoscopic Surgery. Engineering

199. Tonet O, Thoranaghatte RU, Megali G, Dario P (2007) Tracking endoscopic instruments without a localizer: a shape-analysis-based approach. Comput Aided Surg 12:35–42. doi: 770808226 [pii]10.3109/10929080701210782

200. Doignon C, Nageotte F, Maurin B, Krupa A (2008) Pose estimation and feature tracking for robot assisted surgery with medical imaging. Springer, US

201. Haralick RM, Shapiro LG (1992) An integrated linear technique for pose estimation from different geometric features. Prentice Hall

202. Voros S, Long J -a., Cinquin P (2007) Automatic Detection of Instruments in Laparoscopic Images: A First Step Towards High-level Command of Robotic Endoscopic Holders. Int J Rob Res 26:1173–1190. doi: 10.1177/0278364907083395

203. Bradski G, Kaehler A (2008) Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly, Sebastopol, CA, USA

204. Rasmussen C, Hager G (1996) Tracking Objects by Color Alone. Yale University, New Heven, USA

205. Yang CC, Rodriguez JJ (1997) Efficient luminance and saturation processing techniques for color images. J Vis Commun Image Represent 3:263–277.

206. Doignon C, Graebling P, de Mathelin M (2005) Real-time segmentation of surgical instruments inside the abdominal cavityusing a joint hue saturation color feature. Real-Time Imaging 11:429–442.

207. Gonzalez RC, Woods RE (2002) Digital image processing , 2nd ed. Prentice Hall, Upper Saddle River, New Jersey, USA

208. Kalman RE (1960) A new approach to linear filtering and prediction problems. J Basic Eng 82:35–45.

209. Mills S, Pridmore TP, Hills M (2003) Tracking in a Hough space with the extended Kalman filter. In: Br. Mach. Vis. Conf. Norwich., UK, pp 173–182

210. Szeliski R (2010) Computer Vision: Algorithms and Applications. 870. doi: 10.1007/978-1-84882-935-0

211. Welsh G, Bishop G (1995) An introduction to the Kalman filter. University of North Carolina, Chapel Hill, NC, USA

212. Reiley CE, Lin HC, Yuh DD, Hager GD (2011) Review of methods for objective surgical skill evaluation. Surg Endosc 25:356–366. doi: 10.1007/s00464-010-1190-z

213. Megali G, Sinigaglia S, Tonet O, Dario P (2006) Modelling and Evaluation of Surgical Performance Using Hidden Markov Models. IEEE Trans Biomed Eng 53:1911–1919. doi: 10.1109/TBME.2006.881784

214. English J, Chang C-Y, Tardella N, Hu J (2005) A vision-based surgical tool tracking approach for untethered surgery simulation and training. Stud Health Technol Inform 111:126–132.

215. Gor M, McCloy R, Stone R, Smith A (2003) Virtual reality laparoscopic simulator for assessment in gynaecology. BJOG 110:181–7.

216. Pagador JB, Sánchez LF, Sánchez J a, Bustos P, Moreno J, Sánchez-Margallo FM (2011) Augmented reality haptic (ARH): an approach of electromagnetic tracking in minimally invasive surgery. Int J Comput Assist Radiol Surg 6:257–63. doi: 10.1007/s11548-010-0501-0

217. Gonzalez D, Carnahan H, Praamsma M, Dubrowski A (2007) Control of laparoscopic instrument motion in an inanimate bench model: implications for the training and the evaluation of technical skills. Appl Ergon 38:123–32. doi: 10.1016/j.apergo.2006.03.011

218. Birkmeyer JD, Finks JF, O'Reilly A, Oerline M, Carlin AM, Nunn AR, Dimick J, Banerjee M, Birkmeyer NJO (2013) Surgical skill and complication rates after bariatric surgery. N Engl J Med 369:1434–1442.

219. Feng C, Haniffa H, Rozenblit J, Peng J (2006) SURGICAL TRAINING AND PERFORMANCE ASSESSMENT USING A.

220. Lakshmi B, Dhar a. S (2010) CORDIC Architectures: A Survey. VLSI Des 2010:1–19. doi: 10.1155/2010/794891

221. Maithel S, Sierra R, Korndorffer J, Neumann P, Dawson S, Callery M, Jones D, Scott D (2006) Construct and face validity of MIST-VR, Endotower, and CELTS. Surg Endosc Other Interv Tech 20:104–112.

222. Panait L, Akkary E, Bell RL, Roberts KE, Dudrick SJ, Duffy AJ (2009) The role of haptic feedback in laparoscopic simulation training. J Surg Res 156:312–316.

223. McDougall EM, Corica FA, Boker JR, Sala LG, Stoliar G, Borin JF, Chu FT, Clayman R V. (2006) Construct Validity Testing of a Laparoscopic Surgical Simulator. J Am Coll Surg 202:779–787. doi: 10.1016/j.jamcollsurg.2006.01.004