



Technische Universität München
Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Medientechnik

Virtual Manipulations with Force Feedback in Complex Interaction Scenarios

Mikel Sagardia Erasun

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr.-Ing. Sami Haddadin
Prüfer der Dissertation: 1. Prof. Dr.-Ing. Eckehard Steinbach
2. Prof. Dr. Gabriel Zachmann

Die Dissertation wurde am 15.11.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 10.10.2019 angenommen.

Eskerrik asko itxarroteagatik, Ana.

Abstract

Haptic feedback applied to object manipulations in Virtual Reality (VR) extends user interactivity to the physical domain. With the haptic modality, users not only *see* or *hear* as they manipulate, but they are also able to *feel contacts through the sense of touch*, which leads to increased user performance. Multimodal environments comprising haptic feedback can be used in a plethora of applications, such as to verify mechanical designs or to train assembly technicians or surgeons. These realtime virtual simulations help to reduce the need of expensive real environments and contribute to accelerating and improving the product or knowledge generation process.

However, *haptic rendering* and *display* are still active fields of research, in part, due to the challenging technical requirements that they face when considering even practical real-life scenarios: collision forces that belong to a wide impedance range need to be computed and displayed for any geometry every 1 ms in order to guarantee system stability.

This thesis studies *haptic rendering algorithms* that wholly solve those requirements and that can be applied in realistic VR simulations. New methods for achieving fast and accurate *collision and force computation* in complex scenarios are presented. These methods are validated in the context of several *virtual simulations and robotics applications* and, additionally, evaluated in *user studies* that reveal general insights on multimodal haptic interaction.

After identifying the major needs in the state-of-the-art of haptic rendering, the contributions are presented in four parts. First, a new unified method for computing collision, distance, and penalty forces between arbitrarily complex geometries at 1 kHz is introduced. The approach is based on the principles of the Voxelmap Pointshell (VPS) algorithm, but it re-defines the used queries and data structures, which consist of multi-resolution signed distance fields and point-sphere trees.

Second, a novel constraint-based method for contact motion computation and force

rendering is presented. The approach runs robustly at $5\ \mu\text{s}$, it is easy to implement, and it can be applied on top of many penalty-based approaches, increasing their contact stiffness, and considerably reducing object overlap even with non-watertight thin shells.

Third, the integration of the aforementioned algorithms into virtual environments, physics simulators, and robotics frameworks is shown; that accounts for the suitability of the methods for a great variety applications. Special focus is set on an immersive virtual car assembly platform which deals with large multibody scenarios. The presented assembly platform synthesizes the major components characteristic of complex virtual reality setups.

And, finally, the introduced methods are evaluated in two comprehensive user studies. The first shows how haptic rendering must be parametrized for optimum human performance, considering also different haptic devices. The second explains the differences between real and virtual manipulations, dissecting the effects of multimodal rendering methods and display devices, and pointing out guidelines for improved haptic interaction.

Kurzfassung

Das haptische Feedback erweitert die Benutzerinteraktivität auf die physische Domäne, unter anderem in Anwendungen der virtuellen Realität (VR), an denen Objekte manipuliert werden. Mit der haptischen Modalität können Benutzer während der Manipulationsaufgabe nicht nur sehen oder hören, sondern sie können auch Kontakte durch den Tastsinn fühlen; dies führt zu einer erhöhten Interaktionsperformanz. Multimodale Umgebungen mit haptischem Feedback können in einer Vielzahl von Anwendungen eingesetzt werden, beispielsweise zur Überprüfung mechanischer Konstruktionen oder zum Trainieren von Montagetechnikern oder Chirurgen. Diese Art von virtuellen Echtzeitsimulationen reduziert den Bedarf an teuren realen Prototypen und trägt zur Beschleunigung und Verbesserung des Produkt- oder Wissenserzeugungsprozesses bei.

Das haptische Rendern und seine Wiedergabe ist jedoch immer noch ein aktives Forschungsfeld, zum Teil aufgrund der anspruchsvollen technischen Anforderungen, die auch in einfachen Szenarien gelten: Kollisionskräfte müssen über ein weites Impedanzspektrum für beliebige Geometrien im Kilohertz-Takt berechnet und an den Benutzer übertragen werden, um die Systemstabilität zu gewährleisten.

In dieser Arbeit werden haptische Algorithmen für das Kontaktrendering untersucht, die diese Anforderungen vollständig erfüllen und in realistischen VR-Simulationen angewendet werden können. Es werden neue Methoden vorgestellt, um eine schnelle und genaue Kollisions- und Kraftberechnung in komplexen Szenarien zu erreichen. Diese Methoden werden in mehreren virtuellen Simulationen und Robotikanwendungen validiert und in Benutzerstudien evaluiert, die allgemeine Einblicke in die multimodale haptische Interaktion liefern.

Nach der Identifizierung der wichtigsten Erfordernisse im Stand der Technik des haptischen Renderns, werden die Beiträge in vier Teilen berichtet. Zunächst wird ein neues vereinheitlichtes Verfahren zur Berechnung von Kollisionen, Abständen und Kontaktkräften zwischen beliebig komplexen Geometrien bei 1 kHz eingeführt. Der Ansatz basiert

auf den Prinzipien des Voxelmap-Pointshell-Algorithmus (VPS), definiert jedoch sowohl die verwendeten Methoden neu, als auch Datenstrukturen, die aus Distanzfeldern und Punkt-Kugel-Hierarchien mit mehreren Auflösungen bestehen.

Zweitens wird ein neuartiges Verfahren zur Berechnung der Kontaktkräfte vorgestellt, das auch die eingeschränkte Bewegung bei Kontakt simulieren kann. Der Ansatz gehört zu den sogenannten *constraint-based* Methoden, funktioniert robust mit einer Rechenzeit von $5 \mu s$, ist einfach zu implementieren und kann in Kombination mit vielen sogenannten *penalty-based* Methoden verwendet werden. Er ermöglicht eine höhere Kontaktsteifigkeit und erheblich reduzierte Objektüberlappungen, auch bei nicht wasserdichten, dünnen Objekten.

Drittens wird die Integration der vorgenannten Algorithmen in virtuelle Umgebungen, Physiksimulatoren und Robotikanwendungen gezeigt; alle diesen Beispiele bestätigen die Eignung der vorgestellten Methoden für eine Vielzahl von Anwendungen. Besonderer Fokus liegt auf einem immersiven virtuellen Fahrzeugmontagedemonstrator, der sich mit großen Mehrkörperszenarien befasst. Der vorgestellte Demonstrator stellt die Hauptkomponenten dar, die für komplexe virtuelle-Realität-Umgebungen charakteristisch sind.

Schließlich werden die eingeführten Methoden in zwei umfassenden Benutzerstudien evaluiert. Die Erste zeigt, wie das haptische Rendern für eine optimale Mensch-Maschine Interaktion parametrisiert werden muss, wobei auch verschiedene haptische Geräte berücksichtigt werden. In der Zweiten werden die Unterschiede zwischen realen und virtuellen Manipulationen untersucht. Dabei werden die Auswirkungen multimodaler Rendering-Methoden und haptischer Eingabegeräte analysiert und Richtlinien für eine verbesserte haptische Interaktion synthetisiert.

Acknowledgments

This thesis originated during my work at the Institute of Robotics and Mechatronics of the German Aerospace Center (DLR). The Institute is an incredible hub of brilliant engineers that create game-changing robotic systems. That would be impossible without the support of the former and current directors, Prof. Gerd Hirzinger and Prof. Alin Albu-Schäffer, to whom I am very grateful, not only for nourishing such a fruitful environment, but also for their trust in me along all these years. I am also very grateful to my former head of department Christoph Borst and my former team leader Carsten Preusche. They believed in me and gave me the freedom to pursue my research interests at all times.

I would also like to thank Prof. Eckehard Steinbach, Prof. Gabriel Zachmann, and Prof. Jorge Juan Gil, for their academic guidance. Prof. Steinbach, my supervisor at the Technical University of Munich and first thesis examiner, provided me with invaluable comments and corrections of this work. Prof. Zachmann, from the Technical University of Bremen, has always been keen and ready to share his world-class knowledge and experience in collision detection. Prof. Gil was my master's thesis supervisor at Tecnum; he introduced me to the topic of haptics and put me in contact with the DLR.

I am also very grateful to all my colleagues at the DLR for their helpfulness; I feel blessed of having had the chance to be part of such a great team, so full of talent and kindness. My greatest thanks go to all the colleagues that belong or have belonged to the *Telepresence and Virtual Reality Group*, especially to Thomas Hulin, Philipp Kremer, and Katharina Hertkorn. This thesis could not have been possible without the wise mentoring, help, and effort of Thomas or without the expertise of Philipp and Katharina. Similarly, all other inestimable group members and co-workers that helped me with my work are: Simon Schätzle, Bernhard Weber, Bendikt Pleintinger, Michael Panzirsch, Ribin Balachandran, Jordi Artigas, Ingo Kossyk, Stefan von Dombrowski, Florian Schmidt, Robert Burger, Neal Lii, and Cornelia Riecke. Thank you also to my

colleagues Korbinian Nottensteiner and Theodoros Stouraitis, who intensively used the results of my research and pushed me to improve my work.

Likewise, I would like to thank to all the past and present members of the project *Virtual Reality for On-Orbit Servicing* from the facility *Simulation and Software Technology* of the DLR in Braunschweig: Robin Wolff, Johannes Hummel, Janki Dodiya, Andreas Gerndt, Sebastian Utzig, Andreas Bernstein, and Simon Schneegans.

During my research endeavors, I have participated in several cooperations; to all my peers and project partners, thank you for all the fruitful exchanges and for showing me the way for professional growth. Particularly, I will always kindly remember Ralf Rabätje, former head of the Volkswagen VR Lab, Jérôme Perret, from Haption GmbH, and René Weller, from the Technical University of Bremen.

Finally, I would like to deeply thank to my parents *aita eta ama*, my sisters Aintzane *eta* Itziar, my dear wife Ana, and all my friends in Munich, for their love, encouragement, and understanding. Ana, my greatest and most profound thank you to you for your love and patience in the final phase of the thesis.

Contents

Abstract	v
Kurzfassung	vii
Acknowledgments	ix
Contents	xi
1 Introduction	1
1.1 Problem Definition and Motivation	2
1.2 Overview and Key Contributions	5
2 Background	7
2.1 Human Haptic Sensory System	7
2.1.1 Physiological and Neurological Aspects	8
2.1.2 Psychophysical and Psychological Aspects	10
2.2 Immersive Virtual Environments with Haptic Feedback	13
2.2.1 Human Factors	13
2.2.2 Multimodal Rendering	14
2.2.3 Physics Engines: Motion Computation	15
2.2.4 Interaction Devices and Techniques	16
2.2.5 Kinesthetic Haptic Devices and Control Issues	18
2.2.6 Telerobotics	23
2.2.7 Haptic Communications	25
2.2.8 Applications	25
2.3 Collision Computation and Force Rendering	26
2.3.1 Object Representations	31

2.3.2	Collision Computation	36
2.3.2.1	Collision Output	36
2.3.2.2	Basic Methods	38
2.3.2.3	Discrete versus Continuous Collision Detection	43
2.3.2.4	Multibody Scenarios (Techniques for the Broadphase)	44
2.3.2.5	Acceleration Strategies	45
2.3.3	Collision Response: Force Rendering	50
2.3.3.1	Overview of Output in Collision Response	50
2.3.3.2	Three Force Rendering Paradigms	51
2.3.3.3	Direct versus Indirect Force Display: Virtual Coupling	55
2.3.3.4	Degrees-of-Freedom (DoF): Three (Forces) versus Six (Forces and Torques)	57
2.3.3.5	Enhancements for Fidelity and Realism: Friction, Shading, and Transients	58
2.3.3.6	Deformation	59
2.4	Summary, Conclusions, and Perspectives	62
3	Collision Computation	65
3.1	Introduction	65
3.1.1	Related Work	66
3.1.2	Contributions	67
3.2	Data Structures	68
3.2.1	Generation of Basic Primitives	69
3.2.1.1	Voxelized Structures (Voxelmaps)	69
3.2.1.2	Point Clouds (Pointshells)	72
3.2.2	Properties and Limitations	75
3.2.2.1	Properties	76
3.2.2.2	Limitations	78
3.2.3	Enhanced Voxelmaps: Signed Distance Fields	79
3.2.3.1	Generation of the Structures in the Enhanced Voxelmap	80
3.2.3.2	The Signed Distance Voxelmap Function $V(P)$	81
3.2.3.3	Comparison of the Signed Distance Function $V(P)$ Calls	84
3.2.4	Enhanced Pointshells: Point-Sphere Trees	85
3.2.4.1	Point Qualities	88
3.2.4.2	Hierarchy Generation	89
3.3	Proximity and Collision Queries with Complex Objects	93
3.3.1	General Hierarchical Traverse	94
3.3.1.1	Input Data	96

3.3.1.2	Output Data	96
3.3.1.3	Collision Computation ($p_c = 0, \eta_c = 1, q_c = 0$)	97
3.3.1.4	Distance Computation ($p_c \geq 0$)	99
3.3.1.5	Segmented Hierarchical Traverse (Clustered \mathcal{M})	100
3.3.2	Time Critical Level-of-Detail Traverse ($\eta_c < 1, q_c > 0$)	100
3.3.2.1	Maximum Allowed Computational load (η_c)	101
3.3.2.2	Minimum Required Quality (q_c)	102
3.3.2.3	Spatio-Temporal Coherence	103
3.3.2.4	Discussion	103
3.4	Experiments and Results	104
3.4.1	Discussion of Scenario 1: Sphere and Cube	105
3.4.2	Discussion of Scenario 2: Stanford Bunny and Utah Teapot	109
3.5	Summary, Conclusions, and Perspectives	112
4	Force Rendering	115
4.1	Introduction	116
4.1.1	Related Work	116
4.1.2	Contributions	118
4.2	God Object Heuristic	118
4.2.1	Penalty-Based Contact Computation (#1)	122
4.2.2	Correction of the Previous Proxy Frame (#2)	123
4.2.2.1	Computation of the Correction Rotation (θ)	124
4.2.2.2	Computation of the Correction Translation-Rotation Distribution Factor (λ)	127
4.2.2.3	Assembly of the Final Correction Step	128
4.2.3	Computation of the Unconstrained Motion (#3)	129
4.2.4	Computation of the Constrained Motion (#4)	130
4.2.5	Filtering of the Proxy Pose (#5)	131
4.2.6	Coupling Forces Applied to the Haptic Device (#6, #7, #8)	132
4.2.7	Six-DoF Friction (#4)	132
4.3	Theoretical Discussion of Methods	134
4.4	Experiments and Results	138
4.5	Summary, Conclusions, and Perspectives	142
5	Applications	143
5.1	Introduction	144
5.1.1	Related Work	144
5.1.1.1	Virtual Assembly (VA) Systems	144

5.1.1.2	Physics Simulators	146
5.1.2	Contributions	146
5.2	Virtual Assembly with Haptic Feedback	148
5.2.1	Simulation Framework	148
5.2.1.1	Multibody Collision Computation Module	148
5.2.1.2	Game Control	153
5.2.1.3	Completing the Jigsaw Puzzle: Communication, Track- ing, and Visualization	153
5.2.2	Interaction Devices and Techniques	154
5.2.2.1	The Bimanual Haptic Device HUG	154
5.2.2.2	The Vibrotactile Arm Band VibroTac	154
5.2.2.3	Workspace Navigation	156
5.2.2.4	Collaboration with Additional Haptic Interfaces	157
5.2.3	Exemplary Scenario: Car Assembly Sequence	158
5.2.3.1	Performance Results	160
5.3	Integration into the Physics Engine Bullet	161
5.3.1	Data Structures and Workflow in Bullet	162
5.3.2	Integration Interfaces	163
5.3.3	Experiments and Results	164
5.3.3.1	Tests with a Bouncing Ball	164
5.3.3.2	Tests with the Stanford Bunny	165
5.4	Other Application Environments	168
5.4.1	A Virtual Reality Platform for On-Orbit Servicing Simulations	168
5.4.2	Ultrapiano: Playing a Virtual Piano with Ultrasound-Imaging	170
5.4.3	Robotic Autonomous Assemblies Using Virtual Models	170
5.4.4	Realtime Collision Avoidance for Mechanisms with Complex Ge- ometries	171
5.4.5	Shared Grasping: Semi-Autonomous Robotic Grasping Using Vir- tual Models	172
5.5	Summary, Conclusions, and Perspectives	173
6	Evaluation of Methods	175
6.1	Introduction	176
6.1.1	Related Work	177
6.1.2	Contributions	179
6.2	Study 1: Evaluating Haptic Rendering Methods	180
6.2.1	Experimental Design and Implementation	181
6.2.1.1	Tested Scenario: Tasks and Exercises	181

6.2.1.2	Apparatus and Varied Factors	184
6.2.1.3	Sample, Procedure, and Collected Data	186
6.2.2	Results and Discussion	189
6.2.2.1	Haptic Devices: Ergonomy and Workload	189
6.2.2.2	Exercises: Performance and Contact Perception	190
6.2.3	Study 1: Summary of Lessons Learned and Discussion	199
6.3	Study 2: Comparing Real and Virtual Manipulations	201
6.3.1	Experimental Design and Implementation	201
6.3.1.1	Synthetic Haptic Feedback	201
6.3.1.2	Tested Scenario: Tasks and Exercises	202
6.3.1.3	Apparatus and Varied Factors	203
6.3.1.4	Secondary Task and Auditory Privation	206
6.3.1.5	Sample, Procedure, and Collected Data	207
6.3.2	Results and Discussion	208
6.3.2.1	Regular Exercises	208
6.3.2.2	Secondary Task and Auditory Privation	214
6.3.3	Study 2: Summary of Lessons Learned and Discussion	217
6.4	Summary, Conclusions, and Perspectives	219
7	Epilogue	223
A	Used Haptic Devices	227
A.1	The HUG	227
A.2	The Sigma.7	229
B	Implementation and Performance Issues	231
C	Virtual Reality and Haptics: Evolution of Relevance	235
D	Publications by the Author	239
	Bibliography	243
	List of Symbols	271
	List of Figures	283
	List of Tables	285
	List of Algorithms	287

Chapter 1

Introduction

Witnessing the renaissance that Virtual Reality (VR) is experiencing in the past years, it is clear that both *visual* and *auditory feedback* have significantly evolved; this holds for the *rendering methods* and *display devices* used to respectively *compute* and *deliver* these two feedback modalities with the purpose of creating plausible virtual worlds. In contrast, *haptic feedback* seems not to be as mature – yet many common physical interactions in everyday life would not be possible without this sensory modality. That is in part due to the unique and challenging nature of haptics, which comprises both the sense of *touch* and *proprioception*, i. e., the ability to process and comprehend kinesthetic or contact force phenomena occurring in muscles and tendons. Indeed, the haptic sense is distributed on the whole body, it requires action and physical interaction, and it is very sensible to undesired artifacts. Additionally, haptic feedback needs to be generated and provided at least at 1 kHz due to human sensibility and system stability reasons.

This thesis addresses kinesthetic haptic feedback in virtual reality environments. The main focus lies on **kinesthetic haptic rendering**, which consists in creating *synthetic contact force signals* displayed to the user via a *haptic device* or *interface*. For that, efficient *collision and force computation* algorithms have been conceived and implemented, taking into account the human haptic sensory system and the technical requirements around it. These algorithms have been successfully integrated in *virtual assembly and robotic applications*, and their effectiveness has been demonstrated with *user studies* that consider all virtual reality system components in a holistic manner.

Haptics, and, in particular, kinesthetic feedback, can revolutionize interactive virtual simulations and, in consequence, many industries and domains in which object manipulation is necessary. For instance, thanks to it designers and engineers are able to *verify*

non-expensively and instantaneously new virtual prototypes, or assembly technicians and medical doctors can *train* with specific virtual models; even *machines* can boost their *physical interaction abilities* with virtual haptic models that help them understand video images and sensed forces during the interaction with the physical world. This thesis tries to contribute with a step forward in these directions.

1.1 Problem Definition and Motivation

Figure 1.1 shows the components of an interactive virtual reality setup which provides *haptic feedback* in addition to the presentation of video and audio. In such an environment, the user moves a haptic device in order to dynamically and intuitively change the configuration of virtual objects. The haptic device should ideally not be felt by the user during manipulations free of contact in the virtual environment, but it should reflect realistic resistance forces as soon as the moved virtual object collides. The inter-related components necessary for such a setup can be classified in these groups:

- (a) *Virtual Models: geometries* (usually triangle meshes) of the objects to be manipulated in the scenario are transformed into framework-specific *data structures* which contain the necessary information for the simulation (e.g., physical or topological properties).
- (b) *Rendering Loops*: often parallel *haptic*, *graphic*, *audio*, and *motion* (i.e., physics engines) loops produce, respectively, contact forces, images, sounds and object movement displayed to the user; each loop runs at different temporal update rates, uses its corresponding data structures and shares information with the others.
- (c) *Simulation of Scenario and Application*: the task performed by the user (e.g., a virtual assembly) is usually controlled by a *system logic module* (e.g., a finite state machine) that ensures a coherent workflow; *interaction techniques* are defined in the simulation environment, in close relation to the available interaction devices, allowing actions such as navigation or selection of objects.
- (d) *Information Transport*: it is fundamental to guarantee an appropriate *communication* channel for each feedback modality with the corresponding transmission rate; in particular, haptic feedback is bilateral (displacements are read from the user, forces are sent) and requires a challenging update rate of 1 kHz, therefore, compression techniques can be applied on signals across channels with lower qualities; *haptic control* is also essential to guarantee the stability of the haptic manipulator and avoid low fidelity or even dangerous interactions.

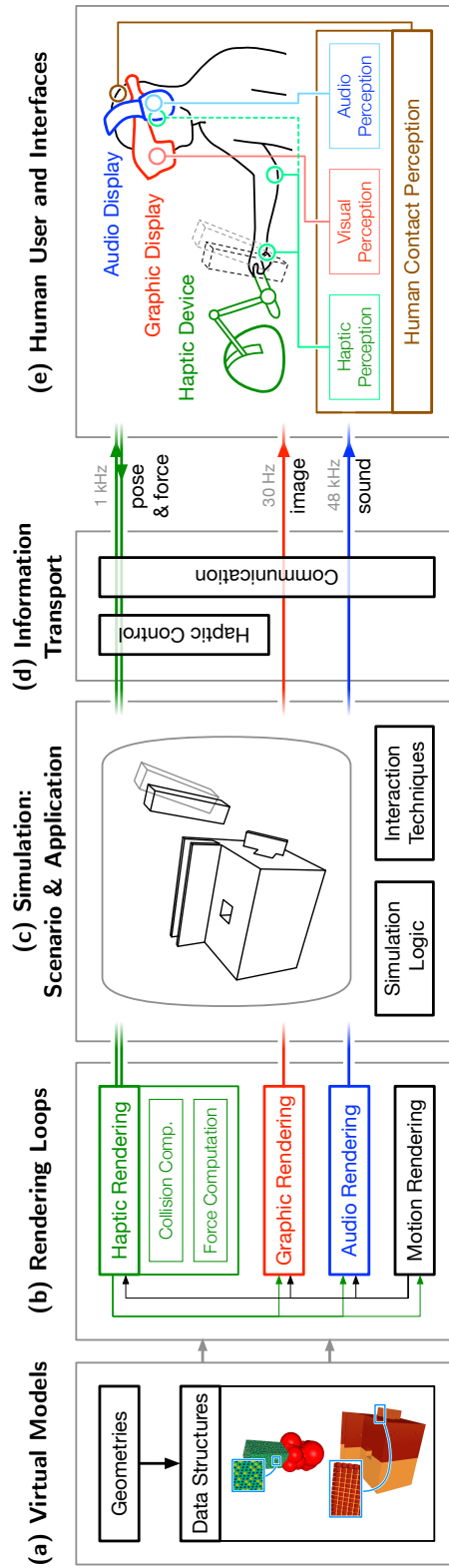


Figure 1.1: A Virtual Reality (VR) simulation with haptic feedback provides the user with haptic contact information (usually in addition to graphic and/or audio feedback) during the manipulations performed in an interactive environment. The user moves a virtual object with the hand coupled to a haptic device; in the absence of collision, the user can move freely, when the object intersects with another one in the virtual scenario, collision forces are displayed through the actuators of the haptic device. Such simulations are called *multimodal*, because they target multiple senses with dedicated *synthetic feedback modalities*, aiming at higher levels of *immersion*, *presence*, and *user performance*.

- (e) Human User and Interfaces: the *haptic device*, *graphic display* (e.g., a HMD or head-mounted display), and *audio display* (e.g., headphones) present, respectively, haptic, graphic, and audio feedback to the user; the *human perception* of each modality is integrated in the cortex to generate contact and, ultimately, (virtual) reality perceptions. Therefore, the human user interacts with all simulation loops; in particular, he/she closes the haptic loop, being mechanically and energetically coupled to the physical events in the scenario.

This thesis presents a virtual car assembly platform for which those listed items have been implemented and integrated. More specifically, the main contribution lies on the investigation, development, and evaluation of new *kinesthetic haptic rendering methods* able to comply with the components and requirements of such a complex multimodal and interactive virtual reality system. The presented or similar platforms can be used with *training* and *verification* purposes, shortening and making less expensive product design cycles. Additionally, the introduced collision and force rendering algorithms are a core element for many robotic and virtual reality applications performing *realtime simulation* and *prediction*.

Preliminary Studies

Prior to the contributions presented in this work, preliminary research was carried out by the author of this thesis. First, methods for generating the basic data structures necessary for the Voxelman-Pointshell (VPS) haptic rendering algorithm [MPT99] were developed [Sag08]. These methods were integrated in an early re-implementation of the VPS, which was evaluated in two studies; a detailed description of their content is out of scope, but the insights of both works are briefly outlined, since they motivate the paths selected for the current thesis.

In the first study, a *benchmark for force rendering performance and quality* was developed [SHPH09], [WSM⁺10]. Using well defined synthetic scenarios, the suit computes the deviation of force magnitude and direction values from expected analytical ones. The experiments pointed out to the need of higher resolutions in the data structures, particularly in colliding areas. This is in conflict with the requirement for performance; however, the trade-off is successfully addressed in this thesis.

In the second study, *force feedback was compared to visual and vibrotactile feedback* in a user study in which participants had to perform virtual peg-in-hole exercises [SWH⁺12], [WSHP13]. Altogether, the results showed that the delivered force feedback was superior for collision display in terms of movement precision, mental workload, and spatial orientation. The importance of correct visual display of contacts was also verified in the

study. All in all, these conclusions motivate the need of kinesthetic haptic rendering algorithms able to simulate collisions realistically.

1.2 Overview and Key Contributions

This thesis is divided in six additional chapters to this one and four appendices; in the following, the contributions and contents of each of them are summarized:

Chapter 2 reports *related works* covering three basic aspects: (i) human haptic psychophysics, (ii) architecture and general human factor details concerning virtual reality systems with haptic feedback, and (iii) technical methods for collision and force computation. In particular, the thorough review provided for this last point presents an up-to-date and complete snapshot of the *state-of-the-art on kinesthetic haptic rendering* and its evolution during the last decades.

Chapter 3 presents a unified *collision, proximity, and penalty force computation method* able to handle arbitrarily complex rigid geometries at an update rate of 1 kHz. The approach is based on the principles of the Voxelman-Pointshell (VPS) haptic rendering algorithm [MPT99], but it re-defines and optimizes data structures and online query methods for time-budgeted applications requiring high object resolutions in collision areas. The researched and developed methods are tested in replicable experiments. The resulting engine can be applied for haptic rendering in virtual environments or for robotic applications that need collision and proximity information between complex rigid geometries.

Chapter 4 introduces a *constraint-based force rendering method* able to optimize overlapping object configurations to surface contact states and to compute stiff and realistic contact forces. The approach can be used on top of a penalty-based haptic rendering algorithm (the one from Chapter 3 is employed). As shown in the experiments, and in contrast to similar algorithms, the method runs robustly at ~ 20 kHz, without dedicated threads.

Chapter 5 reports the integration of the previous haptic rendering methods into *virtual reality and robotic applications*. In it, the presented *virtual car assembly platform* represents the most complete framework; it considers all components necessary for such immersive systems and provides solutions for multibody haptic environments. Additionally, the integration of the presented collision computation methods into the *physics engine Bullet* [Cou03] is covered, and five further *simulation and robotic applications* using the conceived algorithms are described.

Chapter 6 *evaluates and analyzes the researched algorithms* in two user studies. In them, the multimodal nature of contact perception is considered and general insights for virtual manipulations with haptic feedback are provided. Whereas the first study deals with the optimum parametrization of haptic systems for best performance and perception indicators, the second looks in detail into the causes that originate the difference between virtual and real manipulations with haptic feedback.

Chapter 7 *concludes with the most important insights* derived from the previous chapters and points out to future perspectives.

Appendix A describes in detail the *haptic devices* HUG [HHK⁺11] and Sigma.7 [THH⁺11] used in the experiments and evaluations presented in this thesis.

Appendix B compiles relevant *implementation issues* related to the data structures presented in Chapter 3.

Appendix C reports the *evolution of the importance of different terms* related to collision and force computation in virtual reality.

Appendix D provides with a *list of publications by the author* and their weight in the contributions of the thesis.

Chapter 2

Background and Related Work

This chapter presents the background and state-of-the-art related to the contributions of this thesis. It is composed of three main inter-related sections that, nonetheless, can be read separately with the desired order. Section 2.1 introduces the human haptic sensory system, building up from physiological aspects to psychological considerations. Section 2.2 discusses technical issues and human factors that affect any virtual reality system that conveys haptic feedback; it is basically a deeper description of the elements presented in Figure 1.1 from Chapter 1. Finally, Section 2.3 reviews the most important technical works dealing with haptic rendering, i. e., collision detection and response for haptic interaction.

2.1 Human Haptic Sensory System

The *haptic* sense or the sense of touch is the essential feedback channel or mode with which humans perceive contacts during interactions with the environment. Two subsystems can be identified for it [LK09]: (i) the *cutaneous* or *tactile* sense, necessary for the perception of surface properties, and (ii) the *kinesthetic* or *proprioceptive* sense, which leads to the perception of weight and movement properties.

The latter proprioceptive sense is strongly related to the *vestibular* system, which produces body motion signals that ensure, ultimately, balance and spatial orientation [LJ00]. Furthermore, haptic signals are integrated together with *visual* and *auditory* cues in the brain in order to form correct contact percepts necessary for high performance interaction. The rest of this section introduces the fundamental physiological (Section 2.1.1) and psychological (Section 2.1.2) concepts necessary to understand how humans process

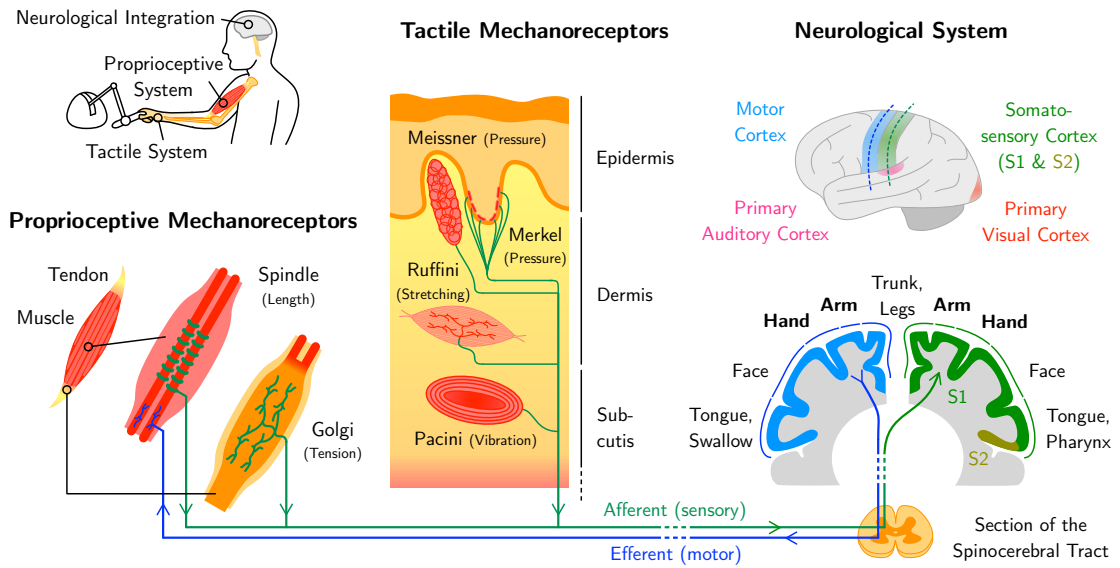


Figure 2.1: Human haptic sensory system. Proprioceptive mechanoreceptors (re-drawn and adapted from [PG12]) are located in muscles and tendons, whereas the tactile mechanoreceptors (re-drawn and adapted from [LK09]) can be found in the skin. Most important perception functions of each mechanoreceptor are in parentheses. Efferent or motor neurons (blue) command movements for exploration to the muscles from the motor cortex, while afferent or sensory neurons (green) carry sensory stimuli encoded as electrical impulses from the mechanoreceptors to the somatosensory cortex. Primary (S1) and secondary (S2) somatosensory cortices are distinguished.

those contact percepts.

2.1.1 Physiological and Neurological Aspects

Figure 2.1 illustrates the most important elements of the human haptic sensory system. Manipulation of objects making use of haptic sense occurs thanks to mechanoreceptors and motor peripherals (i. e., muscles) connected through *afferent* (sensory) and *efferent* (motor) neurons to the somatosensory and motor cortices, respectively. These afferent fibers end up in two types of mechanoreceptors: *tactile* or *proprioceptive**

Tactile mechanoreceptors are located in the skin and they can be classified according to (i) the size of their reception field (small/large: I/II) and (ii) the adaptation rate (slow/fast: SA/FA) [HB08]. Receptors with small receptive fields (I) have sharp borders and rapidly increasing activation thresholds, whereas receptors with large fields (II) have less defined perimeters and gradually increasing thresholds. On the other

*It is worth to mention that another class of haptic afferents named *nociceptors* is in charge of detecting pain and thermal changes [HB08]. These receptors consist basically of free nerve endings and are located in skin and connective tissue. Since they are not as significant as the previous during regular manipulations, further description is omitted.

hand, the adaption property is related to the response during indentation stimuli: slow mechanoreceptors (SA) respond to both dynamic (i. e., advancing into and retracing from skin) and static indentation phases (i. e., steady maintained indentation), whereas fast mechanoreceptors (FA) signal only under dynamic stimuli [GW08].

The mechanoreceptors in the non-hairy hand skin have been the most studied ones, as expected, as the hand is the most usual manipulation tool for humans; however, the receptors in the whole body skin are similar [GW08]. Altogether, as shown in Figure 2.1, the four tactile mechanoreceptors with their functions according to [HB08] and [LK09] are:

- *Merkel disks* [slow adapting, small field: SA I; $\sim 80 - 120$ nm diameter]: pressure, very-low-frequency vibrations (< 5 Hz approx.), coarse texture, patterns, stable precision grasp and manipulation
- *Meissner corpuscles* [fast adapting, small field: FA I; $\sim 100 - 150$ μm length]: pressure, low-frequency vibration ($5 - 40$ Hz approx.), stable precision grasp and manipulation
- *Ruffini corpuscles* [slow adapting, large field: SA II; $\sim 200 - 300$ μm length]: stretching, direction of object motion and force, stable precision grasp and manipulation, finger position
- *Pacini corpuscles* [fast adapting, large field: FA II; ~ 2 mm length]: high-frequency vibration ($40 - 400$ Hz approx.), fine texture, stable precision grasp and manipulation

Proprioceptive mechanoreceptors are located in muscles, tendons, and joints, and have been less studied than their tactile counterparts. Two types are distinguished, with the following functions [HB08]:

- *Muscle spindles* [$\sim 1 - 2$ mm length]: measurement of the muscle length changes
- *Golgi tendon organs* [up to 1.6 mm length]: measurement of the tension exerted by muscles

It is worth to mention that the aforementioned Ruffini and Pacini tactile mechanoreceptors also appear in joints [HB08], where they contribute to proprioceptive sensation. It is still not well understood how data from proprioceptive and tactile afferents is integrated [GW08] to obtain information of the environment objects and their manipulation.

Receptor potentials are converted into action potentials and propagated in usually three relaying or synaptic connection steps along the myelinated axons of the afferent

fibers until reaching the **somatosensory cortex**: the dorsal root ganglia, the spinal cord, and the thalamus [HY08]. Interestingly, responses of single afferent neurons are ambiguous, whereas responses of populations have more clear meanings [GW08]. Indeed, changes in stimulated populations can be mapped to the evolution of perceived shape fragments during exploration.

The processing of afferent signals in the somatosensory cortex occurs in well-defined areas and stages [HY08]. The *primary* somatosensory cortex (S1 in Figure 2.1) receives the afferent signals; this region is further divided in parallel subregions associated to different functions (i. e., proprioceptive, cutaneous, or processing of both). Nevertheless, these subregions are extensively connected, and there appears to be an internal hierarchical structure of neurons with different roles. Beyond the primary somatosensory cortex, signals go to the *secondary* somatosensory cortex (S2 in Figure 2.1), which processes tactile memory, and to posterior areas (Brodmann 5 and 7) where multi-sensory integration occurs.

Finally, it is noteworthy that functional Magnetic Resonance Imaging (MRI) studies support the distinction of two main subsystems within the somatosensory system [LK09]: (i) the *what* subsystem, which processes the properties of surfaces (e. g., material attributes: roughness, stickiness, etc.) and objects (e. g., geometry and structure characteristics: shape, curvature, volume, etc.), and (ii) the *where* subsystem, which determines the layout and localization of those surfaces and objects in the world with respect to the body's reference frame.

2.1.2 Psychophysical and Psychological Aspects

Beyond the anatomical and neurological mechanisms, it is essential bearing in mind human physical abilities and psychological strategies that appear during haptic interactions. In fact, these have a direct effect on the parametrization of immersive virtual environments, and can be exploited in the design of haptic interfaces, as sought by Tan et al. [TSEC94].

At this stage, it is important to clarify that the act of *perceiving* can be described as the comprehension or inference of a *distal stimulus* (i. e., source of *sensory signal*) after processing the *proximal stimulus* (i. e., *sensation* created by signals on receptors) [RA93]. Hence, considering all sensory channels available, *perceiving* consists in creating an internal representation of the reality (aka. *percept*) after integrating *multimodal* (i. e., *multisensory*) signals.

Human Psychophysical Limitations

One of the essential principles in human psychophysics is the Weber-Fechner law [Fec60], which states that the minimum change in magnitude of a stimulus that can be perceived is proportional to the previously perceived magnitude. The principle applies to all senses and several dimensions; in particular, it has been exploited by Hinterseer et al. [HHC⁺08] as a mean for haptic data compression in overloaded communication channels (see Section 2.2.7).

This minimum differential threshold or discrimination value is often referred to as *Just Noticeable Difference* or JND, and usually comes expressed as a percentage of magnitude. In this sense, the JND has an average value between 7 – 10% for force perception over a range of 0.5 – 200 N with a minimum *resolution* of 0.06 N [Jon00]. Similar JND averages have been determined for other proprioceptive characteristics, such as: limb position (7%) and movement (8%), stiffness (17%), viscosity (19%), and inertia (28%).

Other important threshold values consist in the *bandwidth* and the *optimum operation ranges*. In this respect, Shimoga [Shi93] reported that fine vibrations of 10 kHz can be detected by the human hand. However, the 1 kHz maximum frequency is accepted in kinesthetic haptics as a convention due to technical issues, as explained by Basdogan and Srinivasan [BS02] (see Section 2.2.5). Further considering kinesthetic applications, O’Malley and Goldfarb performed several human performance measurements during square and round ridge detection and discrimination with varied force and stiffness magnitudes. A performance improvement saturation was found for force magnitudes of 3 – 4 N [OG02] and stiffness values of 220 N/m [OG04]. Nevertheless, note that a stylus-like and desktop-sized haptic interface was used in the experiments.

Tan et al. [TSEC94] and Lederman and Klatzky [LK09] provide further descriptions of evaluation methods and practical values related to human haptic capabilities, ranging from the study of factors that affect sensitivity (i. e., body part, application surface) to the spatiotemporal resolving of the skin and the perception of surface properties, respectively.

Exploratory Procedures, Haptic Invariants, and Priors

Lederman and Klatzky [LK87] described a systematic relationship between object properties and (free) manipulations required to perceive them. This idea was synthesized in a set of eight *exploratory procedures* that comprises the most important manual exploration patterns used for determining object properties with the haptic sense – these exploratory procedures are: (i) *lateral motion* for texture, (ii) *pressure* for hardness, (iii) *static contact* for temperature, (iv) *unsupported holding* for weight, (v) *enclosure* for global shape and volume, and (vi) *contour following* for global and exact shape; ad-

ditionally, functional testing actions are classified as (vii) *part motion testing*, and (viii) *specific function testing*.

During these stereotypical explorations humans can look for specific haptic *cues* or well-defined signals that provide information about the shape of the object being touched. These cues are the footprint of mechanical *invariants* that relate object shape with the (deformable) finger-pad shape, either during static or dynamic palpation. For example, a blindfolded person who rolls the fingers on a curved edge relates implicitly the linear and angular velocities of the fingers with the curvature of the edge; however, it is the sense of touch (i. e., tactile and kinesthetic sensations) the actual modality used for processing that relationship. Indeed, Robles de la Torre and Hayward [RdlTH01] have shown that humans identify and locate shape features based on the perceived force cues, independently of surface geometry. Furthermore, the fact that humans are able to recognize the properties of surfaces implies that there are *prior* knowledge structures accessed during explorations. Along these lines, Hayward [Hay08] analyzed typical invariants and discussed their applications, among others, in haptic interface design.

Multimodal Integration

In order to be able to improve perception quality, it is essential to understand how the brain combines information from different modes (i. e., senses or feedback modalities) and the positive and negative effects of each mode on others. There is evidence that proves how haptic perceptual tasks demand visual and multisensory cortical areas, supporting the idea of a meta-modal brain organized around task processing instead of separate processing of sensory information [LS08].

Srinivasan et al. [SBB96] found out that the display of visually stiffer virtual springs (achieved by artificially constraining the visual elongation) leads to the perception of haptically stiffer ones. In contrast, Ernst et al. [EBB00] later showed that active and consistent haptic feedback can dominantly reinforce the perception of slants displayed with conflicting visual cues (i. e., disparity and texture). This gave rise to the generalized multisensory integration framework proposed by Ernst and Banks [EB02]: as demonstrated by the authors, integrated percepts are a linear combination of unimodal sensory cues weighted by their relative reliability. This weighting reliability is inversely proportional to the variance or error estimated for each cue or mode. Hence, sensory dominance is governed by the reliability of the sensory inputs; moreover, it can depend on the task and/or prior experiences. Similar experiments with multisensory tasks have also produced results consistent with this model [HR09].

Regarding the bi-modal audio-haptic interaction, DiFranco et al. [DBS97] showed that virtual surfaces taped with haptic devices are generally perceived harder when display-

ing pre-recorded sound cues that are typically associated with tapping harder surfaces. Avanzini and Crosato came up with similar insights using synthetic audio cues [AC06]. In summary, taking into account these multisensory integration mechanisms, contact percepts can be modulated beyond the limitations of display devices.

2.2 Immersive Virtual Environments with Haptic Feedback

After having introduced the nature of human haptic perception in detail, this section deals with the most relevant components of virtual environments (VE) that consider that sense in feedback generation. Basically, the elements illustrated in Figure 1.1 and their interactions are described.

2.2.1 Human Factors

Stanney [SMK98] divided the human factors related to virtual interactions into three areas: (i) *human performance*, (ii) *health and safety*, and (iii) *social impact*. The second area is largely related to *cybersickness* (discussed thereafter), and the last one to addiction and de-sensitization towards violence (not in the scope of this work).

Stanney further identified three factors affecting this first area, human performance: (i) the *navigational complexity* in the VE, (ii) the *degree of presence* experienced by the user, and (iii) *task and user characteristics*, i. e., difficulty, experience, learning effects, spatial capabilities, etc.

From a more current and broader perspective, the first factor could be viewed rather as *interaction complexity*; interaction devices and techniques in VE are introduced in Section 2.2.4. In the following, the practical remaining topics related to human factors are elaborated: cybersickness, and presence issues.

Cybersickness

Cybersickness is the most common malady that appears during and even hours after the exposure to immersive VEs [SMK98]. When watching moving imagery, users experience the classical symptoms of motion sickness, which range from headaches to vomiting. Although the mechanisms are still not completely clear, three cause theories have been proposed, as summarized by LaViola [LJ00]: (i) the *sensory conflict* originated by contradictory signals from the vestibular and visual systems, (ii) as *ingesting poison* affects visual and vestibular channels, directly and uncommonly stimulating those could trigger similar reactions as when consuming a harmful substance, and (iii) virtual images display physically atypical or altered environments that cause a *postural instability* state until new control strategies are learned. Several mitigating factors have also been identified:

accurate and non-delayed position tracking, fast enough visual rendering in peripheral areas, interaction while sitting instead of standing, visualization of coherent rest frames, gradual exposure, and providing control to the user.

Presence Terminology

Slater [Sla03] addressed the terminology around the concept of feeling present in a VE. According to the author, while *immersion* is an objectively measurable fidelity indicator of a technology, *presence* is the subjective perception of “being there”, a psychological reaction to immersion. Typically, presence has been evaluated with questionnaires, such as the widely used from Witmer and Singer [WS98]. Additionally, as presence refers to the form, *involvement* refers to the interest in the content; one might feel very present in a simulation, but dislike what occurs in it.

Following this terminology, Bowman and McMahan [BM07] argued that, rather than focusing on presence constructs, it is *immersion benefits* (e.g., spatial understanding, peripheral awareness, etc.) resulting from implemented immersion technologies (i.e., increased field of view) that should be measured to gain insights on the application *effectiveness* or human performance. Furthermore, the authors pointed out that the goal should be to find sufficiently (i.e., minimally) *realistic* immersion components that increase user performance and sensation of presence. Altogether, the definition, measurement, and relationship between these concepts of *immersion*, *realism*, *presence*, *involvement*, and *effectiveness* is still an active field of research.

2.2.2 Multimodal Rendering

Synthetic rendering of a modality consists in creating artificial signals for a sensory channel, which are then displayed with an appropriate *device*. In the context of virtual manipulations, the most important sensory modalities are vision (graphic rendering), hearing (audio rendering) and touch (haptic rendering); their integration has been discussed in Section 2.1.2. For almost any virtual simulation, the visual channel is essential. Additionally, for manipulations, displaying contacts through kinesthetic haptic rendering is the most natural and effective method to achieve highest performance [SWH⁺12], [WSHP13]. However, kinesthetic contacts can also be *augmented* by other sensory modalities, or *substituted* by them, when complex and expensive haptic systems are not available. It has been proven that rendering several feedback modes increases user performance during virtual assembly tasks. In particular, Petzold et al. [PZFea04] reported that best results are attained when haptic feedback is augmented with either visual or audio cues.

Sreng et al. [SBG⁺07] presented a very interesting approach for multimodal rendering of contacts in haptic interactions. Visual, audio, and haptic cues are superimposed to different collision events (impact or detachment) and states (frictional sliding). These cues include: particles and bubbles, drawing lines following contacts, instantaneous Gaussian-like sounds or longer roughness sounds after preliminary modal analysis of the objects, or haptic force transients. All effects are parametrized with the contact dynamic properties, such as normal or tangent velocities.

An important technical detail is the computational frequency required by each modality: 60 Hz are necessary for rendering images in the visual loop [SMK98], 48 kHz for sounds in the audio loop [SBG⁺07], and 1 kHz [BS02] for rendering contacts in the haptic loop, as already mentioned. That difference in minimum update rates has popularized parallel architectures that handle modularly each feedback mode [OL06]. In these cases, *synchronization* and *low latency* (~ 25 ms) are essential.

For the *haptic* sense, further psychophysical limitations translated into technical requirements have been discussed in Section 2.1.2. Stanney [SMK98] reports additional guidelines for other senses. For instance, the *visual* signal should be rendered with stereo images, which require correct image disparities and custom interpupillary distance; besides, a proper resolution and field of view (desirable 100°) are to be used on the display device. Regarding *audio* rendering, binaural sound synthesis for correct source localization is helpful.

2.2.3 Physics Engines: Motion Computation

In the game development community, physics engines deal with (i) *collision detection* and (ii) *motion computation* of virtual objects. These engines perform collision detection usually with simplified geometries to reach interactive rates. Therefore, in general, interactive virtual frameworks comprising haptic feedback compute collisions and forces with own methods. Along these lines, note that haptic simulations are also possible without physics engines if no motion needs to be rendered. In this sense, only the *motion computation* or *rigid body simulation* functionalities from physics engines are considered in this work.

Computational frequencies of 500 Hz have been used in the physics loop [SBG⁺07]; however, depending on the application, achieving the visual update rate (60 Hz) suffices. Several physics libraries have appeared in past years, such as Nvidia PhysX [Nvi01], Bullet [Cou03], ODE (Open Dynamics Engine) [Smi01], or Havok [RC11a]. Boeing and Bräunl [BB07], and more recently, Hummel et al. [HWS⁺12], have evaluated and compared some of these freely available engines. The latter work highlights the large differences between them, constraining the choice to the application domain.

Bender et al. [BETC14] provide a comprehensive review of approaches used in interactive rigid body simulation, covering common models and numerical methods. In general, once *contact manifolds* are detected, the dynamic state of the rigid body is solved from the equations that govern its dynamics (e. g., Newton-Euler); then, this dynamic state is integrated to obtain future positions, giving place to body motion. Baraff reported an easy and elegant method for motion rendering following these ideas [Bar97a], [Bar97b]. In contrast to the high complexity characteristic of the collision computation problem, in theory, motion rendering of an unconstrained rigid body can be reduced to the step-forwarding of a 6D state variable under well-know laws. However, other issues must be handled, such as stability and physically correct parametrization.

2.2.4 Interaction Devices and Techniques

A plethora of consumer input and output (I/O) devices for immersive VR interactions have appeared since the advent of the so-called “second wave of VR” in 2012 with the Oculus Rift*. Anthes et al. [AGHWK16] give an overview, providing a classification of devices and highlighting the most important ones for each category.

In general, **input devices** can generate *discrete* events (e. g., buttons) or *continuous* streams of signals (e. g., tracking) [BKLJP01]. Common categories include (notable examples in parentheses): flysticks or similar controllers (Razer Hydra[†]), gamepads, joysticks, treadmills (Virtuix Omni[‡]), and tracking devices for the whole body (Kinect[§]) or body parts and hand gestures (Leap Motion[¶], Thalmic Myo^{||}, Kinfinitiy Glove^{**}). Human voice can be also considered to be an input device if speech recognition is applied.

On the other hand, **output devices** can be *fully immersive* (i. e., real world fully occluded) or *semi-immersive* [BKLJP01]. Typical classes for the **visual** sense comprise: head mounted displays or HMDs (Oculus Rift^{††}, HTC Vive^{‡‡}) and multi-screen systems like CAVEs [CNSD93], or powerwalls^{§§}. Regarding **output devices** for the sense of **touch**, most usual are vibrotactile vests (Subpac^{¶¶}) or armbands (VibroTac [SEWP10]) Additionally, kinesthetic haptic devices fall also in this category; these are discussed in detail in Section 2.2.5.

*<https://www.oculus.com>
[†]<https://www.razerzone.com/>
[‡]<http://www.virtuix.com>
[§]<https://developer.microsoft.com/en-us/windows/kinect>
[¶]<https://www.leapmotion.com>
^{||}<https://www.myo.com>
^{**}<http://kinfinity-solutions.com>
^{††}<https://www.oculus.com>
^{‡‡}<https://www.vive.com/eu/>
^{§§}<http://www.lcse.umn.edu/research/powerwall/powerwall.html>
^{¶¶}<http://subpac.com>

For all these devices, management and communication software frameworks are necessary. Taylor et al. [TIHS⁺01] presented in 2001 the VRPN (Virtual Reality Peripheral Network) library^{***}, a device-independent and network-transparent interface for VR peripherals of extended use in distributed settings. VRPN is now integrated into the OSVR (Open Source Virtual Reality) framework [BPT15]^{*}, which provides abstraction, management, and analysis functionalities (e. g., configuration, fusion, gesture detection) for many devices and VR platforms. Additionally, visualization tools have been extended from scene graphs (Coin3D[†], OpenSceneGraph[‡]) to VR frameworks (InstantPlayer[§]), and, more recently, to game engines that already support interaction devices and enable authoring (Unity[¶], Unreal Engine^{||}).

Interaction

Human-machine *interaction techniques* used to act within VEs are in close relationship with the integrated I/O devices. First of all, it is worth highlighting the difference between *natural* and *intuitive* interactions, as done by Zachmann [ZR01]. While the first is related to interactions similar to the ones occurring in the real physical world (e. g., touching and manipulating with our hands and fingers), the second refers to simplified metaphors that link abstract actions and expected outcomes. Intuitive interactions have proven to be very efficient, for instance, in the form of WIMP elements (Window, Icon, Menu, Pointer), or as gestures with the advent of smartphones in recent years.

According to Bowman et al. [BKLJP01] interaction techniques can be classified into three main categories:

- *Navigation* techniques, which should enable spatial awareness for travel and way-finding and be as lightweight as possible for the user to focus on the real task. Therefore, the motor and cognitive abilities requested by these techniques alleviated (e. g., by providing proper velocity and constraint control). Typical metaphors include: tracked body motion, hand space movement, steering, teleportation, and path drawing.
- *Selection* and *Manipulation* techniques are performed with the hands, and are often related to the 6D positioning of grabbed objects. Counter-intuitively, non-realistic

^{***}<https://github.com/vrpn/vrpn>

^{*}<http://osvr.github.io>

[†]<https://bitbucket.org/Coin3D/coin/wiki/Home>

[‡]<http://www.openscenegraph.org>

[§]<http://www.instantreality.org>

[¶]<https://unity3d.com>

^{||}<https://www.unrealengine.com/en-US/>

techniques have been shown to outperform more realistic ones, and limiting degrees-of-freedom (DoF) can increase performance. Usual metaphors comprise: pointing and ray-casting, grabbing of objects at hand-reach, growing of virtual arms for catching, and the manipulation of the relative virtual world scale.

- *System Control* techniques are used to change the state of the system to perform meta-tasks (e. g., configuration of the VR) and should be as transparent as possible. Typical metaphors include: gestures, voice commands, and menus.

2.2.5 Kinesthetic Haptic Devices and Control Issues

The haptic device or interface is fundamental in immersive virtual manipulations with haptic feedback. It consists in an electro-mechanical system that tracks user movements (e. g., position and velocity of the hand holding the device endeffector) and simultaneously displays virtual or remote contacts. Therefore, it is both an input and output (I/O) device. Note that *kinesthetic* haptic interfaces are considered in this section, i. e., devices that display forces through forces or torques applied on their joints, usually through motors. As a general requisite, it is expected that a haptic interface should cope with human *haptic sensitivity*; additionally, it should function as *seamlessly* as possible, and in a *secure* way.

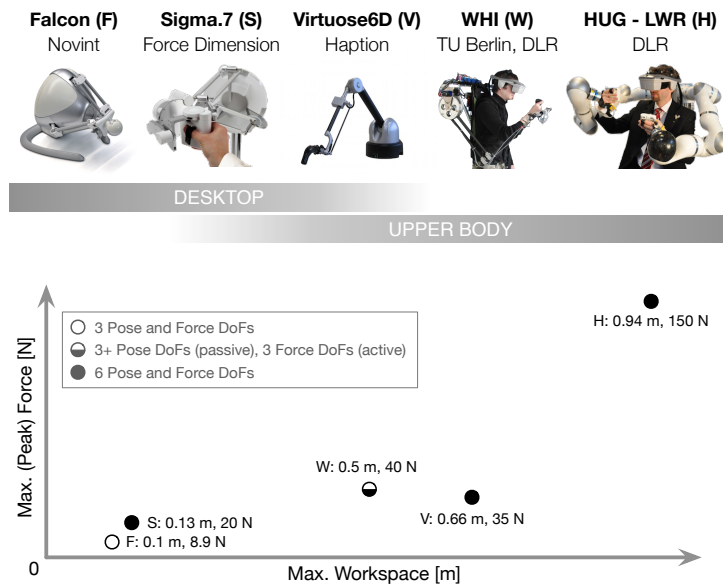
Figure 2.2 depicts the five devices integrated in the context of the presented work: the Falcon [Sta97], the Sigma.7 [THH⁺11], the Virtuose6D [GFL04], the Wearable Haptic Interface (WHI) [KDK10], and the HUG [HHK⁺11]. In particular, the Sigma.7 and the HUG were employed in the experiments reported in this document and they are described in detail in Appendix A. It is worth to mention that many of the early works in haptics were performed using the Phantom device [MS⁺94], one of the first commercially available interfaces. The Phantom is desktop-sized, with a serial kinematic and a stylus-formed endeffector.

The following paragraphs introduce the most relevant *design elements* of haptic interfaces, as well as a *classification*. Furthermore, the minimum or ideal *requirements* are discussed, alongside with *control issues* that have a direct impact in the perception of the user.

Design Elements: Mechanism, Sensors, and Actuators

A haptic device consists of a set of *actuators* and *sensors* arranged around a *mechanism* with several degrees-of-freedom (DoF). The **mechanism** is built with *links* connected with *joints* in *series* (e. g., Virtuose6D) or *parallel* (e. g., Falcon). The handle grabbed and moved by the user in which the mechanism “ends” is called *endeffector*. The resulting

Figure 2.2: Exemplary haptic interfaces, integrated in this work: the Falcon [Sta97], the Sigma.7 [THH⁺11], the Virtuose6D [GFL04], the Wearable Haptic Interface (WHI) [KDK10], and the HUG [HHK⁺11]. These devices cover the whole *workspace-force* domain and have also different *degrees-of-freedom (DoFs)*. In particular, the Sigma.7 and HUG interfaces were used for the experiments reported in this document and are thoroughly described in Appendix A. Pictures of the Sigma.7 and Virtuose6D are courtesy of Force Dimension and Haption, respectively.



mechanism kinematic should avoid the human operator from reaching *joint limits* and falling in *singularities*, which are degenerate configurations that restrict the mobility of the endeffector. Besides of that, the mechanism should not have unbalanced weights during operation, which can be achieved with passive mechanical elements, with electronics and software (e. g., HUG), or both (e. g., Sigma). In general, a mechanism and its kinematics can be evaluated by analyzing the singular values of the *Jacobian* matrix that relates joint and endeffector velocities [HO08]; for instance, the product of all values accounts for the overall *manipulability*, and the ratio between the extreme values for the *isotropy*. The elasticity of the mechanical structure can also be a factor to be considered in these evaluations. As a final note, it is worth to mention that mechanisms can be designed for single handed or *bi-manual* manipulation [TML14].

The **sensors** measure the position and forces (torques) of the joints. Rotary optical encoders that operate at least at 1 kHz [BS02] are typical. Additionally, they should be able to measure minimum values of 0.1 N, 1 mm, and 2° [FDS90]. Worse resolutions can have negative effects on the noise, the maximum stiffness, and stability. The **actuators** are usually DC motors able to exert custom torques. Low values should be sought for *friction*, *torque-ripple* (torque span in a revolution), and *backlash* (clearance or lost motion due to gaps between gear teeth) [HO08]. Furthermore, *back-drivability* or interactive transmission between input and output has been shown to be beneficial [PV14].

Classification: Workspace, Force, Degree-of-Freedom (DoF)

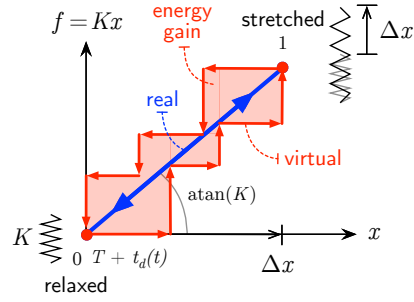
In the last decades, a big variety of interfaces have appeared, some of them very application-oriented, others focusing on specific challenges, and many balancing technological and performance trade-offs. From a practical point of view, it is possible to classify them with the values of a trinomial: the size of their effective *workspace*, the *forces* they are able to display, and their actuated mechanical *degrees-of-freedom* (DoF). Devices that share similar values for these three properties are expected to have similar behaviors.

The effective **workspace** of a device can be measured with the space volume covered by it or its maximum axis length. The latter typically ranges from 0.1 m (desktop) to 1 m (upper body), trying to match the human limb movements that it is targeting, e. g., wrist (Falcon) or whole arm (HUG), while minimizing link length and size; certainly, a smaller structural mass leads to smaller inertias and, ultimately, less fatigue. Devices can be *grounded*, i. e., fixed to a static base, or *ungrounded*, usually fixed to the operator's body, as an *exo-skeleton* (WHI). The felt stiffness is lower with the latter, but they have largely extensible workspaces, since the user can freely move in space wearing the device. In order to artificially enlarge the virtual or remote workspace, *scaling* and *indexing* or workspace drift control strategies have been proposed [CK05]. Along these lines, it is worth to mention that *hand exo-skeletons* constitute a particular category of haptic devices which focuses only on the local workspace of the hand; usually, designing a powerful mechanism that fits in such a small volume for so many (finger) DoFs is a very challenging task.

The maximum applicable **force** accounts for the hardness of the displayable contacts. Usually, higher force values are related to higher device mass and felt inertia, given that heavier motors are required. It is important to distinguish between maximum peak and maximum continuous or controllable forces; high peak forces might be fundamental if transients on contact events are going to be applied [KFN06]. It has been shown that human fingers can handle up to 40 N [SIT89], however, 10 N are rarely exceeded in normal interactions [MS⁺94]. The maximum controllable force reaches ~ 65 N for the wrist and ~ 100 N for the elbow and shoulder [TSEC94].

The **DoFs** of the haptic interface constitute the base directions in which the endeffector can be moved by the user. The user can apply motion along *passive* DoFs, but only *active* DoFs are additionally actuated. The first haptic interfaces had 3 active DoFs (translations recorded, forces applied), and with the time, devices with 6 DoFs have appeared (translations and rotations recorded, forces and torques applied). The human arm has in practice 7 DoFs, considering the null space movement of the elbow; moreover, the human hand has more than 20 DoFs, depending on the used kinematic approximation. As expected, higher numbers of DoFs can increase *dexterity*, but result in more complex and heavier mechanisms.

Figure 2.3: Energy gain in discrete-time simulations depicted with one-DoF, as pointed out in [BS02]. The force f on an ideal (real) spring of stiffness K wanders from point 0 (relaxed) to point 1 (stressed) along the blue slope when it is stretched Δx , following Hooke's law; as the spring is released, it goes back to 0 along the blue line again. A virtual spring modeling that same ideal one follows a stair-wise path (in red) when elongated the same amount Δx due to the sampling time T and the time delay $t_d(t)$. That path leads to an energy gain (shaded pink area) which might cause instabilities if not properly dissipated.



Z-Width: Range of Applicable Impedances

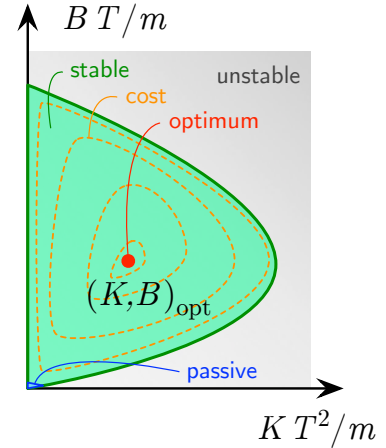
As stated by Massie and Salisbury [MS⁺94], when using a haptic interface, (i) free space must feel free and (ii) solid virtual objects must feel solid. The *Z-Width* or dynamic range of displayable *impedances* (i. e., force and velocity relationships, Z) is an indicator of how close a device is from that goal: (i) on free movement, the impedance displayed should be non-existent (i. e., neither inertia nor friction effects in any direction, $Z = 0$ kg/s), whereas (ii) on contact it should be able reach unlimited values (i. e., infinitely massive objects, $Z \rightarrow \infty$), while still being stable; however, this is mechanically not possible, and haptic devices work in practice within the (smaller) *Z-Width* range ($[Z_{\min} > 0, Z_{\max} \in \mathbb{R}]$). Colgate and Brown [CB94] investigated the most important factors for maximizing the *Z-Width* of haptic interfaces: device damping, sensor resolution, sampling rate, and filtering.

Control of Haptic Interfaces

Two main control architectures are used for haptic devices: (i) *admittance* control, in which user forces are measured and positions are commanded to the device to constraint user motions, and (ii) *impedance* control, in which user motion is measured and forces are commanded to the device. Certainly, the latter is the most widespread approach. Additionally, when possible, impedance-type devices are designed to be back-drivable and to have low friction and inertia so that *open-loop* force control is accurate enough for stable and performant operation [HO08].

However, any active electro-mechanical system can still become *unstable*, e. g., if more energy is generated than dissipated during its operation. Instabilities are characterized by buzzing, vibrations, or even uncontrolled fast motions that can harm the system and the operator. Hence, they decrease the usability of the system and can make it dangerous. Figure 2.3 illustrates how the *energy gain* that leads to instabilities appears in discrete

Figure 2.4: Passive (blue) and stable (green) regions, as well as the optimal operation point (red), according to [Hul17]; despite it is a schematic illustration, the relative proportions are realistic. Dimensionless regions are plotted on the domain defined by the virtual stiffness K and damping B for a sampling time T (ideally $T \leq 1$ ms), a given time delay t_d (ideally $t_d = 0$ ms), and a total mass m (device and human arm). Regions shrink if t_d increases. The optimum operation point depends on the selected cost function c (orange contours), e. g., minimum settling time, system energy, etc. Note that the stable region completely contains the passive one; additionally, the optimum operation parameters according to the selected cost function ($c_{\min} \rightarrow K_{\text{opt}}, B_{\text{opt}}$) are not necessarily at the boundary of the stable region.



electro-mechanical systems that exert forces [BS02] (e. g., kinesthetic haptic interfaces). The energy gain area of the figure increases with the *stiffness* of the virtual spring (K), the *sampling time* (T), and *time delay* (t_d)*. In general, if the energy gain exceeds the amount of energy that the human arm and the inherent friction of the mechanism can absorb or dissipate, the system is in risk of instability. A common and practical approach for dissipating the energy gain consists in applying virtual *damping* (B) to the spring model; however, contacts might feel sticky at some point, and damping itself can cause unstable behaviors in discrete systems if an upper limit is crossed [Hul17]. In this respect, systematic research has been done in the last decades to find out the correct parameters necessary for guaranteeing stability and good performance. Developed approaches can be classified as *passivity* or *stability-based* methods. Additionally, *optimal control* techniques have also been presented, seeking the most advantageous parameter values for given system behavior goals. Figure 2.4 shows the relationship between these approaches.

Passivity-based approaches are probably the most studied ones. They try to control the system so that net energy flow in it is zero, assuming the human operator can interact stably with a passive system. Colgate and Schenkel [CS97] discovered the now popular time-invariant *passivity condition* required by any system composed of a haptic device and a spring-damper virtual contact model (with no time delay):

$$b > \frac{KT}{2} + |B|, \quad (2.1)$$

*Furthermore, in the simplified model of Figure 2.3, the energy area is linear on the stiffness and quadratic on the time variables; in other words, halving the computation time of a haptic rendering algorithm can have the impact of quadrupling the maximum displayable stiffness – or vice versa. Nonetheless, the relationship is not that straightforward in practice, if all factors in the system are considered in the model.

being b the continuous-time damping of the device, and K , B , and T the discrete stiffness, damping, and sampling time of the virtual wall simulation, respectively. Another contribution by Colgate et al. [CSB95] is the *virtual coupling* method, which is explained in Section 2.3.3.3, in the context of haptic rendering. Later, Hannaford and Ryu [HR02] presented a time-variant approach that implemented a passivity observer and a passivity controller; basically, the observer measures the energy in the system and the controller activates a variable damping able to dissipate any energy surplus detected by the observer. Many other works have been published in this line, but the main limitation of passivity-based methods remains to be that they are too conservative: a passive system is stable, but a stable system does not need to be passive.

As an alternative, **stability**-based approaches try to find out the parameter boundaries (usually K , B , and T) within which under any input signal (e. g., step or impulse inputs), the system does not have an oscillatory response of increasing amplitude, but the reaction signal rather converges to a steady value within a finite time period. A notable contribution was done by Adams and Hannaford [AH99], analyzing a simple but complete model that covered most relevant factors.

Finally, **optimal control** approaches look for parameter values that minimize some user defined cost functions in the stable region. These costs can be, for instance, the settling time until the amplitude of the response oscillation decays below a certain percentage of the steady response value, or the mechanical energy over the time of the oscillation. In this respect, Hulin [Hul17] has recently provided decisive and practical insights using a complete system model that takes into account the human operator.

As shown in Figure 2.4, the passive region can be several orders of magnitude smaller than the stable one, and the optimal operation point usually does not lie on the stability boundaries.

2.2.6 Telerobotics

Telerobotics (or telepresence) applications have several similarities with virtual manipulations with haptic feedback. In them, the *human operator* controls with a *master* robotic interface (haptic device) a *slave* manipulator set in a *remote environment*; this control is possible thanks to a *communication channel* that transmits information between the endpoints. If *bilateral control* is used, both sides are input and output, and position-force data is interchanged. Niemeyer et al. [NPH08] distinguished at least three categories when classifying control architectures used in telerobotics: (i) *direct control*, with which the operator is fully coupled with the remote manipulator's actions, (ii) *shared control*, which distributes manipulator control between the user's coarser actions and a local autonomy, and (iii) *supervisory control*, in which the operator interacts usually with more

abstract commands and the remote manipulator can perform almost autonomously.

Especially in direct control approaches, the properties of *passivity/stability* (see Section 2.2.5) and *transparency* are fundamental, and unfortunately conflicting goals, as shown by Lawrence [Law93]. Transparency accounts for how close the *impedances* (Z) of the master (Z_m) and slave (Z_s) sides can be. As explained in Section 2.2.5, there should be no impedance on the master side during collision-free movements ($Z_m = Z_s = 0$ kg/s or N m s/rad) and the impedance should be able to reach unlimited values in contact situations ($Z_m = Z_s \rightarrow \infty$). Nevertheless, the range of impedances is limited to the achievable Z-Width of the device ($[Z_{\min}, Z_{\max}]$). In this sense, the Z-Width is a measurement of achievable transparency. Transparency has also been studied from a human perception perspective by Hirche and Buss [HB12], considering JND thresholds.

On the other hand, in all methods that go beyond direct control, a central task consists in *predicting and rendering contacts* using virtual models. Furthermore, accurate world and robot models are necessary as autonomy of systems increases. A known example is the ROTEX experiment performed by Hinzinger et al. [HBDH93], in which the first robotic manipulator in space was controlled from earth using predictive graphics under delays up to 7 s.

A type of shared control method are *virtual fixtures* [Ros93], which consist in haptic overlays (e. g., support planes) that assist the operator during the task by reducing workload and increasing performance. Virtual fixtures have also been implemented in assembly simulations, as done by Tching et al. [TDP10]. A recent survey of this type of techniques is provided in [BDRyB14]. Focusing more on visual assistance, Hertkorn [Her15] presented recently a shared control approach for telerobotic grasping; with it, the operator moves a robotic hand-arm system to the target object, the cameras detect it, and feasible contact regions and grasp qualities are displayed so that the operator decides to grab the object or change the pose.

Model-Mediated Telemanipulation

Model-mediated telemanipulation can be considered a shared control approach which presents a solution to stability issues and the degraded intuitiveness characteristic of bilateral telemanipulations functioning under large delays; this is achieved by relaying the user interaction to a local virtual model of the remote environment. Mitra and Niemeyer [MN08] presented a proof-of-concept for one degree-of-freedom (DoF). The scene, assumed to be static, was registered with the collisions detected with the slave manipulator and modeled with planes. The authors showed how their method works with delays of 1 s, during which the operator interacted locally with the virtual model computed in runtime.

More recently, the popularization of optical depth sensors has led to approaches able to model more complex environments. For instance, Xu et al. [XCANS14] presented a three-DoF model-mediated teleoperation framework using streamed point clouds. In that work, a time-of-light camera at the slave side registers and filters the point cloud of the remote scenario (at max. 25 Hz); additionally, the physical properties (i. e., contact stiffness and dry friction coefficient) of the environment are estimated through the force sensors at the endeffector. For the communication, human perception deadbands are exploited by sending only values out of the JND thresholds, which reduces the transmission rate. During the interaction, the operator sees the images of the master environment, but feels the forces computed by a haptic rendering algorithm [RC13a] which uses the point cloud. In other words, although the user moves the slave through the master, the contacts recorded at the remote side are not displayed, but rather used to build and update a local model on the master side, which is used to render the applied virtual forces. In the subjective user evaluations, 9 out of 10 subjects failed to distinguish between the measured and these applied virtual forces.

2.2.7 Haptic Communications

The correct transmission of haptic information, i. e., without lags or *delays*, is key for stable and transparent interactions. Unfortunately, the 1 kHz update rate convention leads to high packet loads that may worsen transmission quality. Those points hold true especially for telepresence interactions (see Section 2.2.6), in which, as opposed to virtual simulations, the communication does not occur locally, but end-points might be located far away from each other; in those situations, *packet loss* is often unavoidable, degrading the quality of experience (QoE) with artifacts like bouncing, stickiness, or roughness.

Steinbach et al. [SHE⁺12] summarized additional issues that appear in haptic communications and provided a survey of common approaches to tackle them. Some methods have already been mentioned, for instance: (i) *passivity-based techniques* that observe the energy flow in the system and damp it to guarantee stability (Section 2.2.5), (ii) *perceptual coding approaches* which send data only when JND thresholds are violated relieving channel load (Section 2.1.2), and (iii) *model-mediated strategies* that build local virtual representations with which is interacted under large delays (Section 2.2.6). Regarding packet loss and error-resilience, (iv) *redundant channels* built with multiple UDP connections (User Datagram Protocol) are used.

In the context of virtual simulations, *predictive filters* have also been employed for situations in which computation time could exceed 1 ms. In this sense, Hou and Sourina [HS16] continuously directed computed virtual forces to a buffer rather than to the user; in a separate thread, linear regression was applied on the last 300 values of the

buffer to predict the next force and torque vectors, which were subsequently smoothed using a B-spline function also aware of previous prediction values. Those smoothed values were the finally displayed forces.

2.2.8 Applications

Collision computation and virtual force rendering have many applications in several fields of *robotics*, such as *motion planing* [PM11], *telepresence* with large delays [NPH08], or even *autonomous* robotic applications that can be learned with physically-based virtual models [CB17].

Virtual environments with haptic feedback in which human users interact in realtime have also been shown to have a wide range of application fields. The most popular are probably *medical simulations* [CMJ11] and *virtual prototyping* [SVO11], [Xia16]. With these interactive simulators, doctors and engineers can *verify* techniques and prototypes, respectively, and *train* for real world operations, decreasing risk and required processing time.

Beyond these use cases, virtual environments with haptic feedback have been successfully used for *rehabilitation* [BDSR06], [JCRR09], *scientific visualization* augmented with haptics [COPG15], *atomic or molecular manipulations* [TRC+93], *modeling* and creative arts [LBF+02], *education* [Bar10], and *games* and entertainment, as shown by Novint*, the company that produced the Falcon [Sta97] haptic interface. Furthermore, these examples can be multi-user or *collaborative* [KKT+04].

Hannaford and Okamura [HO08] and Lin and Otaduy[LO08] provided more references to selected applications using haptic feedback. Similarly, Chapter 5 reports varied and practical implementations of use cases in which the technologies presented in this work are deployed and showcased.

2.3 Realtime Collision Computation and Kinesthetic Force Rendering

In interactive physically-based simulations with haptic feedback, (i) *collisions* between objects must be detected, (ii) contact *forces* must be rendered, and, in some cases, (iii) the *motion* of the manipulated object needs to be resolved or adjusted. These three processes occur usually in sequence, and the input of each of them is the result of its predecessor. In particular, force rendering and display has to run at 1 kHz, as explained in Section 2.2.5. Among the three, collision computation is usually the computationally

*<http://www.novint.com>

most expensive task, reaching worst-case performances of $\mathcal{O}(n^2)$ if naïve techniques are followed. Given that, and also due to the fact that it is required in many applications, collision computation has been extensively studied in the past three decades in fields like computer graphics, robotics, and virtual reality.

Table 2.1 and Table 2.2 collect more than 100 relevant works that report methods applied in the fields of collision detection and (kinesthetic) force rendering. The publications cover the advances in the last three decades and are ordered chrono-alphabetically. Almost 70 *features* which account for the attributes of the algorithms were detected. These features belong to the type of *output* delivered by the algorithms, the used *data structures*, and the implemented *methods*. As the reader might see, most algorithms make use of a combination of different data structures and methods. For each feature, the most significant works have been highlighted with a shaded cell. Additionally, for each work, a maximum of two other related works from the table have been selected, in case of strong influences detected. Note that Table 2.2 also classifies the methods presented in this thesis.

These tables do not cover surveys, comparisons, or user evaluations, but only the technical methods that have considerably shaped the landscape of haptic rendering. Upon interest, the reader might also consult other reviews and classifications that have been presented in past years. For instance, Lin and Manocha [LM04] summarized the most important methods for collision and proximity queries. Later, Lin and Otaduy [LO08] presented a book which collects all relevant haptic rendering techniques published until 2008. More recently, Otaduy et al. [OGL13] have surveyed and classified the most important force rendering methods.

The rest of this section explains in a structured manner the classification features from the tables, and it is organized as follows: after a brief *overview*, most common *data structures* are introduced in detail in Section 2.3.1. These object representations are used by *collision detection* techniques explained in Section 2.3.2 and by *force rendering* methods presented in Section 2.3.3. Finally, Section 2.3.3.6 briefly discusses how *deformation* can be handled in computer haptics.

Overview

A fundamental pre-processing step common to all collision computation and force rendering algorithms lies in generating *data structures* for all objects in the scene, which commonly consists in converting a *polygon soup* (i. e., unstructured lists of polygons) into a more useful *object representation* for the task at hand. Indeed, typical polygonal structures used for visualization are often poorly suited for collision computation. A proper object representation should consider defining constraints such as whether bodies

are rigid or deformable, whether topology is constant or can interactively change (e. g., material subtraction), geometrical complexity (e. g., objects can be simplified to convex shapes), and resolution. Moreover, the methods inherent of the data structure used for accessing the conforming *primitive features* of the object (e. g., points, faces, etc.) should be fast and robust. Except in some sections from this review chapter, this work considers *rigid bodies that are not altered during simulation but which are defined with arbitrarily complex geometries*.

Once object data structures are created, the collision detection pipeline could be regarded as a filter which delivers colliding features (e. g., object points in collision) out of some moving objects in space, as described by Zachmann [Zac01]. The process is interfaced with a (i) *front end queue* which stores objects, commands and queries from external modules or users. Then, for each frame or cycle, (ii) *broadphase* and (iii) *narrowphase* collision detection are performed, one after the other. In the broadphase (global level), moved objects are processed to determine the likely colliding pairs, whereas in the narrowphase (object level) object pairs are checked for exact contact features. The survey of this section focuses on narrowphase methods, which in some cases can also be applied to the broadphase; techniques that belong exclusively to the broadphase are described in Section 2.3.2.4, where *multibody* scenarios are discussed. Chapter 3 describes a narrowphase algorithm and Section 5.2 from Chapter 5 reports, among others, a collision computation framework for multibody scenarios.

Although closest or contact features are the most usual result in collision detection, other types of outputs can be expected, such as overlap volume, collision time, or even the constrained movement of the penetrating object to the collision surface. In this respect, a common problem faced in physically-based virtual manipulations is the *pop-through* [ZS95] or *tunneling* effect. This occurs when the collisions of points against *thin shells* or *non-watertight* objects (i. e., objects in which no interior volume can be differentiated) are missed. This is the case when a point is on different sides of a surface in two consecutive time steps, but not close enough to it so that it is classified as colliding. Several approaches that tackle that issue are discussed in this section, and, in particular, Chapter 4 provides a fast and robust method targeting the phenomenon.

Contact information can be directly or indirectly used in force computation, depending on the adopted force rendering *paradigm*; this feature is registered in Table 2.1 and Table 2.2, and can be *penalty-based* (P in the tables), *constraint-based* (C), and *impulse-based* (I), as later explained in this section. Another important property of force rendering approaches are the degrees-of-freedom (DoF) or dimensions of the computed values, which are usually three (forces) or six (forces and torques); this is strongly related to the capabilities of the used haptic interface. Beyond regular contact forces, several additional outputs are also possible, such as friction or material properties. Due to sta-

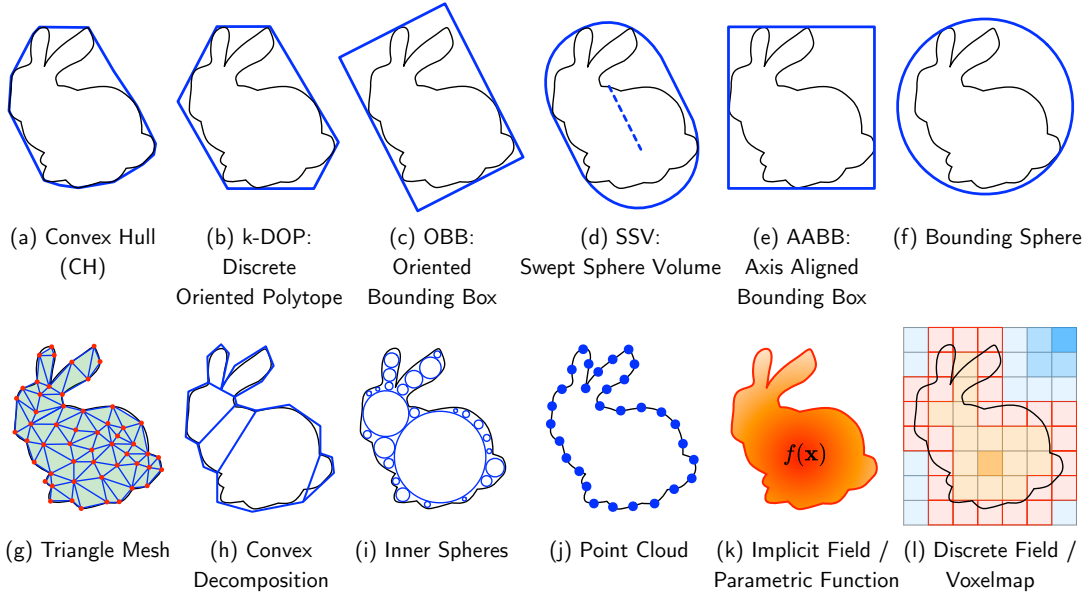


Figure 2.5: *Polygon soups* are usually pre-processed to create object representations or data structures used during collision and force computation. Upper row, (a)-(f): Bounding Volumes (BV), ordered from (left) *tight* and *computationally expensive* to (right) *loose* and *computationally light*; BVs allow for conservative but fast overlap checks and are usually organized in hierarchical representations (see Figure 2.7). Bottom row: (g)-(j) Representations based on sets of discrete *primitive features* or elements; (k)-(l) Representations based on fields, continuous or discrete. Data structures must allow for fast and robust access of the primitive features that conform the object (e. g., points, faces, etc.), and might have to consider deformation or different resolution levels.

bility and perception reasons (seeSection 2.2.5), displayed forces must be rendered at 1 kHz. In order to cope with such a high computational speed, some methods use multi-rate approaches with asynchronous loops, in which the dedicated force rendering thread computes forces using the latest available contact information (usually less frequent) and the current device pose.

2.3.1 Object Representations

Figure 2.5 illustrates the most common object representations used for collision and force computation. A proper description of these data structures is fundamental, since they strongly influence the methods used in realtime queries. It is worth to mention that efficient data structures often encode (offline) necessary *pre-computations* that are used in online calculations, relieving the computation effort in critical situations. Furthermore, one must be aware of the *accuracy* achieved with the used representation, compared to the original object; this is especially true for representations that break down and simplify the object into sets of discrete primitive features.

Object modeling or Computer-Aided Design (CAD) tools such as Blender* or Creo† export generally **polygon soups**. These lists of unordered polygons represent *polyhedra*, and can be structured as *triangle meshes*, which usually describe *connected neighborhood maps*: in them, each geometric primitive feature in the structure (i. e., vertices, edges, and faces) is aware of and can access its neighbor features*. This attribute is fundamental for many collision detection approaches explained in Section 2.3.2 (e. g., [LC91]). Several requirements or additional properties can be applied to general polyhedra, such as normal vectors for all features, or *convexity* of the shape†, which would change the topology of a general polyhedron. Convex meshes have been extensively used in many works (e. g., [GJK88]), because in contrast to non-convex or concave meshes, discarding overlaps between them requires easier and computationally less expensive methods. Along these lines, *convex decompositions* of general or non-convex geometries are also common (e. g., [EL00]); with them, the original polyhedron is divided in a set of convex patches.

Polyhedra are often enclosed in part or entirely by basic shapes or **Bounding Volumes (BV)**. Thanks to them, conservative but much faster overlap or proximity queries can be performed; if two BVs do not intersect, their contained geometries will not either. Typical BVs include: spheres [Qui94], [Hub96], swept spheres or capsules [LGLM99], Axis Aligned Bounding Boxes (AABB) [CLMP95], [vdB97], Object Oriented Boxes (OBB) [GLM96], Discrete Oriented Polytopes (*k*-DOP) [KHM⁺98], [Zac98], and Convex Hulls (CH) [GME⁺00]. Note that all of them are *convex*.

At least three factors need to be taken into account when it comes to choosing the appropriate BV: (*i*) the complexity of the BV geometry, (*ii*) the fitting efficiency of the enclosing volume, and (*iii*) the ease of update after motion. Usually, the simpler the BV (i. e., less bounding faces or constituting vertices), the easier and faster will be the collision queries – but also the more conservative; hence, a trade-off must be often met. Common measurements of the enclosing efficiency of a BV are the volume ratio between the BV and the enclosed polyhedron, as well as the *Hausdorff distance*, which is the maximum Euclidean distance between the contained and the enclosing surfaces. The third factor is mostly related to rotations, which have a bigger impact on BVs defined by fixed planes or directions (e. g., world frame directions, as in the case of AABBs). In those situations, every plane must be rotated each cycle by affine transformations, or, if that is not desired due to other algorithmic constraints, BVs need to be recomputed so as to comply with the allowed directions.

*<https://www.blender.org/>

†<https://www.ptc.com/en/products/cad/creo>

*For instance, a triangular face is composed of three neighbor edges and three vertices, has at least six edges converging to its vertices, and at least six contiguous faces.

†The shortest Euclidean segment (straight line) between any two points inside a convex shape is contained in the convex shape. If that is not possible, the object is *non-convex* or *concave*.

In particular, spheres are the simplest and easiest to update BVs[‡], leading to faster overlap checks and updates, but worse fittings. On the other hand, CHs are the most general and usually better fitting BVs, but they also require the most expensive overlap checks. Ideally, a CH is the convex polytope which optimally (with the lowest volume) encapsulates a geometry. Many works have focused on the computation of accurate and approximate CHs for in several fields related to computer graphics or robotics (e.g., [BDH96], [MG09]). Aforementioned k -DOPs are closely related to CHs, which encapsulate objects with k intersecting half-spaces or oriented planes, approximating the CH.

Less common BVs include sphere sectors [FUF06] and cones that enclose face normal vectors instead of the faces themselves [JWC05]. In addition, Inner Sphere Trees (IST) consisting of spheres that bound the object in its interior by greedily filling it have also been proposed [WZ09a], as well as BVs that result from combining several BVs, such as the Intersections of Spheres or k -IOS [ZK12], built of k intersecting spheres that approximate minimally bounding ellipsoids, or CHs dilated with spheres and tori [EMK07]. The last two BVs have the property of being *strictly convex*, which implies in practice that the distance between BVs is always continuous and derivable; that property is very interesting for path planing and optimization algorithms.

Finally, it is worth noting that most data structures based on BVs are organized in **Bounding Volume Hierarchies (BVH)**, which consist in trees of nodes with different sizes that enclose all the parts of the object (e.g., Sphere BVHs [Qui94] or OBB BVHs [GLM96], among others). Section 2.3.2.5 further deals with BVHs and their associated methods.

Beyond polytopes and BVs, **Point Clouds** have become a usual representation for objects in recent years; this is probably due to public domain libraries that have appeared and enable processing them, such as the Point Cloud Library [RC11b], or also due to the emergence of low cost 3D scanners like the Kinect* that make possible registering them inexpensively. Point clouds are basically point sets placed on the object surface, optimally with a uniform distribution and surface normals. These sets can be unconnected and unordered *point soups* (e.g., as in [MPT99]) or, more commonly, they can be embedded in more complex data structures that allow for fast access of neighbor elements for each point in the set. Among others, these structures can be sphere hierarchies [KZ04], k -d trees [LCS12], or similar. It is common to locally create implicit planes or surfaces from a point or point subset at runtime, as well as local implicit distance functions [KZ04], [LCS12].

Two particular and strongly related types of point clouds are *depth maps* (aka. *depth*

[‡]Only four real values are necessary to describe a sphere in 3D space; additionally, spheres are rotationally invariant due to their symmetry, which reduces queries only to the translational space.

*<https://developer.microsoft.com/en-us/windows/kinect>

images or *Z-buffers*) and *streamed point clouds*. The first are pixelmaps of proximity values that account for the distances of the surfaces in the scene from a viewpoint. These can be obtained from 3D scanners or virtual images, and can be efficiently used for collision computation [KLR15]. The second type are continuously (usually at 30 Hz) updated depth images of real environments (obviously, registered and delivered through 3D scanners). Several collision and force rendering algorithms have been recently proposed, e. g., against single interaction points [LCS12], [RC13a], distance fields [RC13b], or sphere trees [KWZ17]. With streamed point clouds, sensor noise filtering and downsampling are important issues to consider. Methods with streamed point clouds have led to a qualitative improvement of model-mediated telepresence applications [XCANS14], since they allow for the interactive registration of unstructured remote environments, as explained in Section 2.2.6.

Spatial Partitioning or subdivision techniques have also been applied for managing and representing object geometries. With them, object space is segmented or embedded in 3D grids consisting of smaller space cells; these grids are usually uniform (same cell size in all three directions) and often encoded as hashed look up tables. Each cell in the grid can point to an object or object part, which can be represented in various forms: OBB [GLGT05] or AABB trees [PSCM13], bounding planes [AMF16], vertices or points [SC12], spheres [RMB⁺08], occupancy values (solid or free space) [MPT99], distance-to-surface values [XB14], or density values (typical in medical scans)[CCBS11].

A special non-uniform spatial partition often used for point clouds is the *k*-d tree [LCS12], [KLR15], which is basically a binary search tree with *k* depth levels or dimensions. On the other hand, uniform grids are often arranged in *octrees*, hierarchical structures in which each node is divided in eight children of finer resolution. Some collision computation methods using octrees are: [MPT99], [PSCM13], [HDB⁺14], and [AMF16]. Moreover, uniform grids that contain occupancy and distance values are occasionally referred to as *voxelmaps* [MPT99].

Implicit Representations consist in *scalar fields* in which each point \mathbf{x} in space is associated with a scalar or field value, following a locally or globally known relationship $f(\mathbf{x}) = 0$. In other words, and focusing on the 3D Euclidean space, they are mappings with the form $f : \mathbb{R}^3 \rightarrow \mathbb{R}$. Typical field values comprise density values, occupancy values, and distance values.

The last type of value corresponds to *distance fields*, with which for any given point \mathbf{x} its distance to the surface of the represented object can be obtained. Additionally, *iso-surfaces* can be implicitly defined with them by fixing the field value. If interior and exterior spaces can be distinguished in a distance field, it is considered to be a *signed distance field*. Furthermore, implicit representations can be discretized using spatial partitioning, giving rise to the aforementioned *voxelmaps*. Although field values in voxelmap

cells are discrete, continuous and smooth responses can be computed for every point \mathbf{x} with local (*trilinear*) *interpolation*; usually, this requires processing the scalars in the neighboring voxels of the queried point.

Several relevant works have been presented on distance fields with applications in many areas [S⁺03], [JBS06]. Some of them have focused on data structures for collision computation [FSG03], [XB14]. Indeed, efficiently implemented distance fields are very attractive for such time-critical applications, since they provide fast and straightforward proximity responses, being able to achieve $\mathcal{O}(1)$ independently of the complexity of the represented object.

Parametric Representations can be used to describe complex but highly smooth surfaces. NURBS (Non-Uniform Rotational B-Splines) are a particular and well known example of them. These are one of the most used surface data structures in CAD (Computer Aided Design) environments, because of their compact representation, high continuity, and exact derivation of surface normals [TJC97]. In a parametric representation, the object is framed in a *control mesh* constituted by strategic vertices or *control points*; for each control mesh patch, well-known *basis functions* are evaluated and weighted as a function of mesh surface parameters, and they piecewise define the object surface. Hence, following the notation of implicit representations, 3D parametric surfaces are mappings of the form $f : \mathbb{R}^2 \rightarrow \mathbb{R}^3$, i. e., from 2D parameter space on a control mesh to 3D points on a surface.

Constructive Solid Geometries or CSGs are formed combining *mixed basic shapes* (e. g., boxes, spheres, cylinders, cones, or even Platonic polyhedra) in binary trees using Boolean operators (e. g., union, intersection, difference); note that those basic shapes are often defined with closed-form analytical expressions. Additionally, they can be used as analytically defined BVs without any binary trees [vdB99].

CSGs are common in CAD or solid modeling environments designed for manufacturing applications. Not surprisingly, typical modeling actions in those programs (e. g., extrusions, protrusions, etc.) are conceived to resemble common operations in traditional additive and subtractive machining, which in turn, resemble the Boolean operations with basic volumes that form CSGs. Usually, CSGs lack of explicit features common and useful in polyhedra (i. e., faces, edges, or vertices), but have the advantage of containing design history is embedded in the tree. Although not usual for collision detection and haptic rendering anymore, several works have been published using CSGs, e. g., [Zei93] or [SLY99]. Determining whether a point is inside a CSG is broken down to checking this point against the underlying primitives and deducing then the answer by following the correct boolean operations.

Although it is not inherently an object representation, it is necessary bearing the **Configuration Space** when describing data structures. This is the space of all three

translation and three rotation values, i. e., the 6D space defined in $\mathbb{R}^3 \times \text{SO}(3)$. For all feasible relative configurations between the pair of queried objects, all the points leading to collision or overlap are detected, forming a manifold of 6D points in the configuration space known as *contact space*. This structure simplifies the problem of handling two geometries to handling only one point against the new computed 6D geometry (the contact space itself), but at the cost of increasing the dimensionality of the operation space or the complexity of the processed geometry (e. g., number of vertices).

In addition to collisions, generalized penetrations (with translational and rotational components) can be computed in the configuration space [ZKM07], even artificially forcing continuous penetration values [ZKM14]. The construction and refinement of the contact space has also been accelerated with support vector machines [PZM13], usual in machine learning.

The contact space is strongly related to the *Minkowski sums*, or more precisely, the *Minkowski subtractions* [GJK88]. These are computed by adding to each point contained in one geometry all the mirrored points (i. e., negative) of the other geometry. This structure has interesting applications in collision detection and path planing, and therefore, it has been widely exploited, as further explained in Section 2.3.2.2.

Typical structures used for computing **deformations** are *particle* or *mass-spring systems* [CKB03] and *tetrahedron meshes* [SSB13]. It is worth to mention that these data structures used for computing deformation are usually not employed in collision checks. Instead, they are commonly embedded in other representations introduced so far, such as BVs, which need to be quickly updated during the interaction. Efficient methods have been proposed for updating them, such as predicting for each vertex the trespassing of its enclosing BVs according to their velocity, and modifying in each time step the BVs of which violation is most imminent [ZW06].

Similarly, rather than using the structures employed for deformation computation, *intermediate representations* are processed for force computation. These are basically abstractions of the complex environment (e. g., as tangential planes) located closely to the operator’s hand or tool pose [AKO95]. These local simplifications can be force transfer model linearizations [GO09] or compliant spring-damper mechanisms [PND⁺11]. Whereas slower loops take care of simulating deformations, update and processing of collision structures and intermediate representations needs to happen at 1 kHz if haptic interaction is to be guaranteed.

2.3.2 Collision Computation

Collision computation comprises the processes of detecting contacts and providing measurable witnesses that describe them. It is often the bottleneck in physically-based rigid

body simulations, and it precedes to collision response, which deals with force rendering and motion simulation. The following sections introduce most common methods for collision computation from the literature, which are strongly dependent on the object representations explained so far.

2.3.2.1 Collision Output

A pair of independent rigid bodies A and B are *disjoint* if they do not share any space: $A \cap B = \emptyset$; or, otherwise, *colliding*: $A \cap B \neq \emptyset$. This collision state is of *overlap*, if they share the same volume space, or of *surface contact*, if their surfaces are touching without any shared volume space. Furthermore, collision between them can be determined in several stages or levels of information:

- **Binary** collision output, which simply states whether the pair of objects A and B is *colliding* or *disjoint*. Considering objects are commonly represented by discrete elements or *features* f_A and f_B^* , even a binary collision check can have quadratic complexity in a worst case scenario where each feature of one object must be tested against all the features of the other object [Cha84].
- If objects are disjoint, the Euclidean **distance** d between them or the shortest path length for surface contact can be determined:

$$d(A, B) = \min d(f_A, f_B), \forall f_A \in A, f_B \in B, \quad (2.2)$$

with the feature pair that realizes (2.2) being the *closest feature pair*. If features were points, $d(f_A, f_B) = \sqrt{\|f_A - f_B\|_2}$. Although the closest features are not necessary for determining the distance between objects (e. g., when distance fields are used), these are interesting for many applications and have been computed already by the earliest algorithms (e. g., [GJK88], [LC91]). In general, there might be several closest feature pairs (e. g., in the case of two parallel planes facing each other); in this respect, the distance vector formed by them is *not inherently continuous in time* and the distance magnitude is *not differentiable*. Since path and motion optimization algorithms often require differentiable distance and constraint functions, several proximity computation methods with *continuous gradient* output have been developed. Some of them use *strictly convex geometries* introduced in Section 2.3.1, such as sphere combinations [ZK12], torii, or convex hulls inflated by the previous [EMK07]. Others tackle the problem by projecting the vertices

*The type of feature is given by the chosen object representation; for instance, a polygonal mesh is constituted by *vertices*, *edges*, and *faces*, whereas a bounding volume hierarchy (BVH) has its BV nodes as features.

of the configuration space onto a hypersphere to make distance and penetration computations close to the real ones but continuous and differentiable [ZKM14].

- If A and B are colliding, the **penetration depth** p that accounts for the size of their overlap is defined as the minimum path length necessary to bring the objects to surface contact:

$$p(A, B) = \min \|\mathbf{p}\|_2 \mid (A + \mathbf{p}) \cap B = \emptyset, \quad (2.3)$$

considering one object (B) still for the sake of simplicity, and without loss of generality. Although this classical definition is used by many works (e.g., [KLM02], [KLM04]), it is effective only in the translational space. Other definitions have also been provided, such as the *generalized penetration* [ZKM07], which considers the rotational space as well. Analogous discontinuity properties as with the distance hold for the penetration depth, especially for the points lying at the medial axis of the penetrated object [TMOT12].

- **Penetration volume** is another measure that accounts for the size of the overlap, $\text{Vol}(A \cap B)$. In contrast to penetration depth, this metric has been less exploited, probably because it usually requires additional computations; common methods include summing up volume slices between penetrating surface features [HS04], [FBAF08], or directly using volumetric elements that are checked for overlap [WZ09a].
- Finally, the **time of impact** can be provided. This value is characteristic of *continuous collision detection* (CCD) methods which consider continuous motion rather than static configurations sampled at each time stamp. If an object pair transitions from a disjoint state (instant t_0) to a collision state (t_1) in two successive time stamps, the exact collision instant is given by

$$t = \min t \in [t_0, t_1] \mid A(t) \cap B = \emptyset, \quad (2.4)$$

again, considering one object (B) is still. In addition to the time of impact, the relative object configuration on collision ($A(t), B$) is interesting in order to simulate realistic overlap-free interactions. CCD methods that approximate the collision time and configuration are explained in Section 2.3.2.3.

Collision computation leads to the processing of further response phenomena, such as forces, motion, deformation, and topology modification (i.e., material subtraction). These will be reviewed in Section 2.3.3.1.

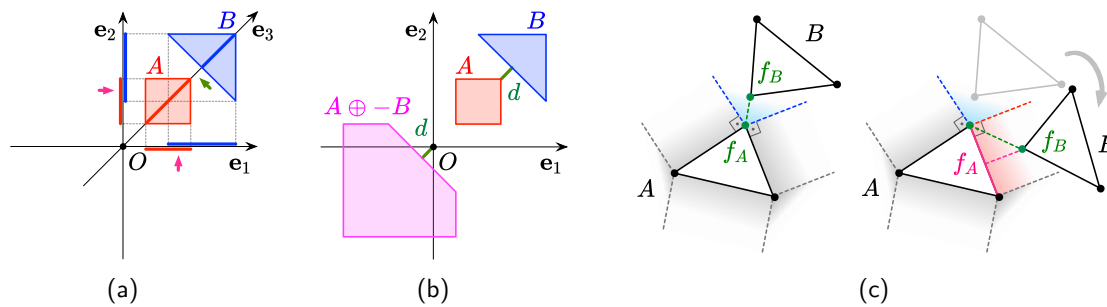


Figure 2.6: Common methods in collision and distance queries. (a) The *Separating Axis Theorem (SAT)* states that two convex objects do not overlap if and only if there exists an axis on which the projections of the objects are disjoint; in contrast to axes e_1 and e_2 , axis e_3 has disjoint projections, thus objects A and B are not colliding. (b) The distance from origin O to the *Minkowski subtraction* $A \oplus -B$ determines the distance (d) or penetration between objects A and B . (c) *Exterior Voronoi regions* of all the features (vertices, edges, faces) of object A are shadowed; the closest feature f_B of object B must be (at least partially) in the exterior Voronoi region of the closest feature f_A of object A (blue), and vice versa; Closest features are tracked on the exterior Voronoi tessellation following the direction of the cell boundaries that are violated.

2.3.2.2 Basic Methods

This section explains some essential theorems and approaches that have had a significant impact in the field of computational geometry and collision detection, and which have a direct application in collision computation; Figure 2.6 illustrates them. Note that all these and similar basic methods are commonly broken down to algebraic operations between basic elements in Euclidean geometry, such as points, lines, planes, or surfaces expressed in closed form. These operations are considered prior knowledge and are not reviewed in this chapter; in this respect, the interested reader might consult [AM01], [Moe97], and [Ebe99].

Separating Axis Theorem (SAT)

The *Separating Axis Theorem (SAT)* states that *two convex objects overlap with each other if and only if there exists no axis on which their projections are disjoint*; or, in other words: *two convex objects do not overlap in case there exists an axis on which their projections are disjoint* (see Figure 2.6(a)). An axis with disjoint projections accounts for the existence of a *separating plane* that fences off the two convex objects in two separate half-spaces. The application of the theorem consists in finding a separating axis among relevant candidates; in practice, for 3D convex polyhedra it suffices to evaluate the projections on the directions of face normals and their cross products.

This theorem is widely used in collision detection, especially with overlap checks be-

tween simple BVs, such as OBBs [GLM96] or AABBs [vdB97]. The amount of candidate axes to be checked increases quadratically with the number of features, thus, it has been proposed to reduce the pool of candidate axes to those which lead to highest probabilities of correct test results [vdB97]. The notion of the *separating plane* inherent to the SAT has also been applied to more complex convex geometries using machine learning techniques [Zac00]. The idea consists in incrementally improving a plane which would separate the vertex set of one convex object from the other's with a perceptron; the method stops if the plane is found or after a maximum number of iterations. Although not as efficient as the previous ones, linear programming or optimization approaches can be used to obtain the coefficients of the separating plane [LM04].

Minkowski Sums, Minkowski Subtractions

The *Minkowski subtraction* of two convex objects A and B is the *Minkowski sum* of one object (A) and the mirrored of the other ($-B$), defined as

$$A \oplus -B = \{\mathbf{a} - \mathbf{b} : \mathbf{a} \in A, \mathbf{b} \in B\}, \quad (2.5)$$

where \mathbf{a} , \mathbf{b} are primitive features (e.g., vertices) of objects A , B , respectively (see Figure 2.6(b)). This structure is called Configuration Space Obstacle (CSO) in the literature and has two important properties:

- (i) convex objects A and B overlap if and only if the origin point \mathbf{O} of the space in which they are defined is inside their Minkowski subtraction or CSO,
- (ii) the distance from the origin \mathbf{O} to the surface of the CSO is either the *distance* or the *penetration* between A and B , depending on whether \mathbf{O} is outside or inside the CSO.

Hence, the problem of collision and proximity computation between two convex objects (A and B) is reduced to checking a point against the convex hull that encloses the CSO. The popular Gilbert-Johnson-Keehrti or GJK algorithm [GJK88] exploits these properties to determine iteratively the distance between convex polyhedra. The approach is similar to the well-known Dantzig's simplex algorithm [Dan63] employed in linear programming. First, a small subset of witness or support points located on the surface of the convex set is defined. The key idea is to iteratively update this subset walking on the surface until the solution point is found. The walking direction is given by the point in the subset which minimizes the distance to the origin.

Several enhancements have been applied to the original GJK algorithm, such as, using hill-climbing by starting with the vertex that yielded the solution in the previous

cycle [Cam97], support of several geometric shapes in addition to polyhedra (i. e., boxes, cones, spheres, cylinders) [vdB99], and computing penetration depth [vdB01].

Feature Tracking Based on Exterior Voronoi Marching

Given a convex hull A , the *exterior Voronoi region* of any of its features f_i (i. e., vertex, edge, face) located on its surface ∂A is the set of exterior points that are closest exclusively to that feature; thus, the exterior Voronoi region of f_i in 3D space can be formally defined as

$$V(f_i) = \{\mathbf{x} \in \mathbb{R}^3, \mathbf{x} \notin A : d(\mathbf{x}, f_i) < d(\mathbf{x}, f_j); \forall i \neq j, \forall f_i, f_j \in \partial A\}, \quad (2.6)$$

being $d(\cdot, \cdot)$ the selected distance function (usually the two-norm for the Euclidean distance); Figure 2.6(c) illustrates the case for a triangle.

The pair of closest features of convex polyhedra has an interesting property: *each closest feature must be fully or partially contained by the exterior Voronoi region of its closest pair*. The popular Lin-Canny (LC) algorithm [LC91] exploits that property for detecting and tracking closest features. In a preprocessing step, a mesh of features is constructed so that each feature is aware of its direct neighbor features. Additionally, exterior Voronoi cells are computed for each feature by setting orthogonal planes to the face features. In runtime, closest pairs of features are iteratively determined by marching on the exterior Voronoi regions. If the candidate pair determined in the previous iteration satisfies the aforementioned property for closest features, the candidate features are the solution and the seed or initial guess for the next cycle. If not, a new candidate pair is defined by selecting neighbor features; this selection happens following the direction of the Voronoi cell boundaries (planes) that are violated by any of the candidate features. Since object configurations change minimally from frame to frame, a greedy walk in the neighborhood of the current approximation leads to a local solution, which turns out to be global due to the convexity of the polyhedra. Hence, the method is able to achieve fast and near to constant computation times after initialization.

The approach works with non-convex geometries after applying convex decomposition (see, for instance [GME⁺00]), but with the cost of quadratic complexity on the number of convex parts. Similarly, multi-resolution hierarchies of convex hulls have been successfully used with the method [EL00]. A significant improvement of the method was presented with the V-Clip (Voronoi Clip) algorithm [Mir98]. The key idea on which the method builds up consists in clipping parametrized candidate closest edges with planes of the Voronoi regions. Depending on the resulting intersection points, further decisions on marching directions can be taken, converging ultimately to the solution. This improvement avoids iteratively computing distances between features or performing unsta-

ble divisions, which boosts computational speed and robustness. Moreover, penetration computation is also supported. Altogether, the LC algorithm and its derivatives have been and are still widely employed, as recorded in Table 2.1 and Table 2.2.

Other Basic Methods

A large body of works uses *implicit* representations for collision and proximity queries, and to lesser extent, *parametric* geometry descriptions. As introduced in Section 2.3.1, implicit fields map space points \mathbf{x} to scalar values through a global or local field function $f(\mathbf{x}) = 0$ – the most common scalar distance values are considered in this section. Given $f(\mathbf{x})$, it is straightforward evaluating the collision state and distance/penetration of a point. Instead, the most complex tasks consist in (i) selecting the relevant features that are checked in the field function, and (ii) building or locally computing the field function itself.

For the first task, the aforementioned methods can be used, often in combination with Bounding Volume Hierarchies (BVH) explained in Section 2.3.2.5. For the second, several methods have been presented:

- If $f(\mathbf{x})$ is a global closed-form function, no additional computations must be carried out to use the field function [ST97].
- When regular grids (e. g., *voxelmaps*) are used to discretely store distance fields, *linear* [SSeS14a] or *trilinear interpolation* [BJ08] can be used to locally approximate the distance field function $f(\mathbf{x})$. That requires fast and robust access to the neighbor cell values. Many works based on the principles of the widely extended Voxelmap-Pointshell (VPS) algorithm [MPT99], [RPP⁺01], [MPT06], follow these methods, such as the approach presented in Chapter 3. In these approaches, points sampling the surface of an object are tested against the discrete implicit distance field of the other object.
- Depth maps computed from different perspectives (e. g., three orthogonal directions) also implicitly encode a discrete field that can be locally processed to render iso-surfaces [KLR15]; similarly, support planes or height maps set on the faces or vertices of the convex hull of the object have also been proposed [Mou15]: each discretely sampled point of a support plane maps to a distance value related to the proximity of object’s surface, and, in runtime, point penetrations are obtained after evaluating them in support plane mappings.
- In the case of point clouds, it is possible to fit planes or surfaces in regions close to the interaction tool, and to build distance functions to them [KZ04]; similarly, a

weighted average of these points can be computed to obtain the local approximation to the scalar field function and its gradient [LCS12]. In order to narrow down the subset of points close to the tool, several techniques have been proposed, such as sphere trees [KZ04], k -d trees [LCS12], or, in the case of streamed point clouds, simply accessing the points inside the projection of the tool’s bounding box on the depth map [RC13b].

Note that an important property of the field function is that its gradient $\nabla f(\mathbf{x})$ accounts for the normals of its iso-surfaces (i. e., geometric loci of same field value); thus, the closest path to collision should be aligned with $\nabla f(\mathbf{x})$.

As far as parametric representations are considered, it is common exploiting the control mesh as a first step for collision queries, either by projecting interaction points on it [TJC97], or by using BVs that enclose mesh parts [JC98]. When closest or colliding mesh points are found, these can be easily mapped onto underlying smooth surfaces [TJC97], and, afterwards, tracked in function of the object motion [NJC99]. To that purpose, first, the movement of the endeffector point in the Euclidean space can be projected onto the tangent plane of the parametric surface; this projection can be then used to compute a first order approximation of the parametric velocity of the previous closest point, which leads to the current closest point on the surface.

2.3.2.3 Discrete versus Continuous Collision Detection

The techniques for collision and proximity queries introduced so far provide an output given a *static* or *discrete* configuration for a pair of objects A and B . However, it is possible to detect collisions for a whole cycle or the time period between two discrete instants – this is done by *continuous collision detection* techniques, sometimes referred to as *spatio-temporal* or *4D* collision computation methods. These algorithms provide the exact impact time, as defined in Section 2.3.3.1, as well as the impact or *surface proxy configuration*. In general, continuous methods consist in discrete collision queries that are applied along a trajectory or a time interval defined by the *initial* and a *target* pose or instant. Therefore, while they make it possible not to miss any collision, they also tend to be computationally more expensive. Most approaches can be classified as methods using (i) *swept volumes*, (ii) *motion interpolation and iterative advancement*, (iii) *point motion parametrization*.

Swept volumes consist in spaces that simple BVs (e. g., AABBs or spheres) might occupy along a period of time. These can be computed in runtime if the initial and target poses of the object they encapsulate are known [Mir00], or by predicting the parabolic flight given their pose and velocity [DZ93]. Swept volumes can also be pre-computed, common when collision avoidance is performed with the links of robotic manipulators

and their environment [HDB⁺14]. In general, collision computation with swept volumes is very conservative: when a pair collides, their underlying BVs do not need to necessarily overlap within the analyzed time period.

Methods that apply **motion interpolation** bring iteratively objects close to each other with an interpolated motion which comprises translation and rotation and connects given initial and final states [ZLK06]; this interpolated motion can be linear [TMOT12] or even a screw-like [RKC02a]. Each approaching step can be defined by applying subdivision and interval bisection along the trajectory [CCBS11], or targeting configurations of expected collision [TKM09]. At each step, discrete collision detection is performed; in order to alleviate the computational load, this is preferably done by refining the accuracy with the number of performed iterations [TKM09] or testing prioritized features each time [CCBS11].

Interval arithmetics [RKC02a] is an elegant approach for tackling continuous collision detection with motion interpolation: instead of computations with floating point numbers, intervals of values are employed, being all basic (arithmetic) operations with intervals well defined. Once a pair of features (defined with initial-target value intervals) is found to be colliding, interval subdivision can be carried out to refine the impact moment.

Finally, **motion parametrization** is usually applied to points and vertices. This can be done by describing the trajectory of a point as a function of time, given its initial pose and velocity [MW88]; if two sets of vertices collide, the system of their parametric representations must have a real root for the time parameter. Another approach consists in testing for collision the line segment formed by the initial and target positions of the analyzed point. This method has been used for single interaction points [GLGT05], [HBS99], [RLB10], and for point clouds [XB17].

Altogether, continuous collision computation is common when *constraint-based* force rendering methods (explained in Section 2.3.3.2) are applied. Due to their computational load, it is also common to perform continuous collision detection in a slower separate loop and to resolve forces with a faster thread that accesses available contact feature information (e. g., see [ORC07]).

2.3.2.4 Multibody Scenarios (Techniques for the Broadphase)

Approaches for *multibody* scenarios or environments with several objects are commonly known as *broadphase* techniques. These employ similar methods explained so far (mainly for the *narrowphase*), but usually work with coarser resolutions that seek to determine only the likely colliding object pairs.

In worst case, collision computation in a multibody environment can reach a $\mathcal{O}(n^2)$

complexity, in the number of objects. However, in practice, considering common *large (sparse) scenarios* (e.g., [Mir00]), rarely do all objects collide against each other. Similarly, for *articulated mechanisms* (e.g., [MW88], [PM11]), collision pairs can be automatically sorted out (e.g., contiguous links of a robot).

A known method for handling the general multibody case is the *sweep and prune* algorithm [CLMP95], which has been widely employed in the community (e.g., by [HLC⁺97] and related). This approach encapsulates the objects from the scene in AABBs that are continuously updated with their movement. Then, following the SAT (see Section 2.3.2.2), intersections between those BVs are detected by projecting them on the three coordinate axes; these projections are basically intervals which are stored in incrementally sorted lists, while registering all overlaps. If a pair yields overlap in all three coordinate axes, exact contact detection can be performed in the narrowphase. The approach runs in $\mathcal{O}(n + m)$, being n the objects and m the overlapping AABBs; high numbers of objects can be handled, but keeping low densities to ensure low m values.

Other algorithms employ continuously updated *priority queues* in which each element points to a pair of objects that might collide, ordered according to their expected impact time [DZ93], [Mir00]. Additionally, *spatial partitioning* or *subdivision* has been exploited to filter out non-colliding object pairs [HDB⁺14]. This consists in marking each space regions or cell with the objects within it; cells with several object occupations lead to all object pairs to check.

This thesis also presents methods to tackle multibody scenarios in Section 5.2 from Chapter 5.

2.3.2.5 Acceleration Strategies

Due to its complexity, collision computation tends to become a bottleneck in the simulation pipeline. Several methods have been proposed to speed it up and this section collects the most important ones.

Bounding Volume Hierarchies (BVH)

Bounding Volume Hierarchies (BVH) consist of *nodes* of BVs that are ordered in trees, as illustrated in Figure 2.7. Each node encapsulates a part of the object, and its size depends on the *level* of the tree where it is located: the *root node* bounds all the object, whereas nodes close to the *leaf level* contain smaller object parts or limited sets of primitive features. The nodes are connected with links that establish *children* and *parent* relationships, so that each parent node bounds all object parts contained by its children recursively.

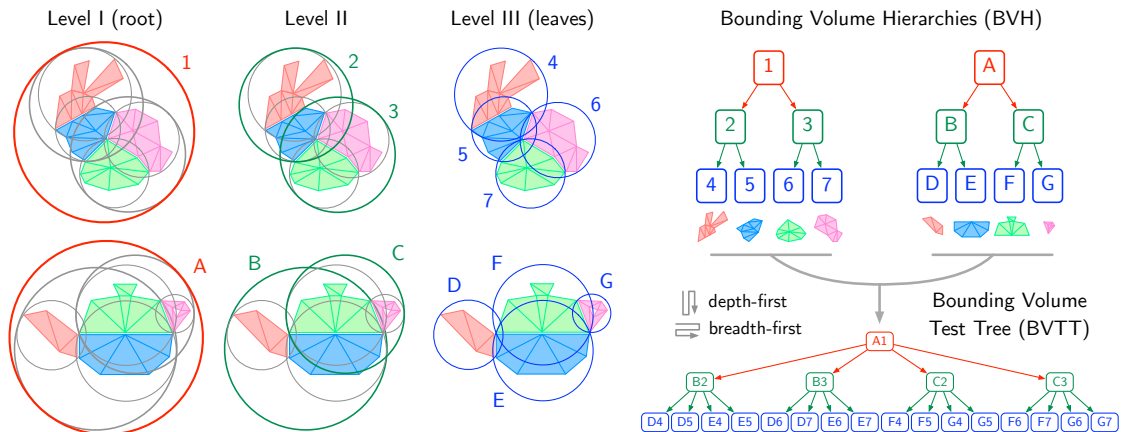


Figure 2.7: Bounding Volume Hierarchies (BVH) of the Stanford bunny and the Utah teapot; in the shown example, BVHs use spheres as *node* BVs and are *binary trees*, i. e., they have two children per node. *Root nodes* (level I, nodes 1 and A in red) contain all *children nodes* and primitive mesh parts. *Leaf nodes*, here in level III, point to primitive mesh parts. A BVH can be created *bottom-up* if leaf primitive features are clustered recursively, or *top-down* if the root BV is split recursively. The Bounding Volume Test Tree (BVTT) combines one-by-one nodes from two BVHs and it can be traversed *depth-first* if whole branches are evaluated for collision in a row (e. g., left to right), or *breadth-first* if each level is checked from root to leaf level.

Thanks to the BVHs, conservative proximity and collision checks between BV nodes are done recursively and refined progressively, determining very fast regions in the underlying objects that might be close to each other or in collision. For instance, in the example of Figure 2.7, if the distance between two sphere nodes is greater or equal than the current distance approximation, the sets of descendant sphere nodes can be culled, otherwise, the children nodes must be further examined. Once leaf nodes are reached, primitive features can be checked using the techniques explained in Section 2.3.2.2.

BVHs have several relevant features:

- The **shape of the nodes** can be any of the BVs introduced in Section 2.3.1, such as spheres [Qui94], [Hub96], swept spheres or capsules [LGLM99], AABBs [CLMP95], [vdB97], OBBs [GLM96], k -DOPs [KHM⁺98], [Zac98], and CHs [GME⁺00]. As mentioned, a trade-off between *fitting* and *testing efficiency* must be found, with the simplest BVs usually being the preferred ones.
- A defining property of the tree is the **degree** or *branching factor*, i. e., the number of children nodes each (parent) node has; this value influences the *height* of the hierarchy, i. e., how many *levels* it has from the root node downwards, and how many nodes of a given resolution each level has. Optimum values for the degree

have been discussed [KHM⁺98], with *binary trees* probably being the most common ones, i. e., trees with degree 2.

- The **building approach** can be *bottom-up* or *top-down* and it can affect the distribution of the nodes. The former starts with leaves or primitives and builds upwards grouping nodes close to each other taking advantage of any local information. The latter starts with the highest root node that encapsulates the whole object, which is then recursively split into smaller nodes; typical splitting heuristics include even or isotropic volume or primitive distribution along directions aligned with the ones determined after principal component analysis (PCA). It is worth to mention that, whereas each node’s BV can fully contain all its children and leaf primitives, in *wrapped hierarchies* [WZ09b] each BV contains exclusively all primitive leafs, without conservatively enclosing children node BVs. Thus, tighter representations are achieved.
- A fundamental characteristic is how the runtime **traverse** is done. Commonly, the Bounding Volume Test Tree (BVTT) is defined as the tree composed of nodes that combine the nodes of the two tested BVHs. The BVTT has the same amount of levels as the BVHs, but the number of nodes in each level is the product of the nodes of the BVHs in that level. Checking two BVHs implies traversing the BVTT, and the most common approaches for that are *depth-first* and *breadth-first*. The former starts in the root and explores one branch after the other, each of them as deep as possible. The latter starts also in the root but visits all the nodes in a level before jumping to the next level. While the first focuses in specific regions at a time, the second refines the sampling on the object gradually reaching the leaf nodes only at the end.

The cost of computing collisions with BVHs (i. e., the cost of traversing the BVTT) has been defined to be [GLM96], [KHM⁺98]:

$$T_{\text{BVTT}} = \underbrace{N_v C_v}_{\text{BV check}} + \underbrace{N_p C_p}_{\text{primitive check}} + \underbrace{N_u C_u}_{\text{BV update}} \quad (2.7)$$

with N being the number and C the unitary cost of each of the types of checks or updates. Clearly, all terms are in conflict; for instance, a tight BV decreases N_v and N_p , it but increases C_v and potentially C_u . In general, the appropriate number of BV checks (N_v) can significantly decrease the number of more costly primitive checks (N_p). However, worst case scenarios in which most of the surface is checked for proximity or collision lead to higher computation times than in situations in which primitive features are checked alone.

BVHs make possible *Level-of-Detail* (LoD) or *multi-resolution* checks, especially when breadth-first traverse is employed. If for each node a simplification or a sample of the underlying primitive features is stored (e. g., a simplified mesh or a set of points), each hierarchy level represents an approximation of the original object with a different resolution. Then, nodes can be further refined if the details of the children are perceptible [OL05], or *graceful degradation* [BJ08] in *time-critical* situations. This strategy requires refining the collision output every node check.

Spatio-Temporal Coherence

Under sampling frequencies of 1 kHz, *the configuration of objects will change minimally from frame to frame* – in other words, objects that are far away will remain so, and regions in collision will probably still collide for successive time stamps. If not an acceleration technique, this notion of *spatio-temporal coherence* is at least a fundamental assumption with strong practical applications:

- Queues of object, node, or primitive pairs to be checked can be *prioritized* or *ordered* according to their expected time to collision [LGLM99], [Mir00], [MPT06]. In the lack of this time value, it can be approximated by assuming an upper bound or maximum velocity and the current distance to collision. If that time lasts longer than the cycle time itself, the pair can be safely neglected; when the expected times are correctly updated, many superfluous checks are avoided.
- Previous colliding objects, nodes, or primitive features can be the seed or starting point for each computation cycle. The LC algorithm (see Section 2.3.2.2) is able to achieve close-to-constant computation times [LC91]. Similarly, *front tracking* [EL01] techniques have been used for BVHs; these consist basically in starting the traverse with nodes from a lower level to which they were previously colliding, instead of the root node itself.

GPU versus CPU

In the past decade, a shift in the focus of processor architectures has been happening: *parallelism* is increasingly sought (achieved with Graphics Processing Units or GPUs) in contrast to the more traditional *single core clock rate* approaches (related to Central Processing Units or CPUs). That shift of focus should be regarded considering the development of General Purpose GPU (GPGPU) frameworks, such as CUDA* or OpenCL†,

*<https://www.khronos.org/opencv/>

†<https://developer.nvidia.com/cuda-zone>

which have provided with abstraction interfaces for leveraging the power of GPUs for computations not necessarily related to graphics.

Although the methods explained so far are in theory agnostic with respect to the used architecture, GPGPU implementations do require specific designs that take into account the particular architectural and memory hierarchy characteristics of GPUs. In general, data structures need to be built in such a way that different parts of them (*i*) can be concurrently loaded in relatively small memory portions (compared to CPU implementations), and (*ii*) independently processed by threads that run relatively simple functions, known as *kernels*. Therefore, the use of hierarchical representations is a challenge.

Several works have started exploiting highly parallel architectures. Given the aforementioned technological constraints, some approaches define kernel functions for each of the (or selected) primitive features of the moved object; in these processes the features are tested against the environment, e.g., points against distance fields [ZLSWF13] or spheres against streamed point clouds [KWZ17]. Other works dealing with BVHs cluster queries that correspond to adjacent configurations in the same process [PM11]. In order to take advantage of the full capacity offered by GPUs, smart workload balancing techniques applied to BVHs have also been presented [LMM10]. In that contribution, tasks or instantiated kernels, which process small work units consisting of node pairs to be tested, are launched concurrently in separate cores. The result of processing each work unit may be a new set of children node pairs to be checked, which are added to the work queue of the task. Every task keeps its own work queue, which is accessible only locally, without needing synchronisation or coordination between tasks. Meanwhile, a balancing is performed between cores with a global counter.

Probabilistic Approaches

Given the computational cost of collision detection, some methods opt for avoiding to check features or elements with a low likelihood of overlap. As mentioned in Section 2.3.2.2, the number of axis tests has been efficiently decreased for the SAT method applied to AABBs [vdB97]. More recently, probabilistic overlap tests with BVHs which have collision confidence values as output have been used [PPM17]. For that, upper bounds of multidimensional Gaussian probabilities for collision are derived, since distribution densities cannot be obtained in a closed-form, and approximating them with Monte Carlo methods is computationally too expensive for realtime applications. This and similar methods could be of interest in several robotic applications.

Moreover, machine learning methods that incrementally compute contact configuration spaces by minimizing sampling have been exploited [PZM13]. This can be done by iteratively refining the resolution of the contact configuration space with support vector

machines (SVM), which are basically the samples used to determine the hyperplane that separates colliding and non-colliding regions in the highly dimensional sample space; in other words, SVMs are the key samples that are close to the contact boundary and therefore implicitly define it. In the refining process, given a configuration which needs to be tested, the nearest sample neighbors are processed to find the closest one. This closest sample is then projected on the contact space approximation defined by the support vectors, leading to a contact configuration.

Probabilistic approaches are not restricted to collision computation. Hou and Sourina [HS16] presented a predictive force filter for smooth haptic rendering. The computed contact forces are continuously directed to a buffer rather than to the user. In a parallel thread, linear regression is applied on the last 300 values of the buffer to predict the next force and torque vectors, which are subsequently smoothed using a B-spline function also aware of previous prediction values.

2.3.3 Collision Response: Force Rendering

Once collisions are registered, contacts must be resolved by simulating body motion and/or rendering contact forces to the user who is moving the object in contact. With respect to the collision computation process, these computations can occur (i) *concurrently* to it (e. g., [MPT99]), (ii) *sequentially* after it (e. g., [Mir96]), or (iii) in *parallel* processes separate from it (e. g., [OL06], [ORC07]).

This third group of *modular* or *multirate* approaches decouple collision and force computation with asynchronous threads to tackle the challenging 1 kHz frequency required in haptics. Linearized contact force models that speed up the display process can be used, for instance, analytical Taylor expansions of force expressions which predict the forces in the *haptic thread* (running at 1 kHz) using the slower but more accurate force and force gradient values computed in the *contact thread* [OL06].

Independently of the execution instant in time, this section presents the most common collision response methods from the literature, focusing on **(kinesthetic) force rendering**.

2.3.3.1 Overview of Output in Collision Response

Resolving collisions can produce the following outputs or effects:

- Contact *forces* and *torques* are the most important values necessary for kinesthetic haptic feedback. As already mentioned in the overview of Section 2.3, the contact resolution principle or *paradigm* and the number of *degrees-of-freedom (DoF)* (three

for forces, six for forces and torques) are defining factors. These properties are elaborated in the following Section 2.3.3.2 and Section 2.3.3.4, respectively.

- The computation of the *constrained movement* of the manipulated colliding object (tool) and *motion* of other objects in the environment is essential to build a plausible interaction. While the second is usually solved by integrating the Newton-Euler equations of motion (see Section 2.2.3), the first refers to an inherent problem in collision computation: in the physical reality objects do not overlap upon collision, but in simulations overlaps are often unavoidable to detect collisions. This issue is tackled with *constraint-based* or *virtual coupling* algorithms, presented in Section 2.3.3.3 and Section 2.3.3.2, respectively.
- Physical contact effects, such as *friction*, *force transients*, *texture*, or *shading*, increase the contact realism and enhance the overall haptic experience. Indeed, in the physical reality, surfaces are not perfectly slippery, transitions from free to contact states are markedly salient, and material properties like roughness are clearly perceptible. These topics are reviewed in Section 2.3.3.5.
- Object *deformation* and *topology modification* (e. g., due to material subtraction / destruction with holes created interactively) are realistic effects that can be produced by collisions between deformable and destroyable objects. Although not in the focus of this thesis, haptic rendering works dealing with them are briefly explained in Section 2.3.3.6.
- Finally, *multimodal effects* that provide collision feedback with *visual*, *audio*, or *vibrotactile* cues can be used to augment collision awareness or display contacts in the absence of appropriate kinesthetic haptic feedback devices. These effects have already been discussed in Section 2.2.2.

2.3.3.2 Three Force Rendering Paradigms

Rigid body simulation and force rendering methods are often classified into *three paradigms* depending on the used contact model*:

*See, for instance, the argumentation given by [CSC05], [WZZX11], [WZZX13], or [YWZX15] to this respect. However, it is noteworthy that hybrid approaches appear commonly, thus, that classification into those three categories should not be taken strictly. Interestingly, the seminal work in [Bar89] classifies contact simulation methods as *penalty-based* and *analytical* (analogous to *constraint-based*). Along these lines, the more recent survey in [OGL13] distinguishes also only *soft constraint* (i. e., *penalty-based*) and *hard constraint* (i. e., *constraint-based*) methods in haptic rendering, providing an elegant formal description of each paradigm.

- *Impulse-based* methods modify or correct the velocity of the colliding object by applying small impulses (distributed along several cycles) to prevent object overlap. They follow the principles of energy and momentum conservation and can be often modeled with the equation of the coefficient of restitution e , which evaluates the elasticity of the collision in terms of the velocities ($\dot{\mathbf{x}}$) of the colliding objects (A, B) *before* and *after* the collision:

$$\frac{\|\dot{\mathbf{x}}_A - \dot{\mathbf{x}}_B\|_{2,\text{after}}}{\|\dot{\mathbf{x}}_A - \dot{\mathbf{x}}_B\|_{2,\text{before}}} = e \in [0, 1]. \quad (2.8)$$

- *Penalty-based* methods approximate constraints or contact forces by penalizing constraint violations in their normal directions (\mathbf{n}) with spring forces that increase with the penetration value (p) or a similar penalty metric, such as overlap volume; they cannot avoid overlaps and lead to softer contacts, but are usually easy to implement and computationally inexpensive. Thus, the penalty force \mathbf{f}_P with stiffness k_P can be modeled in general as

$$\mathbf{f}_P = \sum_i \mathbf{f}_i = \sum_i k_P p_i \mathbf{n}_i. \quad (2.9)$$

- *Constraint-based* methods impose rigid constraints accurately to the body configuration and/or dynamics so that no inter-penetration between objects occurs; in other words, the configuration of the object is optimized subject to the unilateral collision constraints. Usually, these methods are more complex and computationally expensive, but lead to stiffer and more realistic force values. Once the transformation \mathbf{x}_{VC} that represents the difference between the *constrained* (optimized, in contact) and *unconstrained* (original, overlapping) is found, the constraint force can be computed with the appropriate stiffness (k_C):

$$\mathbf{f}_C = k_C \mathbf{x}_{VC}. \quad (2.10)$$

Damping terms proportional to the velocity can be added to the force models of each paradigm. Furthermore, torque values with respect to the object origin can be computed as the cross product between the lever distance from the force application point and the force itself.

In comparison to *impulse-based* approaches, notably more *penalty-based* and *constraint-based* methods have been published in the last years; this work itself presents the definitions and implementations of a *penalty-based* haptic rendering algorithm in Chapter 3 and *constraint-based* method in Chapter 4. Each of the chapters provides a deeper review of related works concerning each paradigm and the developed algorithms. For a

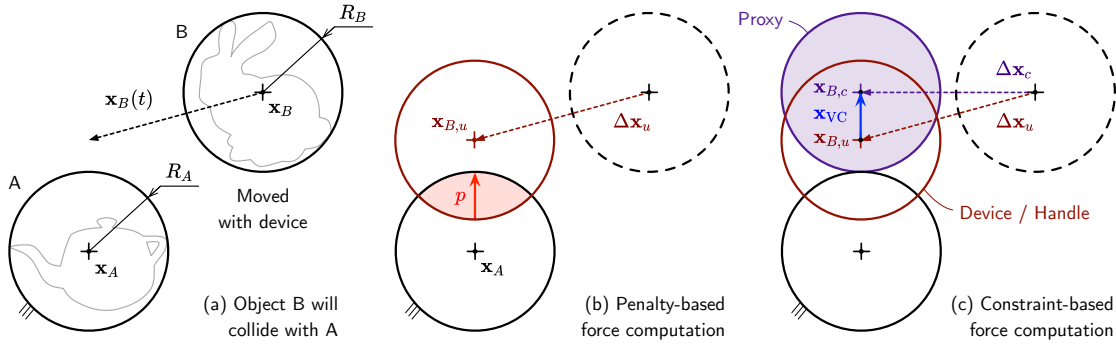


Figure 2.8: Schematics of *penalty-* and *constraint-based* haptic rendering approaches. Penalty-based methods require object overlap in order to compute forces out of an intersection metric (e. g., penetration, volume, etc.). On the other hand, constraint-based approaches restrain the virtual object to the contact surface upon contact and the forces are determined from the difference between the device pose and the constrained virtual object. Therefore, whereas the foremost important task for penalty approaches consists in computing the intersection magnitude, for constraint-based methods, the computation of a realistic constrained movement of the virtual object is the major process. In the constraint-based approaches, the unconstrained (\mathbf{x}_c) and constrained (\mathbf{x}_c) motions are distinguished, as well as the difference between both (\mathbf{x}_{VC}).

schematic though technical understanding of the differences between both paradigms, consider Figure 2.8.

One of the first notable examples of **impulse-based** approaches was presented by Mirtich [Mir96]. In it, after tracking closest features with the LC algorithm [LC91] (see Section 2.3.2.2), micro-impulses are applied to colliding pairs. These are modeled as differential equations and evaluated with local contact and object properties (e. g. relative velocity on contact point or mass distribution, respectively). Altogether, all contact types are unified (i. e., colliding, rolling, sliding, resting), easing the implementation.

Later, Constaninescu et al. [CSC05] developed a hybrid contact rendering method which implements impulsive and penalty forces; the first paradigm is applied when the objects move against each other (increasing overlap), whereas the second one is for resting cases or when objects move away from each other. The impulsive forces lead to an underdetermined system (i. e., more variables than equations) which is tackled with the assumption of Newton’s restitution law, i. e., considering that pre- and post-collision velocities are related with a scalar that accounts for the elasticity of the collision. A similar approach based on the difference in velocities was presented by Wang et al. [WCW+12] for a bone-burring application.

Ruffaldi et al. [RMB+08] also worked with an impulse-based haptic rendering method applied to *implicit sphere trees* embedded in voxelized distance fields. The method computes a cumulative impulse from the subset of colliding voxel pairs, considering only

pairs with a relative velocity that would not resolve the penetration in the next cycle. This process starts with the pair realizing the largest penetration, for which the first correction impulse (i. e., necessary velocity change) is calculated; this value is temporarily integrated for updating the velocities of the colliding voxels, and successive voxel pairs are then processed iteratively in the same manner.

The use of **penalty-based** forces modeled as springs on contact points appears in the earliest rigid body dynamics simulators due to its mentioned simplicity [MW88]. This model defines implicitly a conservative force field that tries to restore a non-penetrating state when objects overlap; in other words, the penalty force is the negative gradient of a penalty energy [TMOT12]. This holds for any penalty metric used, such as penetration [MPT99] or volume [WZ09a].

Despite the popularity of penalty-based approaches (see Table 2.1 and Table 2.2), several drawbacks must be pointed out: (i) if not corrected, objects visually overlap when collision forces are computed, (ii) many redundant contact points lead to an irregular distribution of the stiffness along all possible directions, resulting in lower effective stiffness values in order to keep stability, (iii) in some cases it is unclear which piece of internal volume should be associated with which surface, which might lead to force discontinuities, and finally, (iv) the *pop-through* or *tunneling effect* occurs with *thin shells* or *non-watertight* objects, as mentioned in the overview of Section 2.3.

Several approaches have been proposed to tackle these issues, such as (i) *virtual coupling* [CSB95] (explained in Section 2.3.3.3), (ii) weighting the stiffness in all the directions for each set of contact points to balance its overall distribution [XB16], or (iii, iv) continuous collision detection (see Section 2.3.2.3). However, there is one way to work them all out: *constraint-based* haptic rendering.

Baraff has been one of the first authors to present **constraint-based** approaches for contact force and motion computation following a linear programming or optimization formulation: the idea consists in optimizing object configurations to avoid interpenetration upon contact; necessary contact forces that lead to them can be determined along the optimization process or after it. A heuristic method is proposed for solving the optimization problem in [Bar89], whereas in [Bar94] the task is tackled with a method based on the Dantzig’s pivoting algorithm [CD68]. Although not suited for haptic update rates, these works cover all basic dynamic formulations necessary for plausible computer simulations that are referenced later on in the literature.

Zilles and Salisbury [ZS95] presented the first three-DoF constraint-based haptic rendering method for general polyhedra explored with a single point. In their implementation, two tool poses are considered: (i) the *Haptic Interaction Point* (HIP), which describes the endpoint of the haptic device in the virtual world, and (ii) the *proxy* or *god*

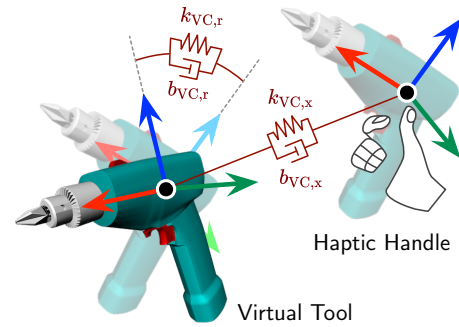
object, which follows the HIP but remains constrained to the surface upon contact*. It is important to note that, although the user moves the HIP, only the god object proxy is visualized. First, the pose of the god object proxy is computed as an optimization problem: the potential energy of the spring between god object and HIP is minimized constraining the proxy to the three closest face planes detected beforehand. Using Lagrange multipliers, the problem is transformed into a system of linear equations, which resolved, leads to the coordinates of the god object. Lastly, contact forces are rendered by a spring-damper system set between the god object and the HIP; this impedance-based force computation is very common under constraint-based algorithms, which usually spend most resources in determining the constrained god object pose.

Ruspini et al. [RKK97] applied the method to spherical proxies and polyhedra embedded in sphere hierarchies, and later, Ortega et al. [ORC07] presented the first generalization of the god object method for six-DoF. The major contribution of this *multirate* (i. e., multi-threaded) method lies in the simulation of the god object, i. e., in computation of the proxy pose. To that purpose, Gauss' least constraints principle [Gau29] is used. A rigid body under no constraints will follow the regular Newtonian motion equations with its given unconstrained acceleration. Assuming quasi-statics (i. e., velocity is considered negligible), the authors model the unconstrained acceleration proportional to the displacement vector pointing from the previous proxy pose to the current device pose. Gauss' principle states that the motion of a constrained body will deviate as little as possible from the unconstrained motion, minimising the kinetic norm or squared difference between the constrained and unconstrained accelerations. Hence, the problem is approached as a constrained optimization in motion-space, being the constraints the proxy's contact points from previous cycle. Formulations in motion-space, in contrast to more common contact-space, lead to improvements in conditioning, memory, and computational efficiency [RKC02b]. Wilhemsen's nearest point method [Wil76] is used to find the solution efficiently. Once the constrained acceleration is obtained, explicit Euler integration leads to the sought current proxy pose. As a last step, continuous collision detection [RKC02a] is performed for the travel between the last two proxy poses in order to detect the set of contact points (including surface normals) for the next god object simulation cycle.

Constraint-based methods based on other principles have also been presented, such as the three-DoF haptic algorithm for implicit functions by Salisbury and Tarr [ST97]. For obtaining the proxy pose, first, the closest surface point to the device point is calculated

*Zilles and Salisbury coined the term *god object*, used thereafter to categorize similar constraint-based algorithms to theirs. However, as noted by the authors, that designation had been previously used by Dworkin and Zeltzer [DZ93] with another meaning: virtual objects in physical simulations controlled by human users, and hence, being their motion not solely governed by dynamics laws.

Figure 2.9: *Virtual coupling* between the *haptic or device handle* held by the user (not visualized) and the *virtual tool* which collides with the environment. The stiffness (k_{VC}) and damping (b_{VC}) applied to the the relative translation (x) and rotation (r) between both frames renders the force and torque values applied to the user. The stability is not dictated by the virtual stiffness of the computed collisions, but by the parameters of the virtual coupling spring-damper system. Colgate et al. [CSB95] introduced the concept of virtual coupling and pointed out to the conditions necessary for the spring-damper parameters to guarantee passive interactions.



by steepest descent; the gradient and the penetration value of the of the implicit function at the device point give the step. Then, that point is tracked in two steps per cycle while the device point moves in solid volume: (i) the new device point is projected on the tangent plane of the previous proxy point (using the gradient) and (ii) the closest surface point of that projection is found by steepest descent, i. e., the new proxy point. This idea has been used in several other works, for instance, with signed distance fields or density fields [LYG02], and even with deformable models [CBS13].

2.3.3.3 Direct versus Indirect Force Display: Virtual Coupling

Independently of the force computation paradigm, two force display approaches can be distinguished in the literature: (i) *direct* and (i) *indirect* methods. With the first, the user is rigidly coupled to the virtual probe with which the collision forces are computed; thus, direct, immediate control is possible, but at the cost of possible instabilities if the environment stiffness increases. The second group of methods apply the *virtual coupling* technique, introduced by Colgate et al. [CSB95] and illustrated in Figure 2.9.

When virtual coupling is used, the user does not directly move the *virtual object* or *tool*, but instead, a *device handle* linked to it through a *spring-damper* system. The virtual object is the one visualized and used to compute collisions. In free space, the virtual object moves together with the device handle controlled by the user (through the haptic device), but upon contact, it is pushed by the environment surface while the user can still move the handle anchored to it. Thus, the displayed forces correspond to the spring-damper system between them that becomes active on collision, not to the contact forces of the virtual object. This implies that even when the stiffness of the contact might tend to infinity, the displayed stiffness is limited to the one of the virtual coupling, which is beneficial for controlling the stability. The authors further provide the conditions for

the stiffness and damping values in order to guarantee a passive system (hence, also stable) if the virtual coupling and the environment are updated in synchrony. However, softer and not the most transparent interactions could be perceived by the user.

Since constraint-based haptic rendering algorithms also use a very similar dual object representation, they could be considered to be virtual coupling methods. However, in general, virtual coupling methods do not optimize any proxy configuration as constraint-based approaches; they simply implement the spring-damper system between the visualized colliding tool and the handle. In that sense, virtual coupling can be used with penalty-based approaches, either by applying the penalty forces to the spring-damper system, or integrating the pose of the virtual tool from them.

Several variations have been proposed for the virtual coupling technique. Renz et al. [RPP⁺01] presented an approach which considers both the human user and the device, improving stability. In order to determine the virtual coupling parameters, Wan et al. [WM03] applied quasi-static equilibrium (i.e., sum of all forces equals to zero, neglecting effects of dynamics) on the system formed by the human hand grabbing the handle, the visualized virtual object, and the coupling spring between both of them. On the other hand, Otaduy et al. [OL06] employed non-linear saturated stiffness functions for the virtual coupling between the (integrated) virtual object pose and the handle pose.

2.3.3.4 Degrees-of-Freedom (DoF): Three (Forces) versus Six (Forces and Torques)

The Degrees-of-Freedom (DoF) supported by an algorithm during haptic interaction is a central property to take into account. Three-DoF methods are usually related to *force* rendering, while six-DoF approaches deal with *force* and *torque* rendering, and are, therefore, more complex. In three-DoF algorithms, the tool is typically represented as a point (or sphere) and the environment as a more complex geometry, whereas six-DoF methods support body-versus-body scenarios. In order to exploit the full potential of a six-DoF algorithm, a haptic interface able to display torques is necessary. Given that technical challenge, and also due to the algorithmic complexity, three-DoF approaches were more common at the beginning of computer haptics (see Table 2.1). One of the first approaches to handle six-DoF with complex geometries was the VPS algorithm (see Section 2.3.2.2) from McNeely et al. [MPT99].

As far as methods for torque computation are concerned, most principles and approaches have already been introduced: (i) application of lever arms from force application point to grasping point of center of mass (e.g., [MPT99]), (ii) the use of energy and momentum conservation (e.g., [Mir96]), (iii) twist forces related to the rotation necessary to match unconstrained and constrained tools (e.g., [CSB95]), or (iv) torques as result of optimization (e.g., [Bar89], [ORC07]).

Although for some applications six-DoF could be necessary, in other cases three-DoF is sufficient; this is especially interesting when six-DoF haptic interfaces (usually more expensive) are not available. Along these lines, researches have analyzed the implications of using three- versus six-DoF interactions. Wu et al. [WKH11] showed that, instead of being processed independently, force and torque are integrated in contact perception. Verner and Okamura [VO09] compared user performance during virtual drawing tasks without force feedback and with force feedback using different DoF. The use of force (three-DoF) and force-torque (six-DoF) conditions led to significantly smaller stylus penetration values ($\sim 16\times$) compared to the condition without haptic feedback; while the six-DoF condition was the best, the three-DoF condition seemed to be close to it. No completion time differences were found. Weller and Zachmann [WZ12] also compared three- and six-DOF interactions, this time in a bi-manual multi-player game where users had to pick objects with virtual hands. The six-DoF condition was shown to be superior in all objective and subjective metrics compared to the three-DoF case. In conclusion, it seems that the choice of DoF is very application-specific, and probably more research is needed in the area.

2.3.3.5 Enhancements for Fidelity and Realism: Friction, Shading, and Transients

Haptic information that arises during physical manipulations does not barely consist of contact forces; in addition to preventing grasped objects to slip away, several object properties can be recognized in the interaction, such as geometry, roughness, or material. That requires the realistic application of effects like *friction*, *shading*, *texture*, *initial contact transients*, or *vibrotactile* signals.

Among them, **friction** is probably the most studied one [ASB07]. Friction forces resist to the relative motion of surfaces and are often represented with the *Coulomb dry friction model*. This model defines the friction force \mathbf{f}_{fr} to be tangent to the contact plane, opposedly aligned with the relative velocity, and within the cone defined by the normal force \mathbf{f}_{n} and the aperture angle $\arctan \mu$:

$$\|\mathbf{f}_{\text{fr}}\| \leq \mu \|\mathbf{f}_{\text{n}}\|. \quad (2.11)$$

The *friction coefficient* μ is characteristic of the materials in contact and remains constant for each of the *static* or *kinetic* regimes the model foresees; the former refers to resistance forces that occur in absence of slipping, whereas the latter to situations with moving surfaces that are triggered when the static friction boundary is surpassed.

Harwin and Melder [HM02] extended the three-DoF god object method in [ZS95] to provide friction forces using a similar approach as in [ST97], which employs the Coulomb model. A friction cone with apex in the device point and base on the tangent plane of

the proxy is defined. The friction coefficient determines the aperture or angle of the cone. As the device moves, the proxy follows it constrained to the surface, however: if the old proxy pose is inside the cone base of the new device pose, the proxy will not move (static friction); in contrast, if the old proxy pose is outside of the new cone base, it will move until the base boundary (kinetic friction), not to the surface point closest to the device.

Beyond the Coulomb model, Hayward and Armstrong presented a very complete friction computation method for haptic rendering based on the Dahl model [HA00]. That model plausibly describes the change of the strain function upon friction, defined as the difference between a fixed point on the moving object and an adhesion point. The work improves the model canceling the drifting effect and provides a set of useful approximations. Both scalar and vectorial versions are discussed, but no torque friction is considered. The method is time-invariant and yields the four friction regimes observed in the physical reality: sticking, creeping, oscillating, and sliding.

Besides friction resistance, it is fundamental to filter edgy polygonal representations or discrete feature samplings to create smooth surface perceptions; in this respect, Morgenbesser and Srinivasan presented a force **shading** algorithm [MS96] that has been widely used. The approach resembles the Phong shading used in computer graphics [Pho75]. At runtime, the contact normal of the proxy is replaced by the average of its polygon's vertex normals; the distance from the proxy to the polygon vertices is used as the weighting. In the reported experiments, users had to distinguish between different polygonal bumps and completely cylindrical shapes. Results show that for the unshaded bumps, participants usually succeeded in identifying the shape; in contrast, under shading, the cylindrical bump was more frequently chosen than any other shape.

Similar shading and smoothing techniques have been successfully implemented by many authors to convey *texture* details, see for instance [RKK97], [ORC07], and [Mou15]. It is common to modulate the magnitude and direction of forces with oscillatory signals with amplitudes and frequencies obtained from contact and material information. Related *vibration* models have also been proposed, e. g., for bone-burring scenarios [WCW⁺12].

Finally, Kuchenbecker et al. [KFN06] showed that contact realism can be increased when high frequency **transients** are overlaid on collision events. While even common physical contact stiffness values are too high to be continuously displayed for most haptic devices, it is possible to exert short high force peaks that mimic initial contact perturbations. The authors implement that idea by triggering decaying transients on contact situations. These transients are superposed to regular contact forces, and include models such as fixed width pulses, exponentially decaying sinusoids, and acceleration matching functions; the latter consist in filtered force-acceleration transfer function models. Parameters such as duration, amplitude, and frequencies are obtained after observing real recordings, and a library of models is set up ready to be looked up at collision events.

At runtime, transient magnitudes can be scaled with the incoming velocity (related to the hand and device momentum which must be canceled). A thorough user study was conducted in which participants had to evaluate (without visual feedback) the realism of a tapping task compared to real wood tapping. Several real scenarios and virtual models and parameters were systematically randomized. The results show that transients have the highest fidelity among the virtual scenarios. Moreover, it is suggested that these could be more significant than stiffness or penetration values when it comes to displaying hard realistic contacts.

2.3.3.6 Deformation

Deformation has been thoroughly studied in computer graphics, but still remains very challenging in computer haptics. Collision and force computation with *deformable* objects is not in the focus of this work, however, this section provides a brief overview of the most relevant methods. For deeper insights, the reader might consider the reviews by Teschner et al. [TKZ+05] and Nealen et al. [NMK+06].

In general, *collision computation*, *force rendering* and *deformation* are treated as separate problems computed in independent threads that share critical information. As introduced in Section 2.3.1, *particle* or *mass-spring systems* [CKB03] and *tetrahedron meshes* [SSB13] are common for rendering deformation. In both cases, nodes (particles or vertices) are interconnected with one another, and their displacement upon external forces is computed. Tetrahedron meshes are common when the *Finite Element Method* (FEM) is used to solve deformation equations (boundary value problems) defined in continuous but discrete mass cells (usually, the tetrahedra themselves). With mass-spring systems, the mass is concentrated at the node points and the forces are propagated along the linking spring edges. While FEM produces more accurate deformations, mass-spring systems are computationally less expensive; however, constant volume deformation is difficult to model with them.

Independently of the used methods, it is common to speed up the processing of equation systems that lead to the deformed configuration with several shortcuts; for instance, the Sherman-Morrison formula for incrementally updating the inverse of the equation system, pre-computed basis deformation functions [JP01], modified Gauss-Seidel algorithms that converge to the solution of the chosen contact model very fast given an error tolerance [DDKA06], etc.

Intermediate representations have also been investigated as a method to support haptic update rates with deformable objects, as done by Garre et al. [GO09]. In their work, the deformable object or tool that interacts with the environment has a rigid part, the handle, which is connected to the haptic device. The coupling force between both

parts is linearized and fed to a virtual coupling model that renders the forces perceived by the user. For updating the linearized coupling force model, the velocity values of the handle, the tool, and the environment need to be solved from a nonlinear complementarity problem. During the simulation, the framework runs in two separate loops; a visual loop simulates the deformations of the tool using FEM at 30 Hz, whereas the haptic loop evaluates the updated linearized coupling force with the current handle state and delivers the subsequent forces at 1 kHz to the user who is moving the device. Peterlik et al. [PND⁺11] presented a related approach with an intermediate representation in the context of surgical simulations with haptic feedback. In this case, the intermediate representation is a system of compliant mechanisms that is shared by the slow dynamics loop, which updates the model according to the deformations of the objects using FEM, and the fast haptics loop, which evaluates the contact force on the model. In other words, the intermediate representation extracts from the FEM model used to compute deformations the mechanical compliance in contact used to render forces.

One elegant six-DoF haptic rendering method which covers all the necessary processes for interaction with deformable objects was presented by Barbič and James [BJ08]. The method builds up on the VPS algorithm [MPT99], which computes contact forces between distance fields and point clouds. These data structures are embedded in reduced FEM models [SSB13], which receive as input values the computed contact forces. Since the point coordinates of the point cloud are anchored in the FEM model, their deformed positions can be obtained straightforwardly. In the case of the distance fields, a low resolution deformable point cloud is embedded during data structure generation. The deformed point coordinates of this auxiliary point cloud are used to locally update the values of the distance field computed for the resting configuration. A similar approach is followed by Chan et al. [CBS13], in this case with intensity fields embedded in FEM tetrahedral meshes. In order to obtain the field values of the deformed environment, first, the barycentric coordinates of the queried point in the deformed tetrahedron where it lies are computed. Substituting these coordinates in the rest configuration leads to the equivalent material point to query in the (undeformed) intensity field.

Finally, related to but beyond deformation, some simulations consider object **topology change** in scenarios that cover *material subtraction* or *cutting*. These are usual in medical simulations, in which tissue dissection is common, either with organs or bones. In this respect, Courtecuisse et al. [CJA⁺10] presented a realtime haptic simulation of soft tissue manipulations, including cutting. The method uses co-rotational FEM models that handle large displacements and geometrical non-linearities. When a tissue is cut, its mesh is modified, as well as the FEM model and the matrix elements in the equations of motion. To that purpose, the deformable model is meshed into set of volume elements which are connected together by a topological map. The state and model matrices are

expanded with element subtraction and subdivision operations, and the efficient update of inverse matrices necessary for stepping forward the dynamic state is tackled using the Sherman-Morrison method, among others. On the other hand, Wang et al. [WCW⁺12] presented an impulse-based haptic rendering framework for the specific scenario of dental bone-burring with a rapidly rotating spherical spindle. The environment (e. g., tooth or bone) is represented with a mesh, whereas the tool (i. e., the spherical burring spindle) is modeled with points on its cutting edges. Rays are shot from the tool center to the points on the edges to detect point and triangle pairs which are colliding. The mesh is reconstructed replacing all vertices of burred triangles to be on the boundary of the spherical spindle.

However, bone-burring scenarios have been handled more commonly using voxelmaps, due to their efficient data structure update capabilities and minimal to non-existent topology changes upon material subtraction. Additionally, this way, patient-specific data structures can be obtained from Magnetic Resonance Imaging (MRI) Computer Tomography (CT) models. Zheng et al. [ZLSWF13] presented a training simulator for dental surgeries using a re-implementation of the VPS for GPUs. A density field embedded in a voxelmap is created from CTs for the teeth, recreating different tissue hardness values. Since drilling is simulated, the field values change during the interaction in the burred regions, and the visualization model is constantly updated using the marching cubes technique [LC87]. Specific spindle tools are supported, placing on their cutting edges the points used for collision and force calculation (i. e., pointshell points). The rotational speed and angle of incidence of the spindle are considered in the force and torque computation, respectively, improving training realism. Kim and Park [KP09] presented a similar penalty-based approach, but using distance fields for the tool and intensity fields and point clouds for the teeth. The authors additionally defined a bone removal rate function to gradually subtract tissue elements on contact.

Other works have focused on proper force rendering methods in bone-burring scenarios using volume-based data structures. Agus et al. [AGG⁺03] derived an analytical contact force and bone erosion model based on the Hertz's elastic contact theory [Her96]; geometric and elastic parameters are taken into account and the model is discretized for voxelmaps. Wu et al. [WYW⁺09] discovered and applied a linear relationship model for resistance force and forward velocity.

2.4 Summary, Conclusions, and Perspectives

This chapter provides an answer to three background questions to be clarified before designing any virtual reality environment enabling **haptic manipulations**:

- (i) how does the *human haptic sensory system* work and which are its requirements and limitations?
- (ii) which are the components of a *virtual reality system with haptic feedback*, how are they inter-related and which effects do they have on the user?
- (iii) which collision and force computation methods for *haptic rendering* have been proposed, taking into account the *human haptic sensory system* (i) and matching the technological puzzle of *virtual reality systems* (ii)?

All in all, the answers try to give a comprehensive picture of the state-of-the-art covering both technological and human factors.

Section 2.1 deals with the first question, describing physiological, neurological, and psychophysical aspects in the processing of kinesthetic and tactile sensations. Taking as basis the properties of mechanoreceptors and processing structures in the sensory-motor cortex, the cornerstones for designing an efficient and realistic haptic simulation are higher level elements like (i) the perceivable *frequency* resolution, (ii) the *JNDs* related to different dynamic magnitudes (e.g., forces, stiffness, etc.), and (iii) the *multimodal* contact percept formation.

Section 2.2 concentrates on the second question and describes in detail the components of the system picture in Figure 1.1 from Chapter 1 (Introduction). Beyond the technical requirements for aural, visual, and haptic modalities, human factors such as cybersickness, immersion, and presence are discussed. Additionally, the motion computation of virtual objects in the scene is briefly explored, as well as interaction techniques and the recently flourished and multiplied interaction devices, covering most types of interfaces. In particular, especial attention is set on *haptic devices*, which are key for properly providing synthetic haptic information. Their workspace, force capabilities, and actuated DoFs define the type of manipulation the user is able to carry out, and their dynamic range of impedances affects significantly the quality of contact perception; with respect to this last point, stability issues are discussed, which are mainly governed by (i) the contact stiffness boundary specific for each system and (ii) the 1 kHz update rate convention for pose-force signals. Additionally, related applications and fields like telerobotics are introduced. Understanding the interplay of all these system components is essential to build an effective virtual reality environment with haptic feedback, as the one reported in Chapter 5.

Finally, Section 2.3 analyzes the third question and provides a thorough survey on haptic rendering methods. The section elaborates the method property classification done in Table 2.1 and Table 2.2, which yield an accurate snapshot of the evolution of collision and force computation since its early days to date. Object representation is

exposed in detail, since efficient data structures contain pre-computed collision and force information in their definition. Similarly, different collision and force computation outputs are also precisely described, as they are the basis principles on which methods are based on. Beyond these definitions, most common collision computation methods and acceleration strategies used in practice are reported, considering the variations and improvements along the years. Regarding force computation, all three major rendering paradigms are discussed, as well as the effects of different DoFs and the use of virtual coupling. The section finishes with notes on fidelity enhancements (e.g., friction, initial contact transients) and a brief overview on how deformation is handled in haptic interactions. Altogether, literature suggests that collision detection is the bottleneck in haptic rendering, and that it can be decoupled from force and deformation computation. Additionally, these points come to light: *(i)* implicit representations oftentimes yield accurate information very fast, *(ii)* wrapped BHVs for discrete sets of elements significantly accelerate computations and enable time-critical LoD processing, and *(iii)* the high spatio-temporal coherence of the collision detection problem should be exploited.

These insights are the starting point for the core technical and scientific contributions from Chapter 3 and Chapter 4, in which novel collision, proximity, and force computation methods are presented. The compilation of these algorithms constitutes a missing piece in the literature for the efficient and robust solutions provided and their unified and self-contained character. With them, rigid objects can be processed for distance or collision independently of their geometrical complexity faster than at 1 kHz velocities, and issues such as time-critical queries, constrained motion, realistic stiff contacts, the tunnelling effect, or large multibody environments are addressed and solved to the core. Furthermore, these techniques are integrated into several simulation and robotics applications in Chapter 5, which validates their versatility.

Lastly, note that Chapter 6 revisits the concepts evolved in all previous chapters, including this one. Indeed, the major elements of a haptic simulation system are evaluated, focusing, in particular, on the parameters of the haptic device and the developed haptic rendering methods.

Chapter 3

Collision and Proximity Queries

This chapter presents the first and most fundamental method used in the rest of the work: a distance and collision computation algorithm able to deliver closest features and six-DoF collision forces between complex rigid geometries at 1 kHz (or faster).

The presented collision detection and force computation algorithm is based on the Voxelman-Pointshell (VPS) algorithm [MPT99]. That approach uses *voxelmaps* and *pointshells* as data structures to compute penalty-based six-DoF collision forces between rigid bodies. Within this work, all offline and online processes were re-defined and implemented from the scratch, improving several key elements; this chapter presents (*i*) the definition, properties, and generation methods of the aforementioned basic data structures out of polygonal models, (*ii*) how these data structures are optimized for faster and more accurate online collision queries, (*iii*) a unified proximity and collision computation algorithm able to perform at 1 kHz or in time-budgeted conditions, and (*iv*) benchmarking experiments that account for the performance of different implemented methods and the global algorithm.

This chapter uses parts from the following peer-reviewed publications written by the author of this work: [SHPH08], [SH13], [SSeS14a], and [SHH⁺15].

3.1 Introduction

As explained in Chapter 2, penalty-based collision computation algorithms use object inter-penetration to compute repulsion forces. Those repulsion forces can be of three-DoF, i. e., the user interacts with a point-probe in a virtual environment and feels only force vectors through the haptic interface, and of six-DoF, i. e., a more complex object-

probe is used and both forces and torques (i. e., wrenches) are displayed to the user. Penalty-based methods are known for being fast, robust and easier to implement than other paradigms, and probably the Voxelman Pointshell (VPS) algorithm is one of the most referenced of them.

The Voxelman-Pointshell (VPS) algorithm was originally published in 1999 by McNeely et al. [MPT99], and it has been re-implemented and improved several times ever since. In the original paper, the authors defined *voxmaps* or *voxelmaps* to be regular 3D grids consisting of voxel elements which contained four basic layer values: inner, surface, proximity, and outer, depending on the voxel position with respect to the polygons of the original model in the grid. Voxelman represented usually large and complex *environments* (e. g., inner aircraft geometries) with a voxel edge length of around 5 mm. The user could interact with that voxelized environment using smaller *probes* modeled as *pointshells*; these data structures were basically point clouds (around 500 points) constituted of the surface voxel centers with their respective inwards pointing normals, representing the probe. In order to compute the penalty forces of a time step, the penetration of points lying in surface or inner voxels was measured along the normal of the point; these unitary normal vectors scaled with the computed point penetration and summed up yielded the total penalty force. The method could perform at near-to-constant computation frequencies (achieved sweeping the same total amount of probe points each cycle), which were as high as 1 kHz if the number of points was kept under a threshold. Additionally, it was (and is) independent of the number of triangles or non-convex features, since these have no effect in the discrete domains formed by voxels and points.

3.1.1 Related Work

Renz et al. [RPP⁺01] improved pointshell models by projecting the points on the polygonal surface, which leads to smoother forces. Additionally, they presented a design framework for virtual coupling [CSB95] which eased the configuration of different haptic interfaces. Virtual coupling was also proposed in the original VPS algorithm as a mean to improve stability (see Section 2.3.3.3): the visualized proxy of the probe was connected to the virtual handle moved by the user through a spring-damper system; the penalty forces were integrated according to the Newton-Euler equations of motion to obtain the pose of the proxy and the force values given by the spring-damper system were the ones displayed to the user. Wan and McNeely [WM03] further researched into the virtual coupling applied to the VPS and presented a quasi-static approximation for computing the proxy pose.

McNeely et al. [MPT06] improved their implementation with layered distance fields and by ignoring redundant polyhedral surface interactions. Spatial accuracy was also

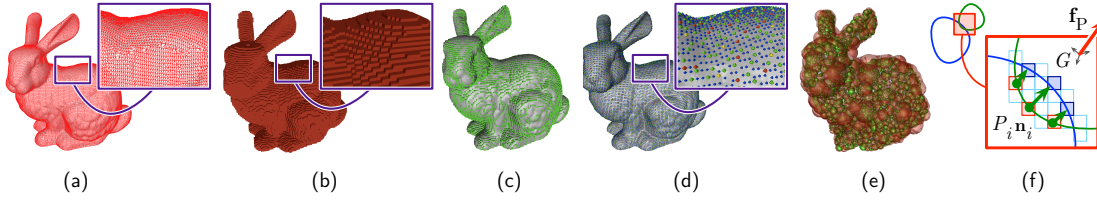


Figure 3.1: Different representations of the Stanford Bunny: (a) Triangle mesh with 35606 vertices; (b) Voxelized representation of the bunny (surface voxels in red); (c) Voxelized representation of the bunny (first inner layer in green). (d) Several point tree levels of the bunny coded with colors; (e) Two successive sphere tree levels of the bunny (the red transparent is the upper level); (f) Basic principle of collision computation with the data structures: The normal vectors \mathbf{n}_i of the pointshell points P_i are weighted by their penetration in the voxelmap to compute the penalty forces \mathbf{f}_p .

improved sacrificing the constant computation time characteristic of the first publication. Barbič and James [BJ08] extended the VPS algorithm for reduced deformable objects using point-sphere hierarchies for faster level-of-detail (LoD) collision detection. Later, Xu and Barbič [XB14] improved the previous re-implementation of the VPS with a robust and automatic approach for generating signed distance fields for arbitrary triangle meshes.

Data structures and force computation methods from the VPS have also been applied to constraint-based haptic rendering approaches. For instance, Chan et al. [CCBS11] applied the *god object* proxy computation method of [ORC07] to point-sampled objects (pointshells) which interacted with volume-embedded surfaces (voxelmaps). Rydén and Chizeck [RC13b] presented a similar approach which additionally supported continuously streamed point clouds.

Algorithms based on or similar to the VPS approach have also been recently employed in several robotics applications, such as mobile manipulation planing using the GPU [HDB⁺14], grasp planing [Her15], assembly planing [NSSB16], or even tracking of complex objects and mechanisms making use of the GPU [SNF14].

3.1.2 Contributions

This work puts the basis for a completely new re-implementation of the VPS principles. New point-sphere hierarchies (pointshells) and signed distance fields (voxelmaps) are defined and generated from arbitrary polygonal models, shown in Figure 3.1.

In a similar fashion as in [BJ08], voxelmaps can contain floating point distance fields in this work. However, a mixed data structure is implemented which supports approximative layer values, real distances which can be locally interpolated in contact areas, and closest points on surface.

Similarly to [BJ08], point-sphere hierarchies are implemented on top of the plain *point-soup*. However, a down-top building approach starting with the highest point sampling resolution where points are uniformly distributed is followed. Additionally, point clusters are bounded with minimal enclosing spheres [FG04], in contrast to the approach in [BJ08], where sphere centers are located on the object’s surface. Thanks to the sphere trees presented in this work, it is possible to quickly and accurately recognize likely colliding areas. Moreover, the point tree enables LoD traversal of the surfaces, which makes possible to provide with a fair contact manifold on an established time budget, even for extremely non-convex geometries.

Once data structures are generated offline in few seconds, the presented collision computation method takes place traversing the point-sphere hierarchy in a breadth-first fashion, similarly as in [BJ08]. Since, all necessary geometry properties and pre-computable values are encoded in a LoD manner in the data structures, a full traverse can deliver contact data in 1 ms.

Altogether, the contributions are presented as follows:

- Section 3.2 explains how the aforementioned voxelmap and pointshell data structures are generated from polygonal models. All methods necessary for fast generation (on the CPU) of signed distance fields embedded in voxelized structures and point-sphere trees that sample the object in a multi-resolution manner are discussed. Additionally, the different functions and properties of these data structures are formally derived and illustrated.
- Section 3.3 presents the unified proximity and collision computation algorithm. The algorithm provides the closest features (points), six-DoF penalty forces, and contact manifolds or colliding point sets. For each arbitrary geometry pair, the offline generated voxelmap and pointshell models are used. The method achieves computation frequencies of 1 kHz and can perform in time-critical conditions gradually improving the quality of the response depending on the time budget.
- Section 3.4 shows the performance of the algorithm with meaningful and reproducible experiments. Methods related to different LoDs are compared between each other and against ground truth or expected values.

As concluded in Section 3.5, such a versatile collision computation framework is suitable for many applications beyond haptic simulations which require the handling of arbitrary geometries.

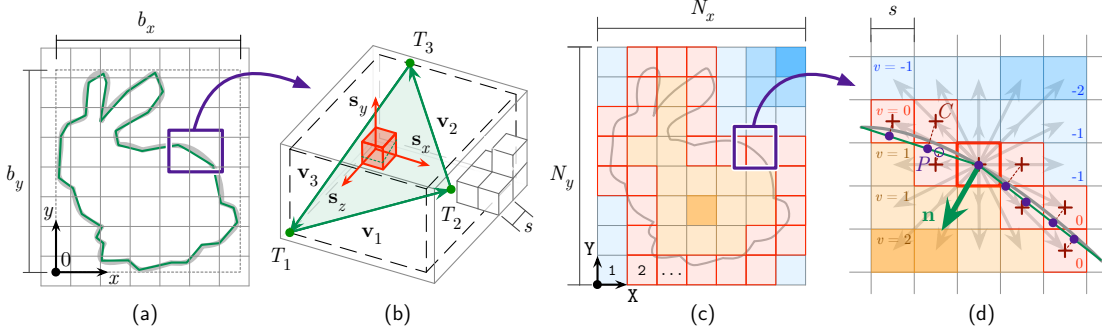


Figure 3.2: Generation of the basic primitives: (a) A triangle mesh of the Stanford Bunny in an empty voxelized grid (simplified coarse resolution). (b) A triangle checked for collision against the voxels in its bounding box. (c) After detecting surface voxels (in red), outer and inner layers are computed with the Scanline algorithm. Surface voxels have voxel value $v = 0$, each n -th inner layer $v = n$, and outer layer $v = -n$, respectively. (d) Each surface voxel center C_i is projected onto the mesh to obtain pointshell points (P_i). The normal vector \mathbf{n}_i of a point P_i is the normalized gradient of the voxel value in the voxel of the point: $\mathbf{n}_i \sim \nabla v(P_i)$.

3.2 Data Structures

This section introduces the definition and computation of the basic primitives (Section 3.2.1), the properties of the data structures and the basic algorithm (Section 3.2.2), as well as the definition and computation of the enhanced data structures (Section B and Section B).

3.2.1 Generation of Basic Primitives

Figure 3.2 illustrates the most important stages for generating the basic primitives. In the following, first the computation of plain voxelmaps from polygonal models is introduced, and second, the creation of the plain or basic pointshells. The section finishes discussing some relevant implementation issues.

3.2.1.1 Voxelized Structures (Voxelmaps)

The first step for creating the voxelmap consists in placing the triangle mesh \mathcal{T} in the voxelmap \mathbf{V} , which represents a 3D regular grid yet to be filled (Figure 3.2(a)). All voxels store a voxel value $v \in \mathbb{N}$ (initialized with $v = -1$), and they are contained in the set

$$\mathcal{V} = \{v_1, \dots, v_i, \dots, v_{N_V}\} = \{v_i\}_{i=1}^{N_V}, \quad (3.1)$$

being $N_{\mathcal{V}}$ the total amount of voxels or voxel values v . This number can be factorized in the number of voxels or cells per axis:

$$N_{\mathcal{V}} = N_x N_y N_z. \quad (3.2)$$

The bounding box of the voxelmap $\mathbf{b}_{\mathcal{V}}$ is determined by the bounding box of the mesh $\mathbf{b}_{\mathcal{T}} = (b_x, b_y, b_z)^{\top}$ and the voxel edge length or voxel size s is chosen by the user; usually, the value of s is such that the largest bounding box axis i of the triangle mesh \mathcal{T} is divided in a maximum number of $N_i \leq 300$ cells, given the memory limits of current computers:

$$\max\{N_x, N_y, N_z\} = \left\lceil \frac{\max\{b_x, b_y, b_z\}}{s} \right\rceil \leq 300. \quad (3.3)$$

After this parametrization and the creation of all sets and values with the correct sizes, all triangles of the mesh are visited and the voxels which intersect with them are marked with the surface voxel value $v = 0$ (Figure 3.2(b)). This process requires (i) selecting candidate voxels within the bounding box of each triangle that might collide with the triangle and (ii) performing collision checks between voxels (cubes) and triangles. Selection is performed by observing the projection of the triangle on each of the orthogonal planes of the coordinate axes. If the voxel is clearly above or below the projections, its collision is automatically discarded for the sake of performance. Otherwise, an accurate intersection check must be carried out.

Those collision checks between candidate voxels and the triangles are computed using the Separating Axis Theorem (SAT), already introduced in Section 2.3.2.2. To recall, the SAT states that two convex objects collide with each other if and only if there is an axis on which their projections overlap. If no such axis exists, both objects are disjoint. In 3D space, the axes to be checked for collision are the normal vectors of the faces and the cross products between each pair of edges. As displayed in Figure 3.2 (b), for each triangle composed of vertices $\{T_1, T_2, T_3\}$, its edge and normal vectors can be defined as:

$$\mathbf{v}_1 = T_2 - T_1, \quad \mathbf{v}_2 = T_3 - T_2, \quad \mathbf{v}_3 = T_1 - T_3, \quad \mathbf{n}_T = \frac{\mathbf{v}_1 \times \mathbf{v}_2}{\|\mathbf{v}_1 \times \mathbf{v}_2\|}. \quad (3.4)$$

On the other hand, the voxel is determined with its center C (considered to be the origin during each check) and its voxel size s . Altogether, there are 13 axes to be checked for overlap [AM01], which can be classified into three groups:

1. The three *coordinate axes* $\{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3\}$, representing the normal vectors to the faces of the voxel. The three vertices of the triangle are projected on each axis to check whether the projection of the triangle overlaps with the projection of the voxel, which is a region of width s centered in the origin.

2. The *normal vector* of the triangle \mathbf{n}_T . The vertices of the voxel that define the vector which is aligned with \mathbf{n}_T are checked for their relative position against the plane of the triangle.
3. The nine \mathbf{p}_{ij} axes that result from performing the *cross product between the edges of the voxel and the triangle*: $\mathbf{p}_{ij} = \mathbf{s}_i \times \mathbf{v}_j, \forall i, j \in \{1, 2, 3\}$. A detailed explanation of the procedure is given in [Sag08].

A triangle-voxel pair is not colliding if and only if all the tests in the 13 axes yield no overlap. As soon as an overlap between a projected triangle-voxel pair on one of the 13 axes is detected, the voxel is marked as surface-voxel.

After having traversed all the objects detecting each surface-voxel, the inner and outer parts of the model are recognized on the voxmap using the Scanline [KS87] algorithm extended to 3D [Ott05]. The algorithm works with a Last In First Out (LIFO) stack that speeds up the filling process and starts at a corner-voxel of the virtual model. Afterwards, layers are added to the voxelized surface: the first outer layer is formed by voxels with value $v = -1$, whereas the first inner layer is created with voxels of value $v = 1$. The absolute voxel value increases linearly according to the number of layers away from the surface, as shown in Figure 3.2(c) and (d). The voxel connectivity can be chosen to be 6-connected (default), 18-connected, or 26-connected. Note, additionally, that the number of inner and outer layers can be selected; in that case, the initial computation of the voxmap bounding volume \mathbf{b}_V must be accordingly modified at the beginning of the computation.

An important feature of the voxmaps consists in the fast access they provide to specific space regions, and therefore, also concrete geometry parts. This property is essential for creating pointshell structures or enhanced signed distance fields, and even for online collision and proximity queries. Targeted voxmap regions can be scanned by accessing the neighborhood \mathcal{N}_V given seed or voxel of interest. In practice, this neighborhood consists in a cube of width of $2n + 1$ voxels, with n being the number of layers outwards from that first seed voxel. This results in a family of n neighborhoods, $\mathcal{N}_{V,n}$, therefore, composed of $(2n + 1)^3 - 1$ voxels each. Usually $\mathcal{N}_{V,1}$ and $\mathcal{N}_{V,2}$ are used, depending on the desired accuracy. For instance, if P is a point in space which lies in a voxel with center C and discrete voxmap coordinates $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$, the neighborhood $\mathcal{N}_{V,n=1}(P) = \mathcal{N}_{V,n=1}(C)$ is the set of voxels with the following discrete coordinates:

$$\begin{aligned}
\mathcal{N}_{\mathcal{V},n=1}(P) : \{ & (\mathbf{X} - 1, \mathbf{Y} - 1, \mathbf{Z} - 1), (\mathbf{X} - 1, \mathbf{Y} - 1, \mathbf{Z}), (\mathbf{X} - 1, \mathbf{Y} - 1, \mathbf{Z} + 1), \\
& (\mathbf{X} - 1, \mathbf{Y}, \mathbf{Z} - 1), (\mathbf{X} - 1, \mathbf{Y}, \mathbf{Z}), (\mathbf{X} - 1, \mathbf{Y}, \mathbf{Z} + 1), \\
& (\mathbf{X} - 1, \mathbf{Y} + 1, \mathbf{Z} - 1), (\mathbf{X} - 1, \mathbf{Y} + 1, \mathbf{Z}), (\mathbf{X} - 1, \mathbf{Y} + 1, \mathbf{Z} + 1), \\
& (\mathbf{X}, \mathbf{Y} - 1, \mathbf{Z} - 1), (\mathbf{X}, \mathbf{Y} - 1, \mathbf{Z}), (\mathbf{X}, \mathbf{Y} - 1, \mathbf{Z} + 1), \\
& (\mathbf{X}, \mathbf{Y}, \mathbf{Z} - 1), (\mathbf{X}, \mathbf{Y}, \mathbf{Z} + 1), \\
& (\mathbf{X}, \mathbf{Y} + 1, \mathbf{Z} - 1), (\mathbf{X}, \mathbf{Y} + 1, \mathbf{Z}), (\mathbf{X}, \mathbf{Y} + 1, \mathbf{Z} + 1), \\
& (\mathbf{X} + 1, \mathbf{Y} - 1, \mathbf{Z} - 1), (\mathbf{X} + 1, \mathbf{Y} - 1, \mathbf{Z}), (\mathbf{X} + 1, \mathbf{Y} - 1, \mathbf{Z} + 1), \\
& (\mathbf{X} + 1, \mathbf{Y}, \mathbf{Z} - 1), (\mathbf{X} + 1, \mathbf{Y}, \mathbf{Z}), (\mathbf{X} + 1, \mathbf{Y}, \mathbf{Z} + 1), \\
& (\mathbf{X} + 1, \mathbf{Y} + 1, \mathbf{Z} - 1), (\mathbf{X} + 1, \mathbf{Y} + 1, \mathbf{Z}), (\mathbf{X} + 1, \mathbf{Y} + 1, \mathbf{Z} + 1) \}.
\end{aligned} \tag{3.5}$$

The generation complexity of the primitive voxelmap is worst case $\mathcal{O}(n + m)$, with n being the number voxels and m the number of triangles. The factor m is negligible in regular situations, but it must be considered in case the object has large amounts of triangles and the chosen resolution is very low (e.g., a unique voxel containing thousands of triangles). Using a resolution of about $s = 5$ mm, usual generation times range from few seconds for desktop-sized objects to few minutes for car-sized objects. The data structures can be saved into files for later use; file size ranges from few kilobytes for desktop-sized objects and few hundreds of megabytes for car-sized objects. Exemplary computation time, file size and snapshots with varied resolutions are provided in [SHPH08] and in Figure 3.16.

3.2.1.2 Point Clouds (Pointshells)

The point cloud or pointshell \mathbf{P} of a triangle mesh \mathcal{T} is obtained from its previously generated voxelmap \mathbf{V} by projecting the surface voxel centers (C) on their respective (closest) triangles. Additionally, normal vectors pointing inwards the object are computed. The primitive pointshell consists basically in the set \mathcal{P} :

$$\mathcal{P} = \{(P, \mathbf{n})_1, \dots, (P, \mathbf{n})_i, \dots, (P, \mathbf{n})_{N_{\mathcal{P}}}\} = \{(P, \mathbf{n})_i\}_{i=1}^{N_{\mathcal{P}}}, \tag{3.6}$$

with $(P, \mathbf{n}) \in \mathbb{R}^{3+3}$ being the pair point-normal and $N_{\mathcal{P}}$ the total amount of pairs or pointshell points.

Figure 3.2(d) illustrates the steps necessary for generating such a set \mathcal{P} . Similarly as in the voxelmap generation, all triangles are traversed, visiting all surface voxels within the triangle bounding box. Given a voxel center $C \in \mathbb{R}^3$, its projection P onto the triangle is computed with quadratic programming, as explained below. There exists a closed-form analytical solution to it and the approach has been shown to be up to 37% faster than using algebraic projections [Sag08].

In order to obtain the analytical expression of the projected point P , first, a point \mathbf{r}_T on the triangle with vertices $\{T_1, T_2, T_3\}$ is expressed in the local triangle coordinates (u_1, u_2) [Ebe99]:

$$\begin{aligned} \mathbf{r}_T(u_1, u_2) &= \mathbf{r}_0 + u_1\mathbf{r}_1 + u_2\mathbf{r}_2 = T_1 + u_1(T_2 - T_1) + u_2(T_3 - T_1) \\ \text{subject to } &\begin{cases} 0 \leq u_1 \leq 1 \\ 0 \leq u_2 \leq 1 \\ u_1 + u_2 \leq 1. \end{cases} \end{aligned} \quad (3.7)$$

The quadratic function that defines the squared distance between C and \mathbf{r}_T is:

$$q(u_1, u_2) = \|\mathbf{r}_T(u_1, u_2) - C\|^2. \quad (3.8)$$

The value of q is minimum on the projection point P , which leads to the formulation of the quadratic programming problem:

$$\begin{aligned} \min q(u_1, u_2) &= a_1u_1^2 + 2a_2u_1u_2 + a_3u_2^2 + 2a_4u_1 + 2a_5u_2 + a_6 \\ \text{subject to } &\begin{cases} g_1(u_1, u_2) = u_1 \geq 0 \\ g_2(u_1, u_2) = u_2 \geq 0 \\ g_3(u_1, u_2) = 1 - u_1 - u_2 \geq 0, \end{cases} \end{aligned} \quad (3.9)$$

where all a_i values are easily obtainable scalars:

$$\begin{aligned} a_1 &= \|\mathbf{r}_1\|^2, & a_2 &= \mathbf{r}_1^\top \mathbf{r}_2, & a_3 &= \|\mathbf{r}_2\|^2, \\ a_4 &= \mathbf{r}_1^\top (\mathbf{r}_0 - C), & a_5 &= \mathbf{r}_2^\top (\mathbf{r}_0 - C), & a_6 &= \|\mathbf{r}_0 - C\|^2. \end{aligned} \quad (3.10)$$

Note that only u_1 and u_2 are unknown in (3.10). Their solution values (u_1^*, u_2^*) yield the projection point $P = \mathbf{r}_T(u_1^*, u_2^*)$ and its distance to the voxel center C : $\sqrt{q(u_1^*, u_2^*)}$. In order to obtain them, the Karush-Kuhn-Tucker (KKT) conditions are applied to (3.10). This results in the following nonlinear system with five equations and five unknowns (variables u_1, u_2 and the Lagrange multipliers $\lambda_i, \forall i \in \{1, 2, 3\}$):

$$\begin{cases} \nabla q(u_1, u_2) - \sum_i \lambda_i \nabla g_i(u_1, u_2) = 0 \\ \lambda_i g_i = 0. \end{cases} \quad (3.11)$$

This system in (3.11) yields the set of seven solutions shown in Table 3.1. One corresponds to a point on the triangle and the other six to points on the boundary: three on each of the edges and three on each of the vertices. The real solution from the set of seven is the one which fulfills the conditions in (3.7) and additionally leads to real $\lambda_i \geq 0$. This projection method is a backbone function or tool used in several data structure improvements explained in later sections.

Table 3.1: List of seven solutions to the system in (3.11). Each of them corresponds to one point inside the triangle or at the boundary (edges or vertices). The values of a_i are parameters defined using the vertices of the triangle (see (3.10)). With this set, a general closed-form solution is given to the problem of projecting any point on any triangle (parametrized as in (3.7)).

	u_1	u_2	λ_1	λ_2	λ_3
\mathbf{r}_T	$\frac{a_2 a_5 - a_3 a_4}{a_3 a_1 - a_2^2}$	$\frac{a_2 a_4 - a_5 a_1}{a_3 a_1 - a_2^2}$	0	0	0
\mathbf{v}_1	$-\frac{a_4}{a_1}$	0	0	$\frac{2(a_5 a_1 - a_2 a_4)}{a_1}$	0
\mathbf{v}_2	$\frac{a_4 - a_3 - a_5 + a_2}{2a_2 - a_1 - a_3}$	$\frac{a_4 - a_5 - a_2 + a_1}{-2a_2 + a_1 + a_3}$	0	0	$\frac{2(a_2(a_4 + a_5 + a_2) - a_3(a_1 + a_4) - a_5 a_1)}{-2a_2 + a_1 + a_3}$
\mathbf{v}_3	0	$-\frac{a_5}{a_3}$	$\frac{2(a_3 a_4 - a_2 a_5)}{a_3}$	0	0
T_1	0	0	$2a_4$	$2a_5$	0
T_2	1	0	0	$2(a_2 + a_5 - a_1 - a_4)$	$-2(a_1 + a_4)$
T_3	0	1	$2(a_2 + a_4 - a_3 - a_5)$	0	$2(a_5 - a_3)$

The pointshell generation is performed in two phases. In the first one, all the projected points are computed using the presented optimization method, but only the projected points that lie inside the triangle are stored into the structure of the pointshell, tagging the surface voxels to which they belong to as *projected*. The rest of the boundary-points are stored separately, indexing also the voxel from which they were projected. Adding a point P_i to the pointshell \mathbf{P} requires also checking that the new point is at least half a voxel size ($s/2$) away from any other added point. To that purpose, a support structure which matches surface voxels and projected points is used. Therefore, instead of checking each candidate point against all other added points (quadratic complexity), only the added points linked to the surface voxels surrounding the candidate point are checked.

In the second phase, all the separately stored boundary points are traversed checking whether their origin voxels were already tagged as *projected*. If not, the boundary point is added to the pointshell, tagging its origin voxel as *projected*; otherwise, the point is rejected.

After generating pointshell points, the next step consists in computing the inwards pointing normal vector \mathbf{n} associated to each one. This is in part motivated by the fact that normal vectors generated by CAD programs point sometimes in the wrong direction.

The normal vectors are obtained analyzing the neighborhood \mathcal{N}_v of the pointshell point P_i in the voxelmap (see (3.5)). Setting the origin in the voxel where the point is located, all the vectors that go from this origin voxel center C_i to each of the neighbor voxel centers C_j are weighted by their voxel value increment ($v_i \rightarrow v_j$) and summed up to obtain the gradient of the voxel value $\nabla v = \nabla v(C_i)$. Recall that the voxelmap

has inner (positive) and outer (negative) layers which increase their absolute value as they get away from the surface layer. Since P_i lies within the voxel with center C_i , its gradient is approximated with the gradient of the voxel center, i. e., $\nabla v(P_i) \simeq \nabla v(C_i)$. The normal vector \mathbf{n}_i of the pointshell point P_i is the normalized value of the gradient $\nabla v(P_i)$:

$$\begin{aligned} \nabla v(P_i) \simeq \nabla v(C_i) = \nabla v_i &= \sum_{v_j \in \mathcal{N}_V(P_i)} \Delta v_{ij} \Delta C_{ij} = \sum_{v_j \in \mathcal{N}_V(P_i)} (v_j - v_i)(C_j - C_i), \\ \mathbf{n}(P_i) \simeq \mathbf{n}(C_i) = \mathbf{n}_i &= \frac{\nabla v(P_i)}{\|\nabla v(P_i)\|}. \end{aligned} \tag{3.12}$$

The computation of the gradient and the normal summarized in Section 3.12 is another backbone function or tool used in other steps during the enhancements of the data structures.

After all normal vectors have been computed, the pointshell \mathbf{P} is complete. Its associated voxelmap and the support structure that matches surface voxels and projected points are destroyed. Therefore, unlike the voxelmap, such a resulting point cloud is at this stage an unstructured *point soup* without any topological information that relates neighboring points.

As in the case of the primitive voxelmap, the generation complexity is worst case $\mathcal{O}(n + m)$, with n being the number of voxels and m the number of triangles. Usual generation times last about few seconds for desktop-sized objects (e. g., with $s = 5$ mm and around 8000 pointshell points). However, the time required for computing a pointshell will always be longer than the one of a voxelmap alone, since a voxelized structure has to be computed prior to the point cloud. Given that the number of points strongly influences the computation time during online collision queries, only smaller (desktop-size) objects are represented with pointshells, leaving voxelmaps for larger ones. Additionally, while primitive voxelmaps can represent arbitrarily complex objects (their memory footprint being the limit), pointshells can model correctly only watertight voxelmaps. Even if an object is non-watertight, it might lead to a watertight voxelmap in case the voxel size s is bigger than the holes of the object. However, if the geometry is a thin shell, it has no inner regions (i. e., all $v \leq 0$), thus, no correct inwards pointing normal vectors can be computed at the surface as defined in (3.12). Therefore, non-watertight thin shells cannot be modeled as pointshells, but only as voxelmaps. This issue is revisited in Chapter 4.

As with the voxelmaps, pointshell data structures can be saved into files for later use; file size ranges usually from few kilobytes to few (less than ten) megabytes – ASCII

encoding is used, without compression, for human readability. Exemplary computation time, file size, and snapshots with varied resolutions are provided in [SHPH08] and in Figure 3.16. Finally, note that Appendix B reports relevant implementation and performance issues related to the primitive data structures introduced in this section.

3.2.2 Properties and Limitations

In the classical VPS [MPT99] algorithm, the penalty collision forces are computed as spring-like forces following Hooke’s law. With the data structures and functions explained so far, this can be approximated by

$$\mathbf{f}_P = k_P \sum_{v(P_i) > 0} v(P_i) \mathbf{n}_i, \quad (3.13)$$

with k_P being the penalty stiffness equal for all points. In words, all the normal vectors (force directions) of the colliding points in the pointshell ($v(P_i) > 0$) are weighted by their penetration $v(P_i)$ and summed up to yield the total penalty force \mathbf{f}_P . As explained in the previous section, the access to the voxel value v is $\mathcal{O}(1)$ and, subsequently, the collision force query in (3.13) has $\mathcal{O}(n)$ complexity, being n the number of checked points. That leads to a central and general characteristic of the VPS:

Voxelmaps can model arbitrarily complex objects with the maximum possible resolution (given memory limits) without affecting the computation time of collision queries; on the other hand, the number of points of the point clouds which are checked for collision increases linearly the required computation time.

Clearly, controlling the number of points checked while keeping the resolution as high as possible is essential for improving time performance. This issue is addressed in Section B.

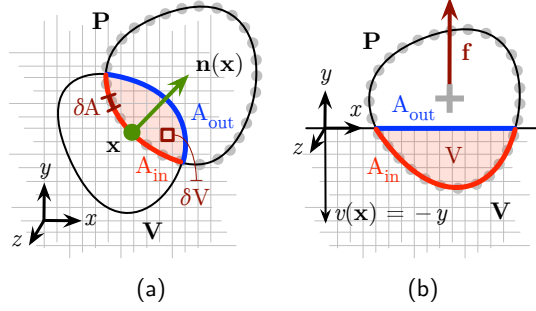
The classical VPS and the primitive data structures presented so far have other specific properties that can be exploited or limitations that should be taken into account. All these are presented in the following subsections.

3.2.2.1 Properties

Two major properties can be defined:

- P1** If the number of elements (points or voxels) for a given resolution (voxel size s) is known, the number of elements for any resolution can be estimated.
- P2** The physical meaning of the penalty collision forces is related to the overlapping volume.

Figure 3.3: Volumetric nature of the forces: (a) A pointshell \mathbf{P} is penetrating into a voxelmap \mathbf{V} . The intersection volume V is enclosed within the surfaces A_{in} (red, surface of the pointshell) and A_{out} (blue, surface of the voxelmap). If the resolution of the models is high, i.e., $s \rightarrow 0$, pointshell points can be represented as the position variable \mathbf{x} constrained to A_{in} and collision forces can be ultimately computed as a volume integral. (b) In the special case where A_{out} is planar, forces are directly proportional to the intersection volume (with $s \rightarrow 0$), as in the case of buoyancy forces.



The first property **P1** is derived from the principle that the real object area A and volume V are invariant for any discretization. Let \mathcal{I} be the subset of \mathcal{V} related to the inner or solid voxels ($v \geq 0$):

$$\mathcal{I} = \mathcal{V}_{|v \geq 0}, \quad \mathcal{I} \subseteq \mathcal{V}. \quad (3.14)$$

The number of points in \mathcal{P} ($N_{\mathcal{P}}$) and the number of solid voxels in \mathcal{I} ($N_{\mathcal{I}}$) can be used to approximate the area (\tilde{A}) and the volume (\tilde{V}), respectively, if the corresponding resolution or voxel size s is known:

$$N_{\mathcal{P}} s^2 = \tilde{A}, \quad N_{\mathcal{I}} s^3 = \tilde{V}. \quad (3.15)$$

Assuming that the ratio between the real and approximated area or volume is close to constant, for a pair of different resolutions s_1 and s_2 , it holds:

$$\frac{N_{\mathcal{P},1}}{N_{\mathcal{P},2}} \simeq \frac{s_2^2}{s_1^2}, \quad \frac{N_{\mathcal{I},1}}{N_{\mathcal{I},2}} \simeq \frac{s_2^3}{s_1^3}. \quad (3.16)$$

This is particularly interesting for point clouds, since their number of elements is linearly proportional to the computation time. In this line, if a pointshell with a given resolution s_1 leads to a certain expected time budget Δt_1 , a smaller computation time Δt_2 can be achieved with the resolution s_2 defined as

$$s_2^2 = \frac{N_{\mathcal{P},1}}{N_{\mathcal{P},2}} s_1^2 = \frac{\Delta t_1}{\Delta t_2} s_1^2 \quad \Rightarrow \quad s_2 = \sqrt{\frac{\Delta t_1}{\Delta t_2}} s_1. \quad (3.17)$$

The second property **P2** is illustrated in Figure 3.3. In there, a pointshell \mathbf{P} overlaps with a voxelmap \mathbf{V} giving an intersection volume V bounded with the surfaces A_{in} (red, surface of the pointshell) and A_{out} (blue, surface of the voxelmap). If the resolution is

increased by decreasing the voxel size $s \rightarrow 0$, the force associated to a colliding point \mathbf{x} acting on a surface element δA of the pointshell is

$$\delta \mathbf{f}(\mathbf{x}) = k_P v(\mathbf{x}) \mathbf{n}(\mathbf{x}) \delta A, \quad \mathbf{x} \in A_{\text{in}}. \quad (3.18)$$

As pointed out in [Bar07], the total penalty force can then be defined as a surface integral, which is equivalent to a volume integral after some operations:

$$\mathbf{f}_P = k_P \frac{\int_{A_{\text{in}}} v(\mathbf{x}) \mathbf{n}(\mathbf{x}) \delta A}{\int_{A_{\text{in}}} \delta A} \stackrel{1}{=} k_P \frac{\int_{A_{\text{in}} \cup A_{\text{out}}} v(\mathbf{x}) \mathbf{n}(\mathbf{x}) \delta A}{A_{\text{in}}} \stackrel{2}{=} -k_P \frac{\int_V \nabla v(\mathbf{x}) \delta V}{A_{\text{in}}}. \quad (3.19)$$

In step 1, the surface integral is extended to A_{out} without any effect on the total value, since $v(\mathbf{x}) = 0 \forall \mathbf{x} \in A_{\text{out}}$, being A_{out} the surface of the voxelmap. In step 2, the divergence theorem is applied, revealing the volumetric nature of the penalty forces. As a result, forces can be interpreted as proportional to the sum of the gradients of the distance field $\nabla v(\mathbf{x})$ acting on each overlapping volume element δV .

In the particular case of planar voxelmap contact surfaces displayed in Figure 3.3 (b), contact forces turn out to be proportional to the total overlap volume:

$$\begin{aligned} v(\mathbf{x}) = -y &\Rightarrow \nabla v(\mathbf{x}) = (0, -1, 0)^T \Rightarrow \\ \Rightarrow \mathbf{f}_P = -k_P \frac{\int_V \nabla v(\mathbf{x}) \delta V}{A_{\text{in}}} &= k_P \frac{V}{A_{\text{in}}} (0, 1, 0)^T. \end{aligned} \quad (3.20)$$

Therefore, VPS penalty forces are closely related to the buoyancy forces. It is important to consider this physical meaning of the forces, especially when collisions against thin shells occur, as explained in the next section.

3.2.2.2 Limitations

Figure 3.4 illustrates the two major difficulties the VPS algorithm faces: (i) the aliasing due to discretization and (ii) the tunneling effect, characteristic of thin shells.

Aliasing occurs when continuous analog signals are sampled. In the case of VPS, this sampling refers to the discrete points on the surface of the pointshell and the regular voxel grid of the voxelmap, and it can affect both the magnitude and the direction of the forces, as analyzed in [WSM⁺10]. For lower resolutions (larger voxel size s), forces and, ultimately, geometries become more indistinguishable. Since the pointshell resolution is governed by the time budget, it is reasonable to address the subject through voxelmaps. In this line, the issue of aliasing is tackled in Section B by increasing the accuracy of the distance values provided by the voxelmap. This can be interpreted as a conventional

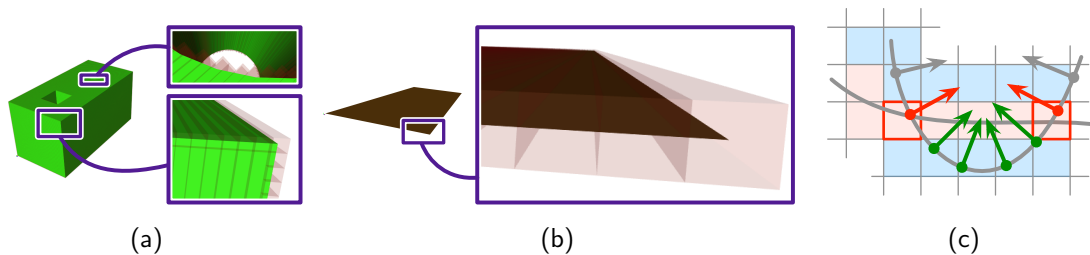


Figure 3.4: Pitfalls of computing collisions with voxelmaps and point clouds: (a) Signed distance and force aliasing can increase when low resolutions are used, especially in round regions. (b) Voxelized thin shell. (c) Tunneling effect or *pop-through* effect characteristic of penalty-based algorithms when thin shells are checked for collision: Since no inner region is distinguished, overlap volumes are formed only with surface voxels; hence, forces tend to be too weak and objects can end up going through (green points).

low pass filter that removes undesired high frequency artifacts caused by the spatial discretization not aligned with the modeled object surfaces.

The tunneling effect, also called *pop-through* effect [ZS95], is named after the quantum tunneling, in which a particle tunnels through a barrier that it should not cross. In collision computation, it occurs when penalty-based algorithms are used to render collision forces against non-watertight thin shells. As displayed in Figure 3.4 (c), no inner regions are distinguished for such thin surfaces. If a point manages to go through the solid surface (green points), it does not contribute to the collision force anymore, as it otherwise would. In other words, taking into account the volumetric force model in (3.20), overlap volumes are formed only with surface voxels of the thin shells; therefore, forces tend to be too weak and objects can end up going through, because the user is not fed with a large enough restriction. This issue is solved in Chapter 4 with a novel constraint-based algorithm for force rendering and proxy simulation.

3.2.3 Enhanced Voxelmaps: Signed Distance Fields

The primitive voxelmap presented in Section 3.2.1 is enhanced for higher accuracy by adding two additional structures on top of \mathcal{V} :

- (i) A set \mathcal{S} (implemented as an array) of N_S surface voxel elements ($v = 0$), in which each surface voxel stores
 - the projected point S of the voxel center C on the object
 - and the three vertices T_1, T_2, T_3 of the projection triangle (see Figure 3.2),

denoted as

$$\mathcal{S} = \{(S, (T_1, T_2, T_3))_{i_S}\}_{i_S=1}^{N_S}. \quad (3.21)$$

(ii) A set \mathcal{W} of $N_{\mathcal{W}} = N_{\mathcal{V}}$ elements or voxels in which each voxel contains:

- a floating point signed distance value w from the voxel center C to the object's closest triangle,
- and the index i_S of the closest surface voxel in the previous structure \mathcal{S} ,

denoted as

$$\mathcal{W} = \{(w, i_S)_{i_{\mathcal{W}}}\}_{i_{\mathcal{W}}=1}^{N_{\mathcal{W}}}. \quad (3.22)$$

This added information increases the precision provided by the primitive voxelmap, yet at the cost of memory requirements, as discussed later in this section. More importantly, the user can choose the resolution (voxel size, s) and the ultimate accuracy (*layer values, floating point distance values*) depending on the application.

The input structures necessary to build the enhanced voxelmap are the primitive voxelmap introduced in Section 3.2.1 and the triangle mesh of the object.

3.2.3.1 Generation of the Structures in the Enhanced Voxelmap

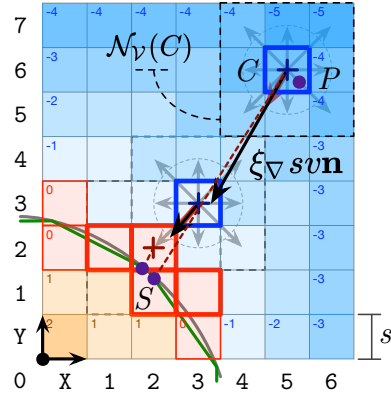
In order to generate the enhanced voxelmaps, first, the surface voxel structure \mathcal{S} is created in a similar fashion as the primitive pointshell: all triangles are traversed and the surface voxels in their bounding boxes are projected employing the methods explained in Section 3.2.1. The sign of the distance in the surface voxels is determined by observing the direction of the vector that points from the voxel center C to the projected point with respect to the gradient of the voxel value (as computed in (3.12)): if the dot product of both is positive, the distance is negative, otherwise it is positive – recall that the gradient or normal vectors point inwards the object.

Second, the floating point distance structure \mathcal{W} is created using \mathcal{S} . All voxels are traversed and their closest surface voxel is computed performing iterative gradient descent in the discrete voxelmap, as illustrated in Figure 3.5. Let C be the voxel center of a point P in an iteration, then the next P' closer point to the surface will be

$$P' = C + \underbrace{\xi_{\nabla} s v(C)}_{\text{step length}} \mathbf{n}(C) = C + \xi_{\nabla} s v(C) \frac{\nabla v(C)}{\|\nabla v(C)\|}, \quad (3.23)$$

where $\xi_{\nabla} = 0.8$ for the first iteration and $\xi_{\nabla} = 1.0$ for the successive ones (values obtained experimentally). This form factor controls the search velocity in the gradient descent.

Figure 3.5: Gradient or steepest descent in the voxelmap: The closest surface voxel of a voxel center C where a point P_i lies is found by moving in the direction of the voxel value gradient in C : $\mathbf{n}(C) = \frac{\nabla v(C)}{\|\nabla v(C)\|}$. The gradient is computed with the voxel values in the neighborhood of C , $\mathcal{N}_v(C)$, as defined in (3.12). The magnitude of the step is given by the voxel size s weighted by σ , being $\xi_\nabla = 0.8$ for the first iteration and $\xi_\nabla = 1.0$ for the successive ones until a voxel with $v = 0$ is found. The distance from C to the triangle stored in the closest surface voxel found is computed as explained in Section 3.2.1. Triangles stored in the neighbor voxels of the closest surface voxel are also checked to minimize the limitations of the gradient descent in discrete coordinates.



The computation finishes as soon as P' lies in a voxel with $v = 0$ (surface voxel), which usually occurs in 2 – 3 iterations. The default neighborhood is of width 3 (i. e., $\mathcal{N}_{v,1}$, see (3.5)). Once the surface voxel is found, the closest surface point S is computed by projecting the original point (or voxel center) on the triangle defined by the vertices stored in the surface voxel, following the optimization method explained in Section 3.2.1.

Depending on the voxel size s (i. e., discretization factor), this heuristic could lead to incorrect surface voxels, particularly as the voxel layer value $v(C)$ increases. In order to reduce that error, all the triangles associated to the surface voxels in the neighborhood of the closest surface voxel are checked, taking the one which yields the smallest distance value.

This gradient descent method is an essential function also integrated in the primitive voxelmap that can be used to deliver surface voxels associated to any point in space during online queries.

The generation complexity is worst case $\mathcal{O}(n + m)$, with n being the number of voxels and m the number of triangles. As in the case of primitive data structures, the factor m is negligible in regular situations, but it must be considered in case the object has large amounts of triangles and the chosen resolution is very small. Using a resolution of around $s = 5$ mm, usual generation times are of about 1 – 3 minutes for desktop-sized objects. The time required for creating an enhanced distance field will always be longer than the one of a primitive pointshell alone. In fact, generating a distance field consists in building (i) a primitive voxelmap, (ii) a structure similar to the pointshell (\mathcal{S}), and, finally, (iii) a field with the same amount of elements as the primitive voxelmap but with more accurate and expensive computations per voxel (\mathcal{W}).

As for the primitive data structures, enhanced voxelmaps can be saved in files for later use. If pre-computed gradient vectors and additional support information are stored per voxel, files sizes easily reach tens of megabytes for regular resolutions ($s = 5$ mm,

desktop-size objects). Exemplary computation time, file size and snapshots with varied resolutions (s) are provided in [SHPH08] and in Figure 3.16. The files size, and more importantly, the resulting memory footprint, are a clear limitation factor when it comes to choosing between primitive layered voxelmaps or floating points distance fields. It is worth to mention that in the latter enhanced voxelmaps only one triangle is registered per voxel. This is not relevant in regular situations (i. e., average triangle area at least bigger than s^2), but it might lead to accuracy losses when the density of triangles per voxel increases.

3.2.3.2 The Signed Distance Voxelmap Function $V(P)$

Given a point and its associated normal vector ($P, \mathbf{n}(P)$), the *signed distance voxelmap function* $V(P)$ is defined to yield during realtime operations the signed distance of the point with respect to the surface encoded in the voxelmap. This signed distance can be the smallest Euclidean *distance to collision* ($V(P) \leq 0$, *non colliding*) or the point *penetration* value ($V(P) > 0$, *colliding*) in the object modeled by the voxelmap. In the latter case, the *penalty* collision force \mathbf{f}_i and torque \mathbf{t}_i associated with the colliding single point P_i (and its normal \mathbf{n}_i) of the pointshell are defined as *spring-like* forces that follow Hooke's law of elasticity:

$$\mathbf{f}_i = k_P V(P_i) \mathbf{n}_i, \quad \mathbf{t}_i = \overrightarrow{GP_i} \times \mathbf{f}_i, \quad (3.24)$$

being k_P the penalty stiffness gain and $\overrightarrow{GP_i}$ the vector from the center of mass (CoM) of the pointshell G to the point P_i . In practice, in order to avoid computing the vector, all pointshell points are defined with respect to the center of mass (CoM) G , which is shifted to the origin (i. e., $G = (0, 0, 0)^\top$).

Three different implementations of the signed distance function are defined (in the order of increasing accuracy):

V_L , distance function using *layered* voxel values (v),

V_S , distance function employing the stored *surface points* (encoded in \mathcal{S}), and

V_I , distance function which *interpolates* the neighbouring floating point distance values (encoded in \mathcal{W}).

The **layered distance function**, illustrated in Figure 3.6 (a,b), can be computed with both the primitive and the enhanced voxelmap and is given by

$$V_L(P) = \xi_V s v(P) + \mathbf{n}^\top \mathbf{d}, \quad (3.25)$$

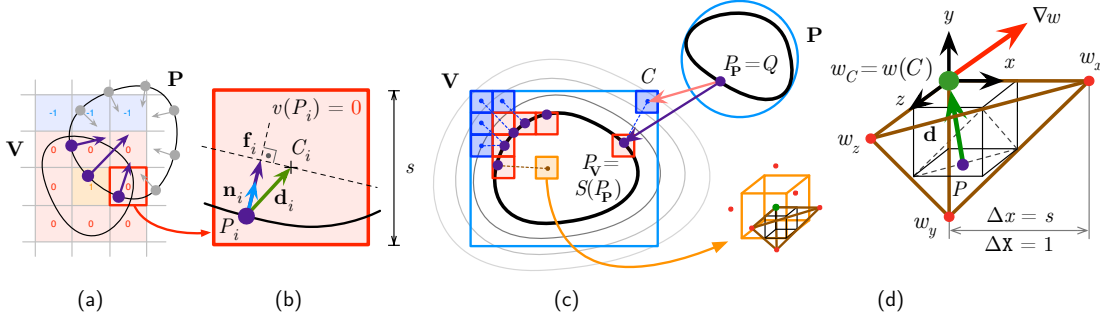


Figure 3.6: (a) Voxelized and point-sampled objects in collision. Each voxel has its voxel layer value v related to its penetration in the voxelmap, and each point P_i its inwards pointing normal vector \mathbf{n}_i . (b) A single point penalty force \mathbf{f}_i can be computed scaling the normal vector with its penetration. (c) Distance computation between a signed distance field A and a point-sphere B : if bounding boxes do not overlap, the first consists in projecting the sphere centers and points P_A on the bounding box of the distance field P'_B ; then, distance values can be interpolated or the closest surface point of the projected point can be found using the surface map. (d) A voxel (blue) surrounded by its neighbor voxel values (red). A valid neighborhood (orange) is selected for linear interpolation of the distance or penetration using the gradient ∇v built out of the neighbor voxel values v_x, v_y , and v_z .

with $\mathbf{d} = \overrightarrow{PC}$ being the vector from the point P to the voxel center C and ξ_V the voxel form factor applied to the voxel size s . This form factor can range from $\xi_V = 1$, for cases in which the voxel value gradient is aligned with any of the voxelmap axes, to $\xi_V = \sqrt{3}$, when the gradient is aligned with any of the voxel diagonals; in practice, its value is usually fixed to the mean $\xi_V = \frac{1}{2}(1 + \sqrt{3})$, although $\xi_V = 1$ yields the most conservative distance to collision computation.

Whereas the first term $\xi_V s v(P)$ from (3.25) refers to the global approximated signed distance in the voxelmap, the second term $\mathbf{n}^T \mathbf{d}$ accounts for the local distance within the voxel. This second term is particularly relevant in case the point is in a surface voxel ($v = 0$), but negligible for high voxel values or small voxel sizes.

The **distance function** which accesses the **surface points** stored in the surface voxel structure \mathcal{S} is defined as

$$V_S(P) = \|P - S(P)\| \quad (3.26)$$

with $S(P)$ being the closest point S constrained to the surface encoded in the voxelmap for a point P in space. This point is obtained in two steps, similarly as when \mathcal{W} is generated: first, the voxel of \mathcal{W} where P lies (analogous to Table B.1 (c)) and which contains the index of the closest surface voxel in \mathcal{S} is determined; second, the pre-computed projected point of that surface voxel is taken, which is the delivered approximation of S . This process is schematically shown in Figure 3.6 (c). In the figure, the special case

in which the pointshell \mathbf{P} is completely out of the boundaries of the voxelmap \mathbf{V} is depicted; in such a situation, the closest boundary voxel to the point P must be found in a previous step (see Table B.1 (c)), and then, \mathcal{W} and \mathcal{S} are accessed to obtain $S(P)$, as described.

In summary, $S(P)$ can be interpreted as a *surface map function*, which, given a point on the pointshell $P_{\mathbf{P}}$, yields its closest point on the surface of the voxelmap $P_{\mathbf{V}}$:

$$S : \mathbb{R}^3 \rightarrow \mathbb{R}^3, P_{\mathbf{V}} = S(P_{\mathbf{P}})$$

$$\text{such that } \begin{cases} v(P_{\mathbf{V}}) = 0, \\ \delta^2 = \|P_{\mathbf{V}} - P_{\mathbf{P}}\|^2 \begin{cases} \min \delta^2 & \text{if } V(P_{\mathbf{P}}) < 0 \text{ (outside),} \\ \max \delta^2 & \text{otherwise.} \end{cases} \end{cases} \quad (3.27)$$

However, given that the S points stored in \mathcal{S} are the projections of the surface voxel centers, the optimum value for δ^2 is not guaranteed with the implemented approach. One possible improvement would consist in projecting the $P_{\mathbf{P}}$ point on the triangle stored in the surface voxel detected in \mathcal{S} (see Section 3.2.1, Table 3.1). Yet, this enhancement decreases computation time performance and is not practicable for large amounts of points every haptic cycle.

The **interpolation distance function** which accesses the floating point distance values w stored in the structure \mathcal{W} is defined as

$$V_I(P) = -(\nabla w)^{\top} \mathbf{d} + w_C, \quad (3.28)$$

where $\mathbf{d} = \overrightarrow{PC}$, as previously, $w_C = w(C)$ the floating point distance value in the voxel where P is (by default, the value in voxel center $C(P)$), and

$$\nabla w = \frac{1}{s} \left(\frac{w_x - w_C}{\Delta X}, \frac{w_y - w_C}{\Delta Y}, \frac{w_z - w_C}{\Delta Z} \right)^{\top}. \quad (3.29)$$

As illustrated in Figure 3.6 (d), the discrete step sizes are $\Delta X, \Delta Y, \Delta Z \in \{-1, 1\}$; their values are chosen depending on the octant of the voxel where P lies. The neighboring floating point distances w_x, w_y, w_z are selected accordingly. This function V_I is considered the most accurate among all three, since local linear interpolation of distances is performed for the queried point using previously generated real distance values.

3.2.3.3 Comparison of the Signed Distance Function $V(P)$ Calls

Figure 3.7 displays accuracy differences between the three signed distance function calls V_L , V_S , and V_I ; additionally, Table 3.2 summarizes all structures and calls with their performance properties. These qualitative values are later quantitatively analyzed in the

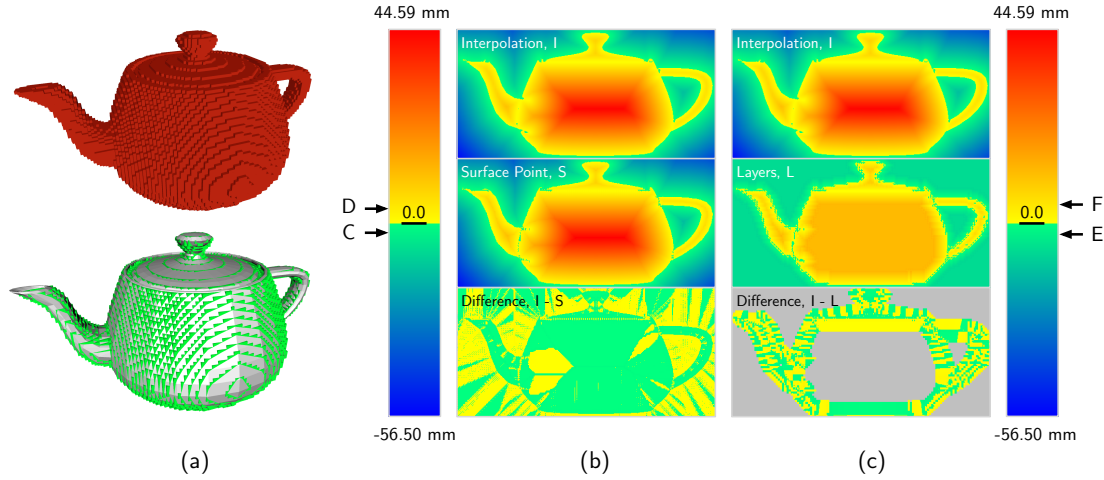


Figure 3.7: The Utah Teapot (2330 triangles) with a bounding box of $236.2 \text{ mm} \times 115.7 \text{ mm} \times 147.0 \text{ mm}$ is voxelized with a resolution of $s = 2 \text{ mm}$ (voxel size), yielding a voxelmap of $122 \times 61 \times 77$ voxels. (a) Surface voxel layer (red, $v = 0$) and first inner layer (green, $v = 1$) are shown. 500×250 pixels of the sagittal plane are rastered computing their signed distance $V(P)$ value. The point normal vector is the gradient of the voxelmap in its respective voxel. Distances are coded in color: warm for positive (inner) values, cold for negative (outer). Gray values denote areas where no significant data is available: The layered representation (\mathcal{V}) was compressed to contain values around the surface only (6 inner and 4 outer layers). (b) Interpolated section (ground truth, V_I), surface point map section (V_S), and difference of both, which is bounded to $[C, D] = [-2.80, 3.40] \text{ mm}$; (c) Interpolated section (ground truth, V_I), layer distance section (V_L), and difference of both, which is bounded to $[E, F] = [-3.19, 4.29] \text{ mm}$.

benchmarking experiments presented in Section 3.4. Although the results are clearly object and resolution dependent, they depict thoroughly the performance of the different calls in the voxelmap.

The differences in computation time are due to the operations performed and the amount of data accessed during each call. In the case of V_L , the unique value v has to be detected in \mathcal{V} for a (pointshell) point P in space. On the other hand, V_S and V_I require accessing four values each: i_S, S_x, S_y, S_z and w_x, w_y, w_z, w_C , respectively.

The distance field sections in Figure 3.7 were generated rasterizing for each of the voxel section 4×4 points on average. The interpolation distance function V_I is considered the ground truth. The surface point distance function V_S overestimates (turquoise) penetration values compared to V_I ; on the other hand, distances are underestimated (yellow) except in the overestimation stripes that emerge in high curvature surface points. In both cases, the error is bounded to $1.7s$.

The error is the biggest with layered distance function V_L , as expected, but still bounded to $2.15s$. Aliasing issues, characteristic of discretized data structures, manifest clearly as alternating under- and overestimation regions. However, it is important to

Table 3.2: Summary of structures, values, and queries that are defined in primitive and enhanced voxelmaps (\mathbf{V}). Speed and accuracy are coded with values in seven-point scala ranging from $---$ (very slow/inaccurate) to $+++$ (very fast/accurate), being \sim mediocre, but acceptable. All queries can be called during online interactive simulations, but those with a speed equal or less than \sim should be used with a unique or few points every haptic cycle.

	Sets	Values		Online Calls					
				Function	Speed	Accuracy			
Primitive \mathbf{V}	\mathcal{V}	$\{v\}_{i_{\mathcal{V}}=1}^{i_{\mathcal{V}}=N_{\mathcal{V}}}$	$v \in \mathbb{N}$	$V_L(P)$	++	+			
				Gradient ¹ $S(P)$	\sim	\sim			
Enhanced \mathbf{V}	\mathcal{V}	$\{v\}_{i_{\mathcal{V}}=1}^{i_{\mathcal{V}}=N_{\mathcal{V}}}$	$v \in \mathbb{N}$	$V_L(P)$	++	+			
				\mathcal{S}	$\{(S, (T_1, T_2, T_3))\}_{i_{\mathcal{S}}=1}^{i_{\mathcal{S}}=N_{\mathcal{S}}}$	$S, T \in \mathbb{R}^3$	$V_S(P)$	+	++
				\mathcal{W}	$\{(w, i_{\mathcal{S}})\}_{i_{\mathcal{W}}=1}^{i_{\mathcal{W}}=N_{\mathcal{W}}=N_{\mathcal{V}}}$	$w \in \mathbb{R}, i_{\mathcal{S}} \in \mathbb{N}$	$V_I(P)$	\sim	+++
				Gradient ¹ $S(P)$	\sim	\sim			
				Fast ² $S(P)$	++	+			

¹ Gradient $S(P)$ delivers the voxel center C of the closest surface voxel of P after performing gradient descent as defined in (3.23).

² Fast $S(P)$ delivers the *pre-computed* closest point of P accessing \mathcal{S} .

bear in mind that the sections depict the results of one single point; in real situations, hundreds of points might collide, attenuating the aliasing effect considerably, as later discussed in Section 3.4. More importantly, the error in critical regions (voxels close to the surface) produced by V_L is not much bigger than the one produced by V_S , while lighter data structures are achieved using only \mathcal{V} . This makes the primitive voxelmaps very useful structures for large objects or situations in which the computational speed and memory space have priority over accuracy.

As mentioned, Section 3.4 provides more results related to the different signed distance function calls. Among others, the effect of different voxelmap resolution is analyzed.

3.2.4 Enhanced Pointshells: Point-Sphere Trees

The primitive pointshell or point cloud described in Section 3.2.1 is a list of unordered 6D points that represent the original mesh. This data structure is enhanced to obtain the enhanced pointshell, which encodes multiresolution point neighborhood information. That improvement is performed as follows:

- (i) The set of points \mathcal{P} defined in (3.6) is extended to contain a quality value q for each point related to the curvature; the updated set of 7D points is

$$\mathcal{P} = \{(P, \mathbf{n}, q)_1, \dots, (P, \mathbf{n}, q)_i, \dots, (P, \mathbf{n}, q)_{N_{\mathcal{P}}}\} = \{(P, \mathbf{n}, q)_i\}_{i=1}^{N_{\mathcal{P}}}. \quad (3.30)$$

(ii) A set \mathcal{C} of $N_{\mathcal{C}}$ elements or clusters c is defined:

$$\mathcal{C} = \{c_1, \dots, c_i, \dots, c_{N_{\mathcal{C}}}\} = \{c_i\}_{i=1}^{N_{\mathcal{C}}}. \quad (3.31)$$

This set contains the information with which points are hierarchically organized in levels or subsets that sample the object completely with different resolutions or point densities. Additionally, a sphere tree is built upon the point hierarchy in order to detect faster relevant regions during realtime calls. The elementary unit in \mathcal{C} , the cluster c , represents a surface patch on the object and which is sampled by points and it is defined as

$$c = \left(L, K, (R, X), \{\&c(P, \mathbf{n}, q)_j\}, \{\&c_j^c\}, \&c^p, q_{\max} \right), \quad (3.32)$$

where

- L is the level where the cluster is, being the highest level $L = 1$ and the level of the leaves $L = N_L$;
- K is the number of *cluster points*, which satisfies $1 \leq K \leq N_K$, with N_K being the branching factor in the tree ($N_K = 4$ default);
- (R, X) are the radius R and the center X of the minimally bounding sphere that contains all children points recursively branched until the leaf level;
- $\{\&c(P, \mathbf{n}, q)_j\}_{j=1}^K$ are the memory addresses of the K *cluster points*, with $P_{j=1}$ being the *cluster parent point* which represents the whole cluster; the addresses point to elements in \mathcal{P} ;
- $\{\&c_j^c\}_{j=1}^K$ are the addresses of the K *children clusters* associated to the *cluster points*; note that each children cluster recursively contains addresses of other cluster points and children clusters; the addresses point to elements in \mathcal{C} ;
- $\&c^p$ is the address of the *parent cluster* in \mathcal{C} ;
- and q_{\max} is the maximum quality value of all points that are recursively branched from the cluster until the leaf level.

In close relationship to \mathcal{C} , the FIFO (First in First Out) queue or set of clusters \mathcal{Q} is additionally defined. This is not an intrinsic structure of the pointshell \mathbf{P} , but it supports

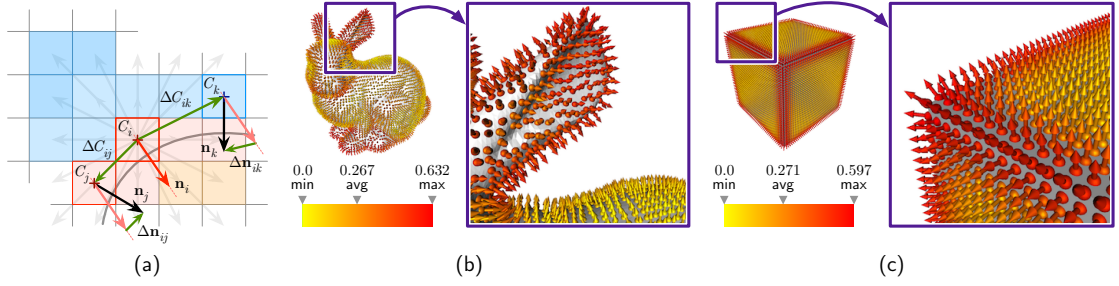


Figure 3.8: Point quality values q (m^{-1}) related to the curvature: (a) The quality value q_i of a point P_i is computed observing the variation of the normal vector ($\Delta \mathbf{n}_{ij}$) in the surrounding voxel centers (C_j) within the neighborhood of the point $\mathcal{N}_{V,2}(P_i)$. (b) Pointshell representation of the Stanford Bunny with a bounding volume of $105 \text{ mm} \times 206 \text{ mm} \times 160 \text{ mm}$: 5584 points are created using a voxel size $s = 4 \text{ mm}$. Normal vectors are flipped to point outwards and coded in color (yellow to red) according to the quality value q of the point. (c) Pointshell representation of a cube with 2000 mm edge length: 9602 points are created using a voxel size $s = 50 \text{ mm}$. Normal vectors are again flipped and coded in color according to q .

the ordered access to clusters during realtime calls:

$$\mathcal{Q} = \{c_i\}_{i=1}^{N_{\mathcal{Q}}}, \quad \mathcal{Q} \subseteq \mathcal{C} \quad (3.33)$$

At the beginning of each proximity or collision query, this queue is empty, $\mathcal{Q} \leftarrow \emptyset$; during the query, it is filled with clusters that are detected to be likely colliding and emptied step by step again by the end of the haptic cycle. This section presents how \mathcal{P} and \mathcal{C} are created, whereas Section 3.3 explains in greater detail how \mathcal{Q} is defined and used for collision checks between complex geometries.

The input structures necessary to build the point-sphere tree are the primitive data structures of the object introduced in Section 3.2.1.

3.2.4.1 Point Qualities

Quality values q represent geometry variations nearby a pointshell point P . In this way, points with low q values can be sorted out during realtime collision checks if the time budget is low, as later explained in Section 3.3.2.

Figure 3.8 illustrates the computation of point quality values q during data structure generation and displays some exemplary cases. As shown in there, the underlying primitive voxelmap is used in the process. First, given a 6D pointshell point (P_i, \mathbf{n}_i) which lies in the voxel with center C_i , the normal vectors \mathbf{n}_j of the voxels within the surrounding neighborhood $\mathcal{N}_{V,2}$ are calculated according to (3.12). Second, the normal and displacement variation vectors of each neighboring voxel are computed:

$$\Delta \mathbf{n}_{ij} = \mathbf{n}_j - \mathbf{n}_i = (\Delta n_{ij,x}, \Delta n_{ij,y}, \Delta n_{ij,z})^\top, \quad (3.34)$$

$$\Delta C_{ij} = C_j - C_i = (\Delta C_{ij,x}, \Delta C_{ij,y}, \Delta C_{ij,z})^\top. \quad (3.35)$$

And finally, the quality q is evaluated as the total normal vector variation per distance unit swept in the neighborhood:

$$q(P_i) = \sum_{\substack{C_j, \mathbf{n}_j \in \mathcal{N}_{V,2}(P_i) \\ \Leftrightarrow j=1, \dots, N_N}} \frac{\|\Delta \mathbf{n}_{ij}\|}{\|\Delta C_{ij}\|}. \quad (3.36)$$

Note that the quality value q is directly related to the curvature of the object in the point P . Indeed, as suggested in [Mas03], the curvature approximation can be extracted from the matrix \mathbf{Q} which relates $\Delta \mathbf{n}_{ij}$ and ΔC_{ij} . To this end, first, the variation vectors in (3.34) and (3.35) are redefined by considering only their components parallel to \mathbf{n}_i :

$$\Delta \mathbf{n}_{ij} \leftarrow (\mathbf{n}_j - \mathbf{n}_i) - ((\mathbf{n}_j - \mathbf{n}_i)^\top \mathbf{n}_i) \mathbf{n}_i = (\Delta n_{ij,x}, \Delta n_{ij,y}, \Delta n_{ij,z})^\top \quad (3.37)$$

$$\Delta C_{ij} \leftarrow (C_j - C_i) - ((C_j - C_i)^\top \mathbf{n}_i) \mathbf{n}_i = (\Delta C_{ij,x}, \Delta C_{ij,y}, \Delta C_{ij,z})^\top \quad (3.38)$$

Next, \mathbf{Q} is defined as

$$\begin{bmatrix} \Delta C_{i1,x} & \Delta C_{i1,y} & \Delta C_{i1,z} \\ \Delta C_{i2,x} & \Delta C_{i2,y} & \Delta C_{i2,z} \\ \vdots & \vdots & \vdots \\ \Delta C_{iN_N,x} & \Delta C_{iN_N,y} & \Delta C_{iN_N,z} \end{bmatrix} \mathbf{Q} = \begin{bmatrix} \Delta n_{i1,x} & \Delta n_{i1,y} & \Delta n_{i1,z} \\ \Delta n_{i2,x} & \Delta n_{i2,y} & \Delta n_{i2,z} \\ \vdots & \vdots & \vdots \\ \Delta n_{iN_N,x} & \Delta n_{iN_N,y} & \Delta n_{iN_N,z} \end{bmatrix} \Leftrightarrow \mathbf{CQ} = \mathbf{N}, \quad (3.39)$$

with N_N being the number of neighbor voxels around P_i . This is a linear regression problem and the optimum \mathbf{Q} can be isolated applying the pseudo-inverse of \mathbf{C} .

Principal component analysis of \mathbf{Q} reveals that the last eigenvector is parallel to \mathbf{n}_i , since both variation vectors in (3.37) and (3.38) have been projected on the tangent plane of \mathbf{n}_i . Therefore, \mathbf{Q} can be reduced to the rank-2 *shape operator* which has the two principal curvatures κ_1, κ_2 as eigenvalues.

In that sense, the point quality could be also defined as the mean curvature $\frac{1}{2}(\kappa_1 + \kappa_2)$. However, the implementation in (3.36) is considerably faster, numerically more robust, and, although it does not yield the curvature, it fulfills the sought purpose: the definition of an indicator which describes geometry change to be able to filter out geometrically less meaningful points during time critical calls.

Figure 3.9: (a) The $K = N_K$ closest points of a seed point are searched ($N_K = 4$ here). The first $K - 1$ points and the seed constitute a cluster, whereas the last point will be the seed for the next cluster (green). For each cluster, a parent point (closest to the CoM of the cluster) is selected (red). The algorithm successively steps to create the next upper level by clustering the just created parent points. (b) Minimally bounding spheres are computed for each cluster. These encapsulate all points within the cluster recursively reachable until the leaf level.

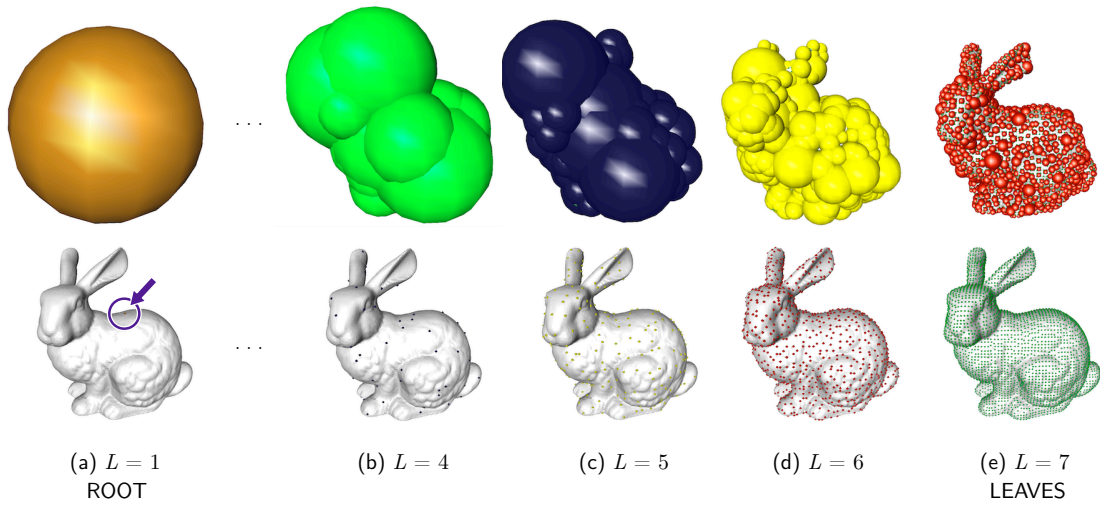
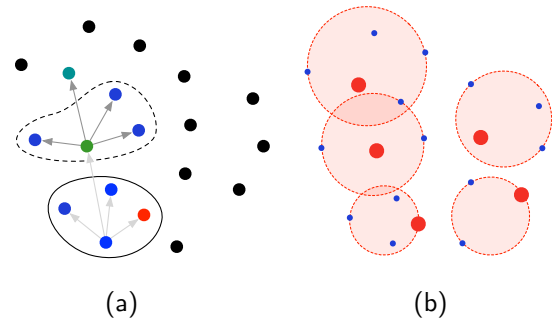


Figure 3.10: Different hierarchy levels of the the same Stanford Bunny as in Figure 3.8 ($s = 4$ mm, 5584 points) are shown: Sphere tree levels are displayed at the top row and the corresponding point tree levels at the bottom row. Note that each point tree level samples the whole geometry with a different resolution. The hierarchy has a branching factor of $N_K = 4$ (points per cluster), which results in $N_C = 1916$ clusters (and, hence, spheres) divided in $N_L = 7$ levels.

3.2.4.2 Hierarchy Generation

Point-sphere hierarchies are completely encoded in the previously introduced cluster set \mathcal{C} and are built bottom-up starting with the leaf points (all the points in \mathcal{P} , thus, the finest resolution). First, clustering and hierarchy level definition is performed, and then, minimally bounding spheres are computed for each cluster recursively. The whole process is outlined in Figure 3.9 and an exemplary point-sphere hierarchy is shown in Figure 3.10.

When clustering is carried out, points are grouped into clusters of K points according to their *similarity*. This *similarity* is currently defined as the regular Euclidean distance, but other criteria are possible, such as the geodesic distance, or implementations that take into account the normal vectors or quality values of the points. The branching factor N_K (default, $N_K = 4$) is the actual parameter defined by the user in order to control the number of points in a cluster, with K automatically adjusted. In regular situations, $K = N_K$, except when there are not enough points or the distance between them is above a given threshold, as detailed later.

The first point or seed point is randomly chosen and the next most similar K points are located in the neighborhood. At this stage, the support structure which matches surface voxels and projected points mentioned in Section 3.2.1 is used in order to avoid quadratic complexity during the search. A cluster of K elements is formed using the initial seed point and the next $K - 1$ closest neighbors. The K^{th} closest neighboring point is used as the initial seed point for the next cluster. Before jumping to the next cluster, the point in the cluster closest to the average or center of mass (CoM) is selected as the *cluster parent point*. This process is repeated until all the elements belong to some cluster. When this occurs, the algorithm starts grouping the parent points of the previously defined clusters. The stopping criterion of this recursion is met when the top level of the tree which contains only one cluster is reached. This one cluster is the *root cluster* in level $L = 1$.

Given a branching factor N_K , each level $L > 1$ in the hierarchy will have approximately N_K times more points than the previous higher level $L - 1$:

$$N_{\mathcal{P},L} = N_K N_{\mathcal{P},L-1}, \quad N_{\mathcal{P},L} = \left\lceil \frac{N_{\mathcal{P}}}{(N_K)^{L-1}} \right\rceil, \quad (3.40)$$

with $N_{\mathcal{P}}$ being the total number of points in \mathcal{P} and $N_{\mathcal{P},L}$ the number of points in level L . The points in each level sample the whole object surface with a different resolution s which can be modeled with (3.16) and (3.40):

$$s_L = s_{L-1} \sqrt{N_K}. \quad (3.41)$$

Additionally, the total number of levels N_L can be estimated from (3.40) as

$$N_L = \lceil \log_{N_K} N_{\mathcal{P}} \rceil, \quad (3.42)$$

and it is the level number of the leaf points, that consist of all points in \mathcal{P} .

The explained sequential clustering approach can yield clusters containing points that are much further away from each other than on average in the level. That occurs because the only criterion for building the next adjacent cluster is finding the closest point to the current cluster; additionally, it is not guaranteed that a list of closest points not assigned to clusters is going to be as compact as a proper cluster. In order to eliminate those exceptions, a distance threshold is defined between a parent point P_i and the other cluster points P_j :

$$\|P_i - P_j\| \leq \frac{3}{4} N_K s \sqrt{3} (\sqrt{N_K})^{N_L - L}, \quad (3.43)$$

this heuristic threshold distance in (3.43) is obtained assuming that points should be a maximum of $\frac{3}{4} N_K$ voxels away from each other (worst case, in diagonal) and considering that the resolution between levels varies according to (3.41). When a point yields a distance value above the threshold, it is pulled out from the cluster and assigned to a neighboring cluster in which it fulfills the threshold criterion. Therefore, although the target size of a cluster is always N_K , the real number of points K of each cluster is determined on the fly. It may happen that a point which was pulled out does not fulfill the threshold for any neighboring cluster; in that case, a single-point cluster is created with it. Relaxing (3.43) helps to eliminate the number of single-point clusters, but it lowers the average compactness of all the other clusters.

After a level L is clustered, all generated clusters are added to \mathcal{C} in their order of creation. As a result, the first clusters of \mathcal{C} belong to the last level $L = N_L$ and contain leaf points, and the very last cluster in \mathcal{C} is the *root cluster* c_1 in level $L = 1$. Every time a cluster c is added to the hierarchy, all available cluster values necessary in (3.44) are registered: L , K , the addresses in \mathcal{P} of the K cluster points and also the reciprocal addresses between children and parent clusters (i. e., $\{\&c_j^c\}$ and $\&c^p$). Note that although the number of clusters $N_{\mathcal{C}}$ is determined at the end of the clustering process, it can be estimated as

$$N_{\mathcal{C}} \simeq \sum_{i=0}^{N_L} (N_K)^i. \quad (3.44)$$

After building the point-tree, the sphere-tree is created upon it. To that purpose, bounding spheres for each cluster are computed recursively. Minimally bounding spheres [FG04] are calculated using the CGAL library [FGH+16] (see Figure 3.9 (b)). Each cluster sphere contains all the cluster points and all the children cluster points until

Table 3.3: Summary of structures, values, and queries that are defined in primitive and enhanced pointshells (\mathbf{P}).

	Sets	Values		Online Call Description
Primitive \mathbf{P}	\mathcal{P}	$\{(P, \mathbf{n})_{i_{\mathcal{P}}}\}_{i_{\mathcal{P}}=1}^{i_{\mathcal{P}}=N_{\mathcal{P}}}$	$P, \mathbf{n} \in \mathbb{R}^3$	All 6D points in \mathbf{P} form an unordered list which is completely accessed in realtime calls without hierarchies.
Enhanced \mathbf{P}	\mathcal{P}	$\{(P, \mathbf{n}, q)_{i_{\mathcal{P}}}\}_{i_{\mathcal{P}}=1}^{i_{\mathcal{P}}=N_{\mathcal{P}}}$	$P, \mathbf{n} \in \mathbb{R}^3, q \in \mathbb{R}$	Only selected 7D points referenced from clusters in \mathcal{Q} are accessed.
	\mathcal{C}	$\{c_{i_{\mathcal{C}}}\}_{i_{\mathcal{C}}=1}^{i_{\mathcal{C}}=N_{\mathcal{C}}}$	see (3.32)	Addresses to clusters (c) in \mathcal{C} are pushed to the queue \mathcal{Q} for later access during realtime calls. Each c contains addresses to points in \mathcal{P} .
	\mathcal{Q}	$\{c_{i_{\mathcal{Q}}}\}_{i_{\mathcal{Q}}=1}^{i_{\mathcal{Q}}=N_{\mathcal{Q}}}$	$\mathcal{Q} \subseteq \mathcal{C}$	The FIFO queue \mathcal{Q} is not a constant structure in \mathbf{P} (as \mathcal{P} or \mathcal{C}), but is created every haptic cycle by adding likely colliding clusters from \mathcal{C} .

reaching the leaf level. Thus, the result is an optimally wrapped sphere-tree, similar to the one defined in [WZ09b], where each cluster sphere contains all its children points, but not the children spheres. All points to be enclosed are detected by simply following the children cluster addresses stored in each cluster during the point-tree generation. In this phase, the maximum quality value q_{\max} along all the points within the sphere is also computed and saved in the cluster.

This sphere hierarchy enables rapidly locating likely collision areas performing sphere checks, as explained in Section 3.3. Figure 3.10 shows the point-sphere tree of the Stanford Bunny and Table 3.3 summarizes and compares the most important structures of primitive and enhanced pointshells.

Enhanced pointshells can also be saved in files. All files used for this work were generated in less than 20 seconds and have a size smaller than 2 MB using resolutions slightly smaller than $s = 5$ mm in desktop-sized objects. Refer to [SHPH08] and Figure 3.16 for more information on generation time, file size, and further exemplary snapshots.

3.3 Proximity and Collision Queries with Complex Objects

This section builds on the data structures and functions introduced and presents how LoD, time-critical proximity and collision queries can be performed between arbitrarily complex geometries. Three basic elements should be recalled:

- (i) The three signed distance voxelmap functions $V(P)$ defined in (3.25), (3.26), and (3.28) (Section B). The three are considered interchangeable in this section unless one is specified due to accuracy reasons.
- (ii) The definition of six-DoF forces in (3.24) (Section B).
- (iii) The clusters from the set \mathcal{C} used to populate the FIFO queue \mathcal{Q} , in (3.31), (3.32), and (3.33) (Section B).

A brute force approach (for CPU implementations) for detecting the minimum distance or collision forces between two objects consists in representing one with a voxelmap and the other as a pointshell, and then checking all points in \mathcal{P} , as proposed in the original VPS [MPT99]. This procedure has the advantage of yielding almost constant-time collision data, but the drawback of being limited to few thousands of points is the 1 kHz update rate for haptics is to be reached.

A more efficient technique consists in using the sphere-tree for discarding non-colliding regions and narrowing the likely colliding ones. The FIFO queue \mathcal{Q} is used for that purpose, as explained in detail in the next Section 3.3.1. Algorithm 3.1 shows this general hierarchical traverse (called at least every 1 ms) of the point-sphere tree in order to compute distance values and six-DoF collision forces between voxelmap and pointshell structures that can represent complex geometries. This traverse is graphically outlined in Figure 3.11.

Two types of traverses are explained for the same Algorithm 3.1: (i) a general traverse detailed in the next section Section 3.3.1 and (ii) a time-critical traverse which can reduce the resolution described in Section 3.3.2. Both deliver the same type of information, but with different resolutions.

3.3.1 General Hierarchical Traverse

First, the input and output data of Algorithm 3.1 are defined:

3.3.1.1 Input Data

- (i) Voxelmap \mathbf{V} and pointshell \mathbf{P} data structures. The voxelmap can be either a basic layered structure or an enhanced signed distance field (with \mathcal{V} , \mathcal{S} , and \mathcal{W}), whereas

Algorithm 3.1: $(p_d, \mathbf{f}_P, \mathbf{t}_P, Q, S(Q), \eta, \mathcal{M}) = \text{collisionQuery}(\mathbf{V}, \mathbf{P}, d, p_c, \eta_c, q_c)$

Data: Signed distance field (voxelmap) \mathbf{V} containing \mathcal{V} , \mathcal{S} , and \mathcal{W} ; Point-sphere hierarchy (pointshell) \mathbf{P} containing \mathcal{P} and \mathcal{C} , transformed into voxelmap coordinates with $\mathbf{V}\mathbf{H}_P$; Safety margin d ; Critical distance p_c ; Critical load η_c ; Critical quality q_c .

Result: Signed distance p_d between voxelmap and pointshell; Penalty forces \mathbf{f}_P and torques \mathbf{t}_P ; Closest features (points) on pointshell Q and voxelmap $S(Q)$; Computational load η ; Optionally, contact manifold \mathcal{M} . Values in voxelmap coordinates.

```

//Get critical level from load in last loop; if first loop or  $\eta_c = 1$ ,  $L_c = N_L$ 
1  $L_c \leftarrow Q.\text{getCriticalLevel}(\eta_c)$ 
//Reset FIFO queue with root cluster that bounds all points
2  $Q \leftarrow \emptyset$ 
3  $Q.\text{setCriticalQuality}(q_c)$ 
4  $c_1 \leftarrow \mathcal{C}.\text{getRootCluster}()$ 
5  $Q.\text{push}(c_1)$ 
//Initialization of output values
6  $p_d \leftarrow 0$ ,  $\mathbf{f}_P \leftarrow \mathbf{0}$ ,  $\mathbf{t}_P \leftarrow \mathbf{0}$ ,  $Q \leftarrow c_1.P_1$ ,  $\eta \leftarrow 0$ ,  $\mathcal{M} \leftarrow \emptyset$ 
7 while  $Q \neq \emptyset$  do
8    $c \leftarrow Q.\text{pop}()$ 
9   if  $V(c.X) + d + c.R \geq -p_c$  then
//Sphere of cluster  $c$  is colliding or closer than critical distance  $p_c$ 
//Check parent point for collision
10  if  $\text{unchecked}(c.P_1)$  AND  $V(c.P_1) + d \geq 0$  then
//Cluster parent point colliding  $\rightarrow$  Compute forces and extend manifold
11     $\mathbf{f}_P \leftarrow \mathbf{f}_P + (V(c.P_1) + d)c.\mathbf{n}_1$ ;  $\mathbf{t}_P \leftarrow \mathbf{t}_P + c.P_1 \times (V(c.P_1) + d)c.\mathbf{n}_1$ 
12     $\mathcal{M}.\text{tryAdd}(c.P_1, c.\mathbf{n}_1, V(c.P_1) + d)$ 
//Save parent point as closest feature if it optimizes distance
13  if  $V(c.P_1) + d > p_d$  then
14     $p_d \leftarrow V(c.P_1) + d$ ;  $Q \leftarrow c.P_1$ 
//Check all cluster points for collision if  $c$  belongs to the critical level  $L_c$ 
15  if  $c.L == L_c$  then
16    for  $k = 2$  to  $K$  do
17      if  $V(c.P_k) + d \geq 0$  then
//Point in last level colliding  $\rightarrow$  Compute forces and extend manifold
18         $\mathbf{f}_P \leftarrow \mathbf{f}_P + (V(c.P_k) + d)c.\mathbf{n}_k$ ;  $\mathbf{t}_P \leftarrow \mathbf{t}_P + c.P_k \times (V(c.P_k) + d)c.\mathbf{n}_k$ 
19         $\mathcal{M}.\text{tryAdd}(c.P_k, c.\mathbf{n}_k, V(c.P_k) + d)$ 
//Save cluster point as closest feature if it optimizes distance
20      if  $V(c.P_k) + d > p_d$  then
21         $p_d \leftarrow V(c.P_k) + d$ ;  $Q \leftarrow c.P_k$ 
//Push children of colliding or close-to-collision sphere
22     $Q.\text{tryPush}(c.\{\&c_k^c\}_{k=1}^K, c.q_{\max})$ 
23  else
//Sphere is not colliding or at least further away than the critical distance  $p_c$ 
//Save distance if its sphere distance optimizes  $p_d$ 
24  if  $V(c.X) + d + c.R > p_d$  then
25     $p_d \leftarrow V(c.X) + d + c.R$ 
26  $\rho \leftarrow \text{computeCorrectionFactor}()$ 
27  $(\mathbf{f}_P, \mathbf{t}_P) \leftarrow (\rho\mathbf{f}_P, \rho\mathbf{t}_P)$ 
28  $\eta \leftarrow \text{computeLoad}()$ 
29  $Q.\text{saveLoad}(\eta)$ 
30  $S(Q) \leftarrow \mathbf{V}.\text{getSurfacePoint}(Q)$ 
31 return  $(p_d, \mathbf{f}_P, \mathbf{t}_P, Q, S(Q), \eta, \mathcal{M})$ 

```

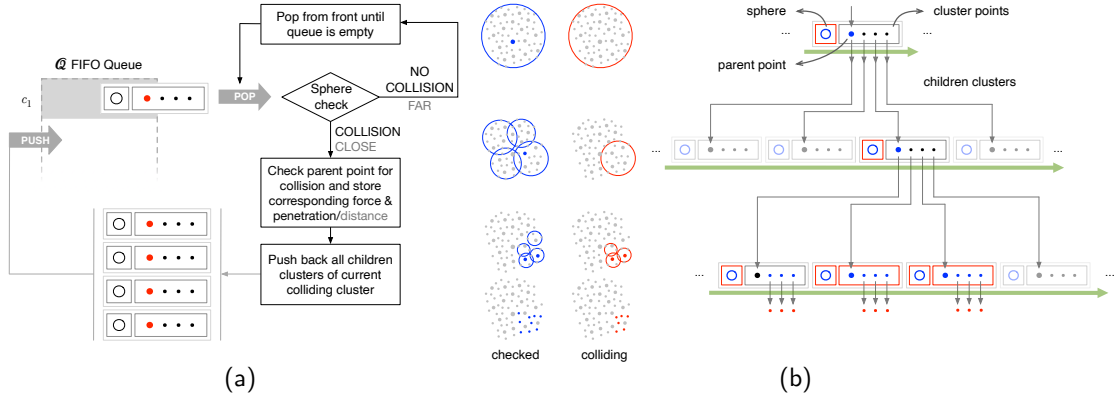


Figure 3.11: Graphical explanation of the unified proximity and collision computation procedure in Algorithm 3.1: (a) The algorithm is summarized in block diagrams. (b) Breadth-first traverse (green arrow) of the clusters in the point-sphere tree through the last three levels. Blue spheres and points represent checked elements, while red elements are the colliding ones. The algorithm converges to the collision area in the lower right part.

the pointshell must be an enhanced point-sphere tree (with \mathcal{P} and \mathcal{C}). Additionally, all elements in \mathcal{P} are given in voxelmap coordinates with the homogeneous transformation $\mathbf{V}\mathbf{H}_P$.

- (ii) Safety margin d with which the voxelmap is artificially dilated. This value is typically set to be similar to the voxel size s and produces breaking forces in the safety region defined by the layer created by d , as well as more conservative distance-to-collision values. It is also used during constraint-based force rendering, explained in Chapter 4. However, if no breaking forces and rather real distance values are required, it can be set $d = 0$.
- (iii) Critical distance or distance computation threshold p_c . This value controls distance computation with point-sphere trees. If $p_c = 0$, a poor distance approximation p_d (see output data) is going to be provided, but the algorithm is going to be extremely fast (less than $10 \mu s$) in non-overlapping configurations. Intuitively, p_c defines the voxelmap isosurface which contains the volume in which pointshell points need to be evaluated in order to determine accurate distance-to-surface values.
- (iv) Critical load $0 \leq \eta_c \leq 1$ and quality $0 \leq q_c \leq 1$. These values control, respectively, the deepest level visited and the minimum relative point quality in time-critical traverses, explained in Section 3.3.2. If $\eta_c = 1.0$, the critical level will be the deepest level in the hierarchy, i. e., $L_c = N_L$; with $q_c = 0.0$, no points are filtered out due to low quality values.

3.3.1.2 Output Data

- (i) The signed distance p_d between the voxelmap and pointshell structures. If $p_d > 0$, objects overlap, being p_d the maximum penetration value; if $p_d < 0$, objects are a distance $|p_d|$ apart from each other; otherwise, if $p_d = 0$, objects are in surface contact. Note that the value p_d is the distance computed with a voxelmap dilated with a safety margin d

$$p_d = p + d, \quad (3.45)$$

with p being the real distance/penetration between geometries, evaluated as the maximum signed distance value across all points on the pointshell:

$$p = \max\{V(P_i)\}, \quad P_i \in \mathcal{P}. \quad (3.46)$$

- (ii) Penalty force \mathbf{f}_p and torque \mathbf{t}_p vectors computed according to (3.24). These values are computed with the voxelmap dilated with the safety margin d .
- (iii) Closest points on pointshell Q and voxelmap $S(Q)$, computed on the real surface, without considering the safety margin d for voxelmap dilation. If $p > 0$, in order to solve the overlap, the objects need to be translated the distance p so that Q matches $S(Q)$.
- (iv) Computational load η , evaluated essentially as the ratio between colliding and checked elements (points and spheres). The value of η is related to time-critical traverses, thus, its definition is elaborated in Section 3.3.2.
- (v) Contact manifold $\mathcal{M} = \{P_i, \mathbf{n}_i, V(P_i)\}_{i=1}^{N_{\mathcal{M}}}$, which contains information of colliding points. In the implementation, addresses to the points in \mathcal{P} together with their penetration $V(P_i)$ are registered. The set \mathcal{M} is empty if objects are disjoint, i. e., $p_d < 0$.

3.3.1.3 Collision Computation ($p_c = 0$, $\eta_c = 1$, $q_c = 0$)

As already mentioned, the FIFO queue \mathcal{Q} is the main support structure with which collision and proximity computation can be performed as shown in Algorithm 3.1. First, \mathcal{Q} must be cleared (line 2) and filled with the root cluster c_1 which has the sphere that bounds all points in \mathcal{P} (line 4). Note that \mathcal{Q} is more than a simple set; it also has methods (it is implemented as a class) with which inclusion and extraction of selected elements to or from it can be performed in a FIFO ordered manner. Besides that, the structure handles the last or critical level L_c , as further explained later in Section 3.3.2.

In the loop (line 7), a cluster c is popped (line 8) every iteration from \mathcal{Q} until the queue is empty, and it is checked whether cluster c 's sphere collides (line 9). If the cluster

sphere collides, children clusters are added to \mathcal{Q} (line 22) for later processing (for they might also collide), and the collision forces of the parent point are computed (lines 10-11) in case its signed distance is positive. If the sphere is not colliding, the inter-object signed distance p_d is checked (line 13) and updated (line 14) if necessary, but no children clusters are added to \mathcal{Q} : since the current cluster sphere is not colliding and bounds all points divided into children clusters, it can be assured that none of them is colliding. If accurate distance computation is not performed (i. e., $q_c = 0$), the sphere collision check of line 9 can be simplified as

$$\underbrace{V(c.X) + d}_{\text{distance sphere center}} + \underbrace{c.R}_{\text{sphere radius}} \geq 0, \quad (3.47)$$

which basically tests whether the center of the sphere is at least as far as its radius from the surface of the voxelmap object – recall that the points in the outer region yield negative distance values.

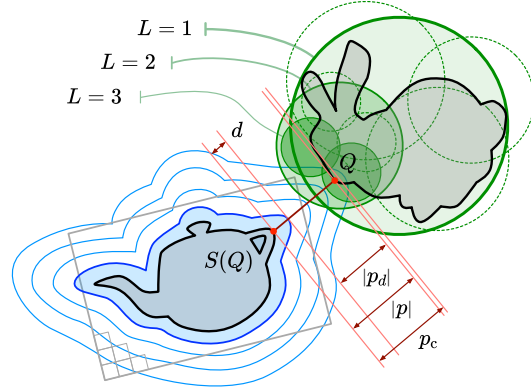
In case the cluster belongs to the critical level ($L_c = N_L$ if $\eta_c = 1$), the rest of $K - 1$ children points are also checked for collision, as done with the cluster parent point; in this sense, line segments 10-11 and 17-18 are analogous, but applied to parent and children points, respectively. This loop is schematically outlined in Figure 3.11 (a). Given the structure of \mathcal{C} and \mathcal{Q} , clusters are swept in a breadth-first manner, level by level (see Figure 3.11 (b)).

The type of distance function V_L , V_S , or V_I (Section B) is deliberately not specified, since any of them can be used. Note that the safety margin d is always added to the signed distance function. Additionally, as previously indicated, six-DoF forces are computed according to (3.24), but the safety distance d is considered in them; therefore, breaking forces are computed only if the objects overlap in the safety region.

As in the case of \mathcal{Q} , the contact manifold \mathcal{M} is implemented as a class which has methods for selecting the points that are trying to be added and these can be ordered inside the structure in clusters, as further explained in Section 3.3.1.5.

The force and torque values explained so far increase with the area in contact; however, in real situations, the contact normal forces are independent of the contact areas: for instance, a long stick or cylinder should experience a similar contact stiffness when colliding with the tip or the long lateral side against a large plane. Therefore, the penalty forces computed in the main loop of Algorithm 3.1 should be inversely scaled with the magnitude (or area) of the contact surface. That scaling is achieved with the correction factor ρ (lines 26 and 27), defined as:

Figure 3.12: Distance computation between a signed distance field (voxelmap, Utah Teapot) and a point-sphere tree (pointshell, Stanford Bunny). The distance field is dilated by the safety margin d (shaded blue) and several iso-surfaces are illustrated. Three levels ($L = 1, 2, 3$) of the sphere tree are shown: discarded spheres with dashed lines, colliding or close spheres with continuous, filled shapes – darker regions indicate higher collision probability. The inter-object distance p_d between the dilated voxelmap and the pointshell is computed if spheres are identified to be closer than the critical distance p_c . Closest features Q and $S(Q)$ are separated by a distance p (without dilation).



$$\rho = \begin{cases} \left(\frac{\ln(0.01 N_{\mathcal{P},L_c} + e)}{\ln(N_{\mathcal{P}}^c + e)} \right)^2 \frac{1}{0.01 N_{\mathcal{P},L_c}} & \text{if } N_{\mathcal{P}}^c < 0.01 N_{\mathcal{P},L_c} \\ \frac{1}{N_{\mathcal{P}}^c} & \text{otherwise,} \end{cases} \quad (3.48)$$

where $N_{\mathcal{P},L_c}$ is the total number of points that can collide in the critical level L_c and $N_{\mathcal{P}}^c$ is the number of *colliding* points. If the critical level is the last level (i. e., $L_c = N_L$), $N_{\mathcal{P},L_c}$ corresponds to the total number of points in the pointshell ($N_{\mathcal{P}}$).

The exponent 2 and the threshold percentage of colliding points 1% were obtained empirically after several experiments, and the natural number $\ln(e) = 1$ avoids singularities. The heuristic in (3.48) basically divides the force values by the number of colliding points (equivalent to the surface magnitude) if more than 1% of all possible points collide; on the other hand, that scaling is amplified if less than 1% of all possible points collide. Through that amplification, the relevance of collisions consisting of few points (which would have been otherwise faded out) is recovered. Other authors have approached the issue by adapting the stiffness of individual contacts employing a Gauss map of the normal distribution [XB16].

Finally, the last lines in Algorithm 3.1 handle the computational load (explained in Section 3.3.2) and the closest feature on the voxelmap $S(Q)$ (explained in the following subsection).

3.3.1.4 Distance Computation ($p_c \geq 0$)

The input variables d and p_c , and the output variables Q , $S(Q)$, and p_d control and predict the inter-object distance, respectively. Their relationship during an exemplary distance computation process is shown in Figure 3.12. While d artificially widens the

voxelmap reducing the distance between the objects exactly by d , the distance computation threshold p_c controls the accuracy of proximity queries. A value of $p_c = 0$ renders coarse unusable distance values, but, in turn, it demands only one sphere check if objects are far away. On the other hand, very high p_c values could imply checking all spheres and points, subsequently increasing the computational effort.

In practice, the value of the current distance computation threshold $p_c(k)$ is set outside the call of Algorithm 3.1 by slightly increasing the value of the previous real inter-object distance $p(k-1)$:

$$p_c(k) = 1.1 |p(k-1)| = 1.1 |p_d(k-1) - d(k-1)|. \quad (3.49)$$

Assuming the worst case scenario in which objects are approaching each other, the definition in (3.49) leads to close-to-optimum performance, since it (i) assures that the spheres closer than p_c (and the points within them) are going to be visited while (ii) discarding further ones.

Once Q is iteratively improved (lines 14 and 21) in the hierarchy traversal with sphere checks (line 9) extended to distance p_c , $S(Q)$ is computed applying the surface map function in (3.27) to Q in case \mathcal{S} is available, or with the gradient descent function in (3.23) otherwise.

3.3.1.5 Segmented Hierarchical Traverse (Clustered \mathcal{M})

The contact manifold \mathcal{M} is especially interesting for applications that require contact surface information, e. g. in grasp planing [Her15]. It is also possible to perform constraint-based force rendering having the set of colliding points as input, as analyzed in Chapter 4. In this line, reducing the number of possibly colliding points to the most significant ones can be essential. The segmented hierarchical traverse achieves that by re-defining how colliding points are added to the contact manifold \mathcal{M} .

Given a user specified level L (usually $2 \leq L \leq 5$), a unique deepest point is tracked for each of the m clusters in that level L . In other words, the object is divided into m segments and the deepest penetrating point is delivered for each of them. No changes are performed in Algorithm 3.1, but the structure \mathcal{M} internally sorts the added points (lines 12 and 19). This is implemented fixing m elements in \mathcal{M} before the online calls, one for each of the cluster branches in level L ; during collision computation, every time a colliding point increases the penetration value of the previously registered point in its corresponding segment or branch, it is admitted to replace it.

Other handling policies are also implemented in the contact manifold \mathcal{M} , like completely rejecting the inclusion of any point (computationally less expensive), or admitting only the first n points with the highest penetration values from the whole \mathcal{P} .

The segmented hierarchical traverse is compared to the convex decomposition approach [MG09] from the Bullet Physics Engine [Cou03] in Chapter 5. That approach segments the object in m convex hulls that preserve the shape of the object; then, the segmented parts are checked for collision with the GJK algorithm [GJK88]. Each convex segment can deliver a contact manifold consisting of at most one contact point.

3.3.2 Time Critical Level-of-Detail Traverse ($\eta_c < 1$, $q_c > 0$)

In this section, three methods implemented in order to perform time critical queries are explained:

- (i) graceful traverse breaks on an upper level than the leaf level if the previous computational load exceeds a given threshold,
- (ii) filtering out clusters with too low point quality values, and
- (iii) avoiding collision checks in successive cycles if objects are already far apart.

3.3.2.1 Maximum Allowed Computational load (η_c)

The **load value** η is a parameter computed and saved every cycle (lines 28 and 29) which is essential to determine whether successive traverses should stop at upper levels than the leaf level due to reduced time budgets. It consists of the sum of two load ratios related to the amount of spheres and points (omitted in the pseudo-code for clarity):

$$\eta = \omega_c \frac{N_C^v}{N_C} + \omega_p \frac{N_P^c}{N_P^v}, \quad (3.50)$$

where

N_C^v is the number of spheres (or clusters) visited or checked for collision,

N_C the total number of spheres,

N_P^c the number of colliding points,

N_P^v the number of points visited or checked for collision (which are inside the colliding spheres),

and $\omega_c = \omega_p = 0.5$ are the weighting factors chosen for the sphere and point loads, respectively.

Together with the critical load η_c provided in each query, the load η in (3.50) is used to compute the critical or last level L_c that the traverse reaches (lines 1 and 15). If at the beginning of the query (line 1) in the call instant k it is $\eta(k-1) > \eta_c(k)$, the critical level is decreased one unit, and increased if $\eta(k-1) < \eta_c(k)$. Note that

$$\left\lceil \frac{N_L}{2} \right\rceil \leq L_c \leq N_L, \quad (3.51)$$

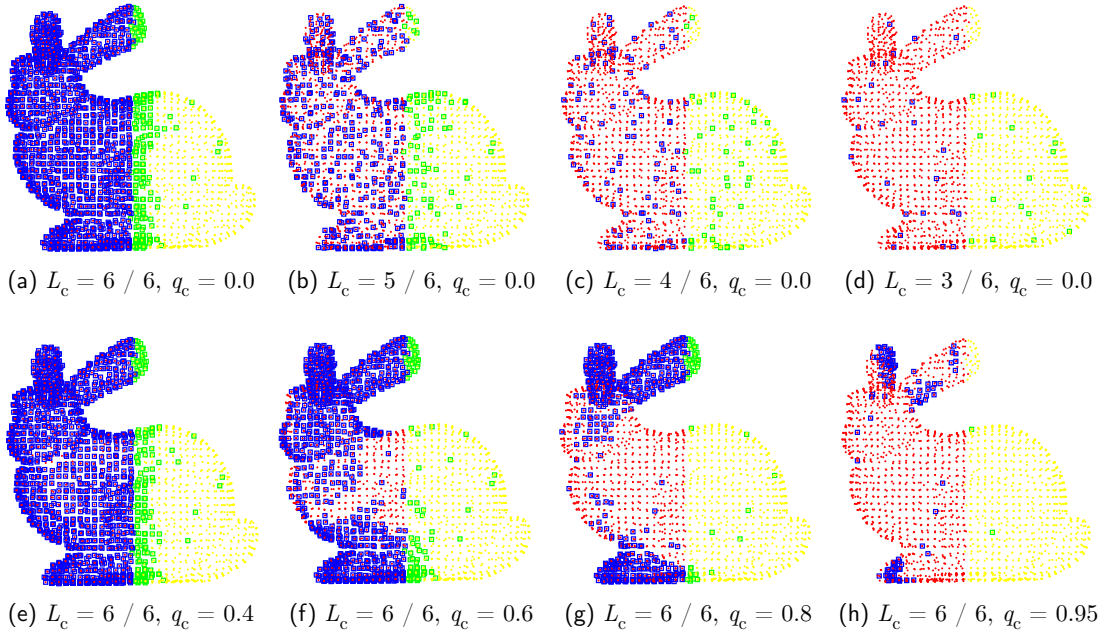


Figure 3.13: Time critical queries with varied load and quality thresholds: A pointshell of the Stanford Bunny ($s = 0.006$ m, 2468 points, 845 clusters, $K = 4$, $N_L = 6$) is partially introduced into the voxelized cube from Figure 3.8 and the points are colored depending on their collision status (projected on XY plane): Yellow, non-colliding points; Red, colliding points; Green, checked but non-colliding points; Blue, checked and colliding points. (a)–(d) Critical level L_c is decreased. Checked points converge to the collision regions. This convergence is less clear as L_c decreases, since the regions covered by the clusters increase and cluster children points must be checked in level L_c (what would be avoided if the algorithm were let to continue deeper in the hierarchy). (e)–(h) Critical quality ratio q_c is increased, allowing to check only clusters with larger gradient variation for higher values. Further descriptive values are given in Table 3.4.

and $L_c = N_L$ in the initialization.

Similarly as in [BJ08], graceful LoD degradation is achieved with this approach. Assuring that a whole level is checked before stopping the traverse guarantees that all points have the same weight in the force computation. Note, additionally, that each upper level (lower L) still samples the whole object with uniform density. This graceful degradation by means of adjusting the last critical level according to the previous computational load is illustrated in the first row of Figure 3.13 for different L_c values. Table 3.4 provides the most important descriptive values related to the figure.

If practical application issues are considered, pre-defined worst case hierarchy traverses can be performed during initialization (e.g., check all cluster spheres and points for collision) in order to predict the critical load η_c for an object pair to match the goal computation time in those worst cases.

Table 3.4: Descriptive values during time critical queries with varied load and quality thresholds. The table is related to the scenarios depicted in Figure 3.13. Parameters from (3.50) are evaluated for each critical level or quality threshold.

	L_c	q_c	Spheres		Points		Points				η
			N_C^y	N_C	N_P^c	N_P^y	Colliding		Visited and ...		
							yes	no	colliding	not coll.	
						red, ●	yellow, ●	blue, □	green, □		
(a)	6	0	664	845	1509	1713	1509	959	1509	204	0.83
(b)	5	0	194	845	387	476	1509	959	387	89	0.52
(c)	4	0	54	845	98	141	1509	959	98	43	0.38
(d)	3	0	14	845	25	40	1509	959	25	15	0.32
(e)	6	0.4	632	845	1431	1606	1509	959	1431	175	0.81
(f)	6	0.6	446	845	1014	1119	1509	959	1014	105	0.72
(g)	6	0.8	247	845	563	605	1509	959	563	42	0.61
(h)	6	0.95	66	845	117	119	1509	959	117	2	0.53

3.3.2.2 Minimum Required Quality (q_c)

Another similar method for time critical calls consists in admitting only clusters in \mathcal{Q} (line 22) that have a relative **quality value** (see Section 3.2.4.1) at least as high as the user specified critical quality q_c , i. e., iff

$$\frac{q_{\max}}{\max\{q_{\max}\}_C} \geq q_c. \quad (3.52)$$

The maximum quality value across all clusters $\max\{q_{\max}\}_C$ is computed before the simulation; the critical quality threshold q_c is registered in \mathcal{Q} at the beginning of every call (line 3), and the current cluster quality q_{\max} is passed when clusters are pushed (line 22). The second row of Figure 3.13 and Table 3.4 show the effect of varying q_c .

3.3.2.3 Spatio-Temporal Coherence

Finally, the last method for time critical queries exploits the **spatio-temporal coherence** [MPT06] characteristic of collision detection. The idea behind it is based on the notion that if two objects are far away from each other, they are going to remain similarly far away in the next cycle. Therefore, if the distance is known and a realistic maximum velocity is assumed (derived from expected human hand movements), a period of time can be easily established in which the objects will not collide; maintaining the process

idle or simply not calling the collision queries during that period of time saves considerably resources. This approach is applied to multi-body scenarios and further discussed in Section 5.2.

3.3.2.4 Discussion

The three approaches that implement time critical queries introduced so far can be used in combination. Limiting the **computational load** with η_c takes advantage of the multi-resolution nature of the point-sphere tree: in the defined breadth-first traverse, the whole object is rasterized with increasing resolution every level, thus improving the contact answer until a load (time) threshold is achieved. In this sense, this approach is sensible when contact forces are in focus, rather than distances between objects. In contrast, requiring a minimum **quality** with q_c is more reasonable for tracking the distance between edge objects. Instead of assuring uniform point densities as in the previous method, non-uniform point densities are simulated, favoring regions with higher geometry variation or less smooth areas.

Avoiding to carry out calls between separated objects due to **spatio-temporal coherence** can satisfactorily lead to thousands of idle cycles, but at the cost of not having any real up-to-date information during those idle cycles. Therefore, this approach is practicable only when force rendering is important, and accurate inter-object distance is not desired.

3.4 Experiments and Results

In previous sections some features of the presented methods have been evaluated independently. This section deals with the assessment of the global proximity and collision computation algorithm from Section 3.3 applied to specific 3D geometries. Two synthetic scenarios have been considered in which distance and collision data are computed and compared varying different factors:

- (i) A sphere (pointshell) is partially introduced into a cube (voxelmap) with the same edge length as its diameter. Figure 3.14 illustrates the scenario and its parameters. A plain pointshell (\mathcal{P} , P) and a point-sphere tree (\mathcal{C} , C) are tested, as well as the three different signed distance computation functions: V_L , V_S , and V_I . The combination yields up to four curves for each of the evaluated contact variables. The results are displayed in Figure 3.15. In this experiment, the behavior of the different data structures is compared; the simple environment enables the straightforward computation of expected or ground truth distance and penetration signals.

- (ii) The Stanford Bunny (pointshell) is translated and rotated towards the Utah Teapot (voxelmap) according to the parameters shown in Figure 3.16. Two resolutions of the point-sphere tree are used ($C(1)$ and $C(2)$), as well as two different resolutions for the layered signed distance computation function ($V_L(1)$ and $V_L(2)$); the maximum and minimum resolutions (or voxel sizes) of each pair have a ratio of 2. The combination yields up to four curves for each of the evaluated contact variables. The results are plotted in Figure 3.17. In this experiment, the effect of varying the resolution is evaluated in a more complex scenario.

These simple experiments are easy to replicate, yet assess some of the most important properties of the collision and force rendering algorithms. It is worth to mention that none of the presented time-critical approaches were activated during the experiments in order to have a more simple and fair quality assessment. In terms of relative size and density of elements (i. e., points and voxels), the following combinations from each scenario were chosen to be equivalent:

- (i) Scenario 1: Point-sphere tree and layered signed distance function – C , V_L (green in Figure 3.15)
- (ii) Scenario 2: High resolution of point-sphere tree and normal resolution of the voxelmap – $C(2)$, $V_L(1)$ (green in Figure 3.17)

It must be considered that these two experiments were carried out following predefined trajectories, without coupling any of the objects to a haptic device; this enables a more controlled and analytical evaluation. Further benchmarks and comparisons of the presented penalty-based haptic rendering algorithm in which human users interact with haptic interfaces are provided later in this work, as additional features are introduced:

- Next Chapter 4 presents a constraint-based force rendering approach that is implemented on top of the penalty-based algorithm introduced in this current chapter; in this line, the force values of both algorithms are collated in Section 4.4.
- Section 5.3 in Chapter 5 presents the integration of the penalty-based algorithm of the current chapter into the physics engine Bullet [Cou03]; the presented algorithm is compared to two approaches available in Bullet.
- Section 5.2 in Chapter 5 presents a virtual assembly framework that features a realistic assembly sequence of car parts into an engine bay; the presented algorithm is employed and evaluated in it.
- Section 6.2 in Chapter 6 presents the results of a user study in which the introduced penalty algorithm is compared to the constraint-based algorithm of next Chapter 4.

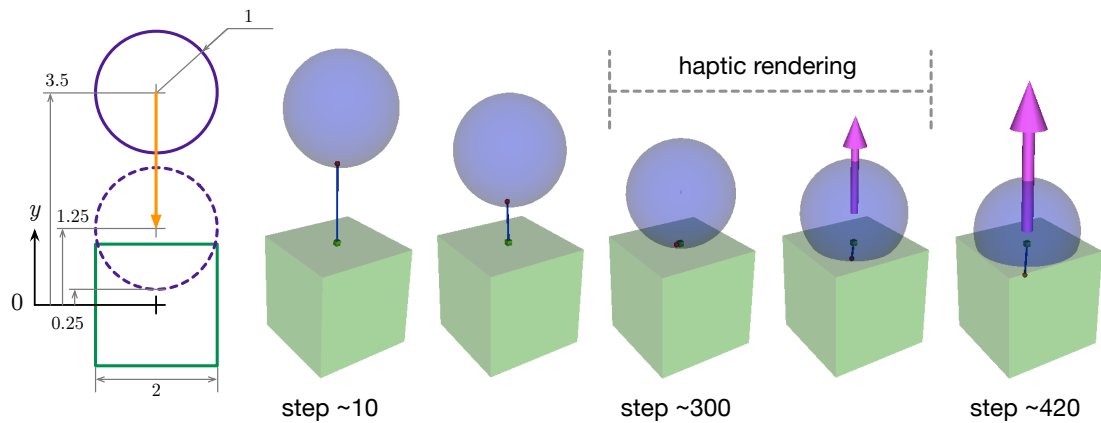


Figure 3.14: Sphere and cube benchmarking scenario (all length sizes in m): a sphere is progressively introduced into a cube yielding to some extent predictable contact data at each step; Figure 3.15 plots the behavior of the different explained methods in this scenario. Closest discrete points are represented with small red and green features and joined with a blue line. Collision forces and torques are represented with magenta and cyan (imperceptible) vectors, respectively. The pointshell of the sphere has a resolution $s = 0.025$ m (16299 points, 5607 clusters, 4 children per cluster, 8 levels). The cube is voxelized with $s = 0.04$ m ($151 \times 151 \times 151$ voxels taking into account the added 50 outer layers).

3.4.1 Discussion of Scenario 1: Sphere and Cube

This subsection discusses the curves in Figure 3.15. Probably the most important values are the penetration error, force, torque, and computation time for different combinations. It is worth to mention that the regions in which haptic rendering should occur have been bounded between vertical dashed, gray lines. In fact, if device stiffness and damping were correctly set in a haptic interaction with two rigid bodies as in this scenario, the resultant penetration region should be much smaller, probably about a tenth of the area marked with dashed lines, and around step 300. Overall, all curves present a reasonable and in most cases expected shape, and the computation time remained always under the 1 ms threshold in the mentioned haptic rendering region.

Penetration Error \diamond The penetration error diagram (third) results from combining the first relative position y (ground truth) and the second computed penetration plots. Ideally, the penetration error should be 0 voxels. The interpolation method (black) produces the best result (highest accuracy), followed by the layered distance function (green) and the surface point distance function (blue). This last one presents bumps at the surface boundary and in inner voxels which are probably due to errors in the model originated in the generation of the structure \mathcal{S} according to the heuristic introduced in Section B. Those errors do not appear in all models, as shown, for instance, in Figure 3.7; in any

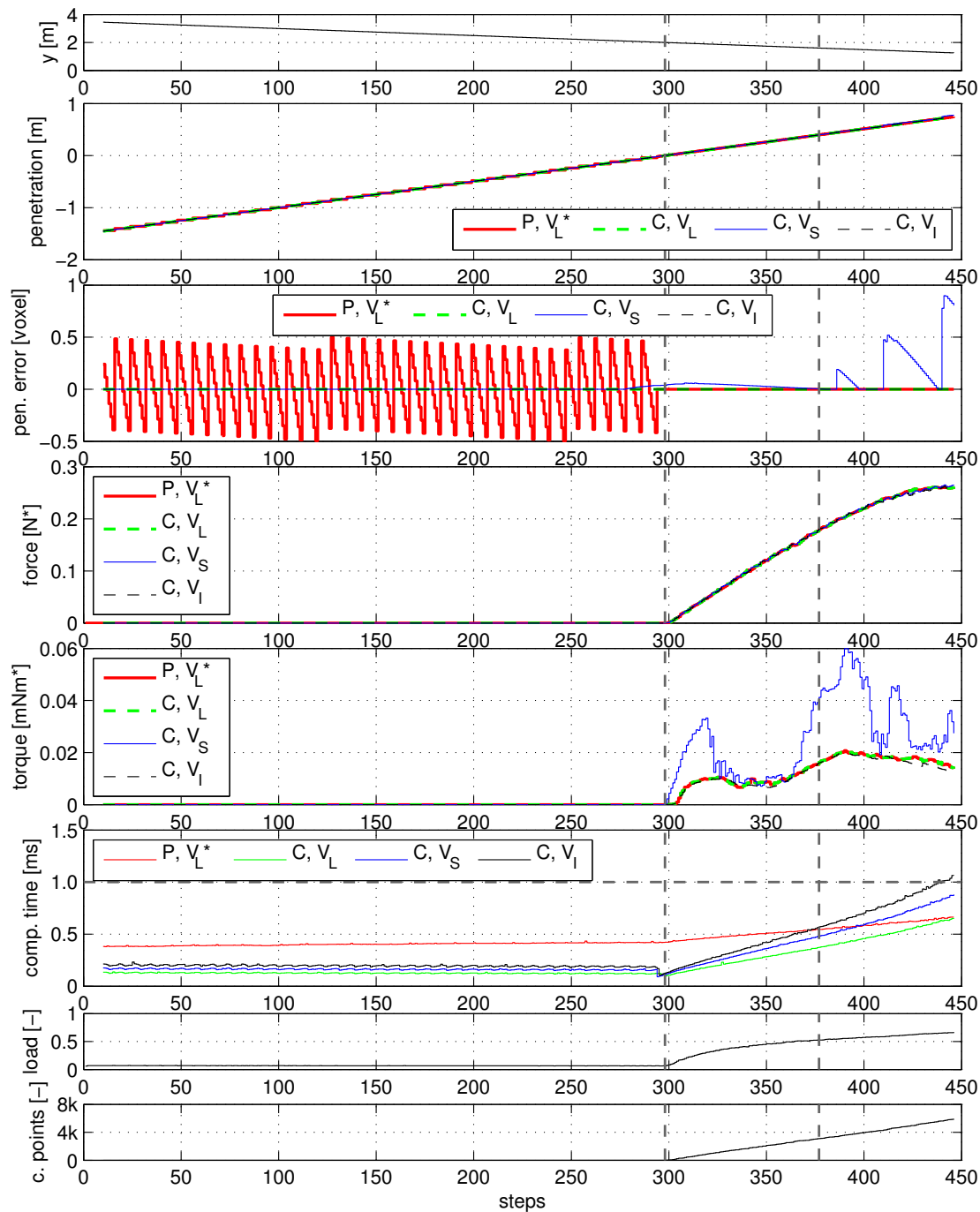


Figure 3.15: Cube and sphere benchmarking results; the scenario is described in Figure 3.14. General values applicable to all or most methods are plotted in one curve for each diagram (sphere position, algorithm load, and number of colliding points). Values of specific methods are plotted in several curves and appropriately designated in the legend. P, V_L^* : plain pointshell (\mathcal{P}) and plain voxelmap (\mathcal{V}), accessing the signed distance function V_L defined in (3.25) – only the global distance component is computed in case there is no collision. C, V_L : hierarchized pointshell (\mathcal{C}) and plain voxelmap (\mathcal{V}), accessing the signed distance function V_L defined in (3.25); C, V_S : hierarchized pointshell (\mathcal{C}) and enhanced voxelmap (\mathcal{S}), accessing the signed distance function V_S defined in (3.26); C, V_I : hierarchized pointshell (\mathcal{C}) and enhanced voxelmap (\mathcal{W}), accessing the signed distance function V_I defined in (3.28). The region in which haptic rendering should happen is bounded between two dashed vertical lines. The force and torque magnitudes are marked with * because these values usually need to be amplified by a gain before application. The used computer was an Intel(R) Core(TM) 2 Quad with CPUs at 2.66GHz and running Open Suse 42.2 Leap 64B (not realtime).

case, this a subject for further investigation. The layered distance function in combination with the plain pointshell (red) has the same penetration values as its counterpart with the hierarchized pointshell (green) in penetration configurations, but it presents a saw shape with the amplitude of a voxel unit. That is because the layered distance function V_L^* from the red curve was modified to deliver distance values only with the global component in (3.25), neglecting the local distance term. Hence, the zig-zag features the error introduced when the penetration into the voxel is ignored, as done in the early versions of the VPS algorithm [MPT99].

Force and Torque \diamond While few differences are appreciable between the force values of different combinations, the torques obtained with the surface point approach (blue) differ from the values of the rest of the combinations. That difference comes from the penetration error discussed in the previous paragraph; in contrast to the penetration values, force and torque values correspond not only to one point, but to the sum of values of all points plotted in the last diagram of Figure 3.15. Additionally, it needs to be taken into account that the units marked with * (i. e., N^* and Nm^*) refer to unscaled magnitudes: in real interactions, device stiffness and damping values would have to be applied to those values, obtaining signals up to 2–3 orders of magnitude larger, depending on the haptic interface. It is also worth to mention that in this scenario no torques should appear ideally; however, due to imperfections in the triangulation of the sphere and the distribution of the points, small values arise. Note that most values lie below $10 \mu Nm^*$ in regular situations for the haptic rendering region, though.

Computation Time \diamond The effect of using a point-sphere tree (all except red) over the plain pointshell (red) is clear in terms of computation time: a hierarchy leads to $\sim 3\times$ lower computation times when distances are computed in collision-free configurations and up to $\sim 30\%$ smaller time values in worst case overlap configurations (boundary of haptic rendering region). This change in the improvement ratio is due to the cost associated to the bounding volume traverse, as explained in Section 2.3.2.5. The interpolation method (black), although the most precise, is the one which takes longest for computing the values, as expected. The layered distance function (green), on the other hand, presents the smallest computation times, while achieving similar error values as the interpolation method. Although the accuracy could vary with different geometries and the same resolution, the time should be similar for the same amount of colliding points and spheres.

Conclusions \diamond The previous results can be summarized as follows:

- (i) Signed distance values computed with the layered function (V_L) are close to the most accurate ones provided by the interpolation function (V_I); while the surface point function (V_S) presents better accuracy in other irregular geometries (see Figure 3.7), it might present higher errors if the heuristic used for building the models fails.
- (ii) The force and torque values behave closely to the expected ones; the discrete nature of the models could lead to force and torque residuals, but these are probably imperceptible by users.
- (iii) The computation time is significantly lower for distance, penetration and force computation if the point-sphere hierarchy is used, even in worst case configurations. The time improvement is expected to be even better if time-critical approaches are used.

3.4.2 Discussion of Scenario 2: Stanford Bunny and Utah Teapot

This subsection discusses the curves in Figure 3.17. As in the previous experiment, penetration values, forces, torques, and computation times are analyzed. This experiment is fully synthetic: no such large penetrations and trajectories would be achieved if a haptic device were coupled to one of the objects. Overall, all curves present a reasonable and in most cases expected shape, and the computation time remained always under the 1 ms threshold.

Penetration and Penetration Difference \diamond Distance and penetration signals seem to be similar at an overview scale (second diagram); the penetration difference plot (third) results from subtracting the values of the combination with both finest resolutions (black) to the other three; the difference is scaled to the finest voxel size of the voxelmap ($s = 1$ mm). Few patterns are visible, except that the combination of resolutions equivalent to scenario 1 (green) presents the highest peaks of almost ± 5 voxels. That is due to the decrease of the resolution of the voxelmap.

Force and Torque \diamond The torque-force ratio is considerably larger compared to the one in the previous experiment, meaning that in the current scenario torque values are significant and cannot be neglected. The resolution of the voxelmap makes a difference in both force and torque signals around step 250 (ear of inverted bunny penetrating teapot). Recall from Figure 3.8 that the ear has a high surface normal variation; modeling the bunny with half the resolution implies re-sampling it with a $4\times$ lower point density, which leads to lost normal directions in areas with high curvature as the ear. This could explain why those differences in magnitude appear in the vectorial space when the resolution of

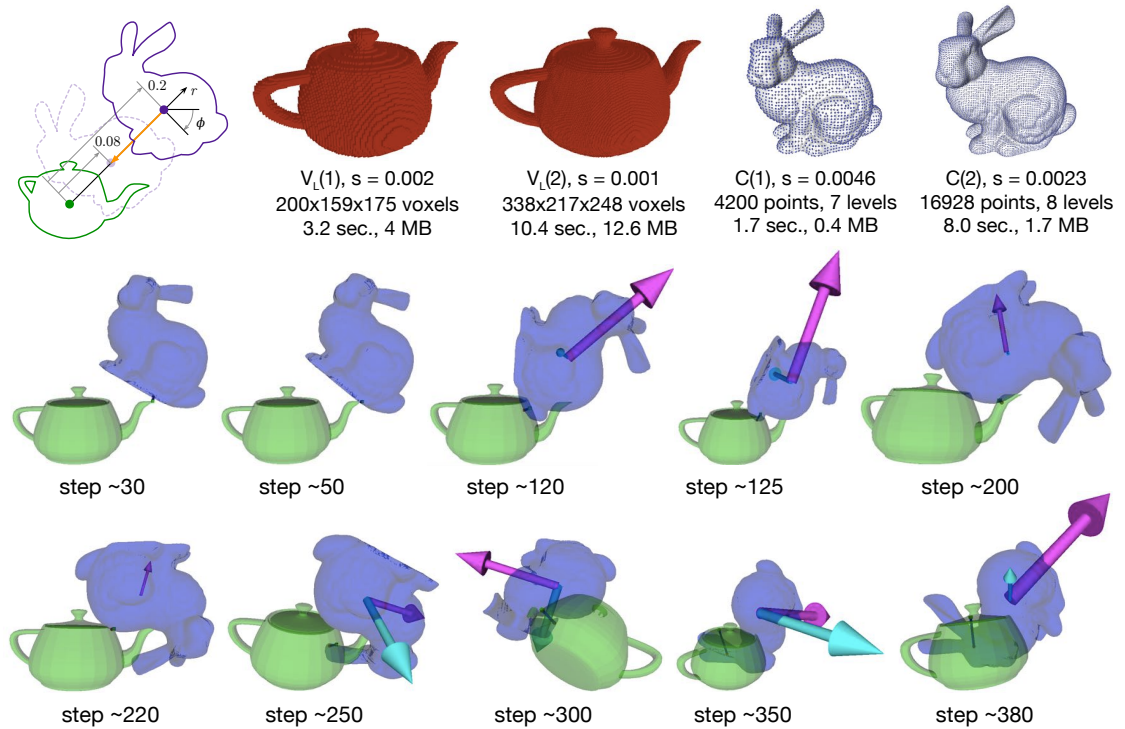


Figure 3.16: Bunny and Teapot benchmarking scenario (all length sizes in m), data structures (including generation time and file size), and snapshots: a Stanford Bunny is progressively introduced into a Utah Teapot by translating (r) and rotating (ϕ); Figure 3.17 plots the behavior of the proximity and collision computation algorithm under different resolutions in this scenario. Closest discrete points are represented with a small red and green features and joined with a blue line. Collision forces and torques are represented with magenta and cyan vectors, respectively. Note that the torque vectors have a scaling factor of $10\times$ compared to the forces in order to properly visualize them. All pointshells have 4 children per cluster and all voxelmaps 50 outer layers. The voxelmap-pointshell combination $V_L(1) + C(2)$ is equivalent to the cube and sphere scenario in Figure 3.14 in terms of relative resolution.

the pointshell is altered. Visually, the combination with the finest resolution (black) has the smoothest curves, as expected, followed by the combination equivalent to scenario 1 (green), and then the rest; in this sense, the resolution of the pointshell seems to be the most relevant one, both for the quality or smoothness of curves and the computation time, as explained in the next paragraph.

Computation Time \diamond The effect of the resolution associated to the pointshell is notable, as expected: halving the voxel size of the pointshell (i. e., double resolution, $4\times$ more points) leads to $5 - 6\times$ higher computation time values in worst case configurations (around step 325); however, further benchmarking is necessary to assess the general relevance of this quantitative value. More interestingly, there seems to be an unexpected

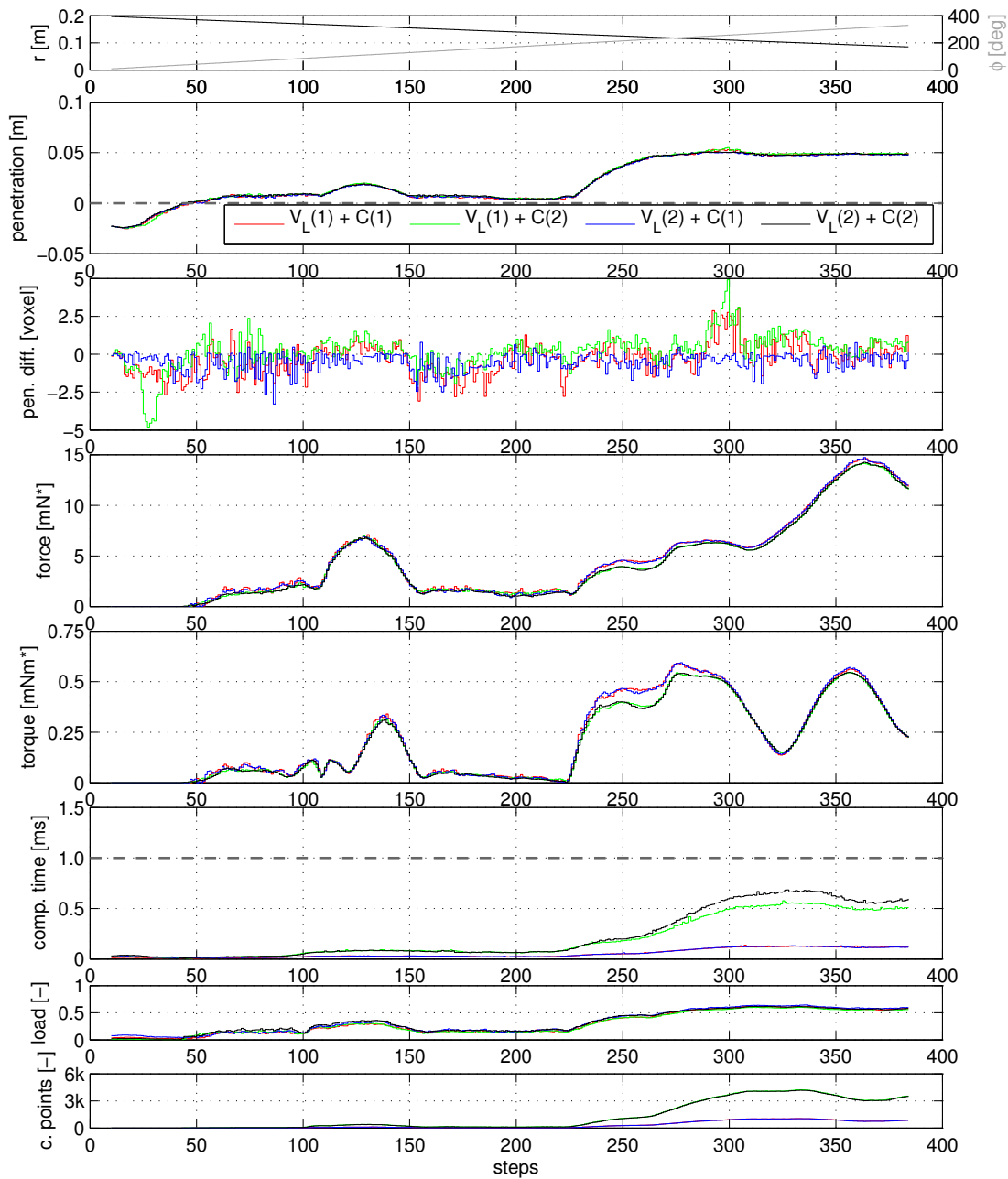


Figure 3.17: Bunny and Teapot benchmarking results; two resolutions for each of the data structures are used, as shown in the scenario description from Figure 3.16. This results in four curves of combined resolutions, as properly tagged in the legend. The computation with point-sphere trees (\mathcal{C}) and layered signed distance functions (V_L) is plotted only. In the case of the penetration difference diagram (third from above), the penetration of the combination with finest resolutions ($V_L(2) + C(2)$) is subtracted to the penetration of each of the other combinations and divided by the finest voxel size ($s = 0.001$ m). The force and torque magnitudes are marked with * because these values usually need to be amplified by a gain before application. The used computer was an Intel(R) Core(TM) 2 Quad with CPUs at 2.66GHz and running Open Suse 42.2 Leap 64B (not realtime).

effect on the time caused by the resolution of the voxelmap. As explained in Section 3.2.1, the voxelmap is implemented as an `std::vector` array in C++, and accessing a random voxel value of a point should run with $\mathcal{O}(1)$ complexity, independently of the resolution used for the voxelmap, if the whole array is assumed to be allocated in a unique memory chunk [Ric14]. Yet, in the plot, a voxelmap with half voxel size (i. e., roughly $8\times$ more elements) requires 20% larger computation times when roughly 4000 points are colliding (around step 325). In practice, the consequences of this fact are probably negligible, since configurations with such high penetrations are automatically avoided through displayed forces and the time-critical methods should prevent from having more than 3000 colliding points, independently of the area percentage in contact.

Conclusions \diamond The previous results can be synthesized as follows:

- (i) The layered signed distance value V_L works robustly even with coarse resolutions if high resolutions are used for pointshells; the most notable errors due to coarse voxelmap resolutions appear probably in the penetration values.
- (ii) The resolution of the hierarchized pointshell significantly affects the computation time, as expected, but can be controlled with time-critical methods; higher resolutions of the voxelmap led to unexpected higher computation time values in worst case configurations, but that effect can be probably neglected in regular scenarios.

3.5 Summary, Conclusions, and Perspectives

This chapter presented the theoretical definition and an implementation basis of a *penalty-based* haptic rendering approach able to render six-DoF forces in 1 kHz between arbitrary geometries. The method follows the principles of the Voxelmap-Pointshell (VPS) algorithm [MPT99]; in that sense, it could be considered a complete re-implementation of the VPS, from the scratch, with novel improvements. The chapter covers the algorithms for data structure generation and their properties, the proximity and collision computation queries, and reproducible experimental results considering varied parameters that benchmark different functionalities.

Signed distance fields embedded in voxelmaps and point-sphere trees or pointshells which sample objects with several point densities are generated in few seconds and used during distance and collision computation. Voxelmaps contain sets of distance values at different accuracy levels, being possible to compute the distance of a given point with respect to the surface of the object modeled with three different functions. Point-sphere trees are traversed in a breadth-first manner, improving in each tree level the delivered contact output. Additionally, time-critical queries are possible within that traverse scheme, according to the minimum required depth in the hierarchy (i. e., point

density) or culling point clusters with low additional geometry information. Along these lines, and in contrast to previous works, the new contributions presented in this chapter can be summarized as follows:

- (i) fast and robust data structure generation algorithms,
- (ii) multi-resolution properties for both data structures and methods to exploit it,
- (iii) compact point-sphere trees with minimally bounding spheres, which are built bottom-up exploiting local information,
- (iv) a unified collision, signed distance (i. e., distance or penetration), and penalty force computation algorithm, and
- (v) time-critical queries that are able to filter out regions with low geometrical variation, or that carry out conservative coarse checks for situations with low time budgets.

All signed distance computation functions for different LoDs were compared between each other and against ground truth values, validating their performance. Errors vary from negligible to the voxel size in different configurations, being bounded to the resolution used. Regular and time-critical hierarchy traverses improve considerably the required computation time compared to the classical VPS approach and are able to achieve complete loops with tens of thousands of points in less than 1 ms.

As far as the data structures are concerned, future work should address GPGPU implementations for generation, since the explained methods for polygonal model conversion practically break down to the same processes at voxel or point level. In contrast, GPGPU is not that straightforward for realtime proximity and collision queries if hierarchical traverses are to be used. Additionally, researching into methods for collision computation between deformable objects that still behave with a similar efficiency as the presented approach is also interesting.

The *penalty-based* haptic rendering algorithm presented in the current chapter was integrated into a virtual assembly simulation platform and into the physics engine Bullet [Cou03], as thoroughly explained in Chapter 5. Additionally, further benchmarking experiments and results of comparisons to two other algorithms from Bullet are provided. Chapter 6 presents user studies which compare and validate the current *penalty-based* approach and *constraint-based* force rendering method from the following Chapter 4.

Chapter 4

Constraint-Based Force Rendering

Collision detection and force computation between complex geometries are essential technologies for virtual reality and robotic applications, as shown in Chapter 2. Penalty-based haptic rendering algorithms as the VPS re-implementation presented in previous Chapter 3 provide a fast collision computation solution; however, these methods cannot avoid the undesired interpenetration between virtual objects, and have difficulties with thin non-watertight geometries. In contrast, *god object* methods or constraint-based haptic rendering approaches have shown to solve this problem, but are typically complex to implement and computationally expensive.

This chapter presents an easy-to-implement *god object* approach applied to six-DoF penalty-based haptic rendering algorithms. Contact regions are synthesized to penalty force and torque values and these are used to compute the position of the *god object* proxy on the surface. Then, the pose of this surface proxy is used to render stiff and stable six-DoF contacts with friction. Independently of the complexity of the used geometries, the implementation of the presented method runs in only around $5\ \mu\text{s}$ and the results show a maximal penetration error of the resolution used in the penalty-based haptic rendering algorithm.

In other words, the methods introduced in this chapter extend the more general proximity and collision computation algorithm from the previous chapter for more robust and realistic force rendering: object interpenetration is minimized, visualizing the moved probe proxy on the surface, and stiff contacts are rendered even with thin shells.

This chapter uses parts from the following peer-reviewed publication written by the author of this work: [SH16].

4.1 Introduction

Penalty-based methods detect the overlapping error (usually penetration or volume) to either compute a force or simulate a plausible motion upon contact using the Newton-Euler equations. On the other hand, constraint-based approaches prevent overlap between objects. Basically, the user controls the *device pose* or the *haptic tool*, but a *proxy* or so-called *god object* is visualized. Even though the device object would go through the other geometry, that *god object* would always remain on the surface boundary. In this sense, the focus lies on determining the constrained movement of the object. Forces are rendered out of the difference between the *device* and the *proxy* pose.

Constraint-based methods are in narrow relationship with the concept of *virtual coupling* [CSB95]. Usually some type of *virtual coupling* is performed during realtime simulations even with penalty-based haptic rendering algorithms, i. e., the virtual object does not match the device pose. That difference can be realized by making the device follow the virtual object coupled with a spring (and a damper), or, in the case of *god object* methods, it is given by the proxy constrained to the surface. The displayed force is related to the coupling distance between the device and the virtual object.

4.1.1 Related Work

To the best of my knowledge, Zilles and Salisbury coined the term *god object* for their constraint-based three-DoF haptic rendering algorithm [ZS95] in 1995*. This method gives rise to a series of works based on optimization approaches. In case of contact, in order to obtain the pose of the proxy on the surface, the authors minimized the energy of a spring between the penetrating point linked to the haptic device (known) and a parametrized proxy point, constrained to the collision plane. The problem is easily solvable by using Lagrange multipliers. Similarly, Ruspini et al. [RKK97] minimized the distance between the penetrating point and the proxy, but constrained the region outside of several contact half-planes. In both cases, the idea is related to the Gauss' principle of least constraints [Gau29], which states that the motion of a mechanical system satisfies the minimum of the differences' norm between the constrained (proxy) and unconstrained (haptic device) accelerations. Redon et al. [RKC02b] analyzed the advantages of this principle for rigid body simulations and Ortega et al. [ORC07] applied it for six-DoF haptic rendering. The method proposed by the last authors computes the force rendering in a separate asynchronous thread in order to achieve the 1 kHz update rate necessary for haptic interaction [BS02], since the used continuous collision detection [RKC02a], in

*However, the expression *god object* had already been used by Dworkin and Zeltzer [DZ93] with another meaning: they referred to objects that move at their own will in a physics simulation, e. g., objects moved by a human hand.

combination with the god object pose simulation, exceeds that performance threshold when contact regions increase. This decoupling opens up the possibility to testing other collision detection methods and experimenting with simplifications in the proxy pose computation.

In recent years, several optimization-based approaches have also been presented. Chan et al. [CCBS11] applied the Gauss' least constraints principle for six-DoF haptic rendering using volumetric medical imaging and point clouds formed by unordered object vertices. Rydén and Chizeck [RC13b] applied the same basic principle of rigid body mechanics to streamed point clouds and voxelmaps to obtain six-DoF haptic rendering. Wang et al. [WZZX13] adapted the quadratic programming approach from [ORC07] to the six-dimensional configuration space. These authors use a sphere-based collision detection and then minimize the distance between the proxy and the device.

It is also possible to constrain the god object to the surface without explicitly formulating the task as an optimization problem. In this sense, Salisbury and Tarr [ST97] presented a very interesting three-DoF haptic rendering algorithm for implicit surfaces. The method works as follows: first, when the device point penetrates the surface, its closest surface point is detected with a deepest descend algorithm, and a support tangent plane is computed on it; then, in each cycle, the projection of the device point on the tangent plane is computed, which leads to the closest surface point using the same deepest descend algorithm. The tangent plane is updated every cycle. Displayed forces are related to the distance between the device and the surface point. This algorithm has been exploited in recent years for three-DoF haptic interaction with streamed point clouds by Leeper et al. [LCS12], and with deformable volumetric medical image data by Chan et al. [CBS13]. The heuristic approach presented in this thesis works with a similar idea as the one presented by Salisbury and Tarr for six-DoF haptic rendering.

Besides collision forces, friction is an important contact phenomenon which contributes to manipulation realism, particularly in virtual interactions with haptic feedback. The reader is referred to [ASB07] for a brief but thorough glimpse on basic concepts related to this topic. That work describes and presents simulation results of several friction models applied to a one-DoF system. Additionally, most important phenomena, model properties, advantages and disadvantages are discussed.

Unfortunately, many collision, movement and friction simulation methods from computer graphics such as the work in [KEP05] usually require longer computation times than the 1 kHz necessary in haptics. Therefore, simplifications or heuristics are required. Hayward et al. [HA00] presented a very complete three-DoF friction model suited for haptic rendering. They improved the Dahl friction model cancelling the drifting effect and provided a set of useful approximations. Their model yields the four friction regimes observed in physical reality: sticking, creeping, oscillating, and sliding. Harwin and

Melder [HM02] presented a three-DoF friction computation method similar to the one introduced in [ST97]. The method is easy to implement and applied upon the *god object* algorithm presented in [ZS95]: a cone is placed on the penetrating *device* point and the *proxy* is allowed to move until the boundary of the intersection between the cone and the surface. This approach allows for static (dry) and kinetic (sliding) friction. Kawasaki et al. [KOKM11] extended the previous approach to six-DoF. Their method is able to compute friction moment based on the torsion angle between the *god* and *device* reference frames. Additionally, they implemented the model in a hand-finger force feedback device to provide finger torque friction and conducted a user study on torque friction perception.

4.1.2 Contributions

The presented heuristic operates in the configuration space (six-dimensional pose) of rigid bodies with arbitrary geometry and provides with six-DoF constraint (frictional) forces and correct *proxy* pose simulation with 1 kHz. Upon contact, the *proxy* is constrained to the surface using the forces and torques and the penetration depth computed by a penalty-based algorithm. These penalty forces inherently model contact geometry and, thus, restrict the motion of the object. The contributions (and the overview of the chapter) are summarized as follows:

- A six-DoF *god object* simulation and constraint force computation heuristic which is fast, robust and easy to implement on any penalty-based haptic rendering algorithm that provides signed distances (Section 4.2).
- A six-DoF friction model applied to the *god object* heuristic that comprises static, kinetic, and viscous friction regimes (Section 4.2.7).
- Experimental results that show the behavior of the presented method in several usual and worst-case scenarios (Section 4.4).

As it is concluded in Section 4.5, following the provided implementation steps, it is possible to easily convert virtually any penalty-based haptic rendering algorithm to be a constraint-based approach which benefits from the advantages of both paradigms: ease, speed, stability, and stiffness.

4.2 God Object Heuristic

This section presents step by step the *god object* simulation and force rendering method giving implementation details. Nevertheless, initializations and division-by-zero, satura-

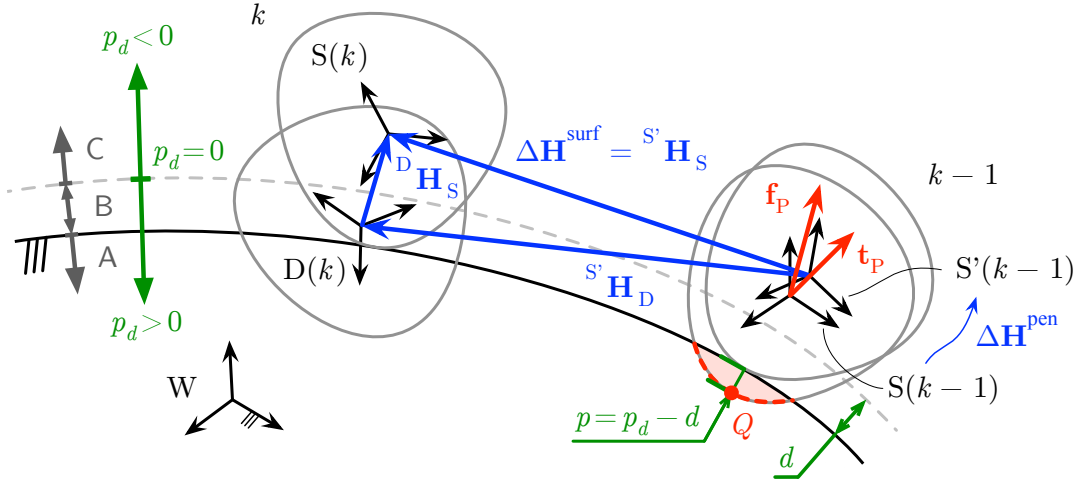


Figure 4.1: Overview of the *god object* simulation in two consecutive time steps $k - 1$ (previous) and k (current). Main frames corresponding to the *world* (W), the *device* (D) and the *proxy or god object* that remains on the *surface* (S) are displayed. In the presented method, first the previous *surface* frame is corrected to $S'(k - 1)$. Then, the motion from this $S'(k - 1)$ to the current *device* pose $D(k)$ is constrained with the force and torque values $\mathbf{f}_P, \mathbf{t}_P$ computed by the penalty-based collision detection algorithm. This yields the new current *god object* pose $S(k)$.

tion, and similar checks are omitted for the sake of clarity. Furthermore, the two-object scenario is considered: the first object is moved by the user via the haptic device with respect to the second one; if relative movements are observed, one can assume the second object stands still without loss of generality – although it may actually be moving.

As convention, bold capital symbols (\mathbf{H}) denote homogeneous transformation matrices in $\mathbb{R}^{4 \times 4}$, bold small symbols (\mathbf{x}, \mathbf{h}) vectors in \mathbb{R}^3 or \mathbb{R}^6 , and small italic symbols (p) scalars in \mathbb{R} . Points, lines and surfaces in \mathbb{R}^3 are denoted with capital italic symbols (P, L) and a vector between two given points P and Q is denoted \overrightarrow{PQ} . In the case of poses (translation and rotation), the matrix representation is used for homogeneous coordinate transformations and the vector representation for all other transformations. Values are transformed from one representation to another with functions like `setMatrix()`, `getRotation()` or `getTranslation()`. Additionally, a transformation from the coordinates W to D is denoted ${}^W\mathbf{H}_D$. All values correspond to the current cycle (k) except when properly indicated (e. g., $\mathbf{H} = \mathbf{H}(k)$ vs. $\mathbf{H}(k - 1)$). The reader is referred to the Notation appendix (page 271) for further details on conventions and used symbols.

As shown in Figure 4.1, there are three main frames:

W The *world frame*, which will be considered fixed in the center of mass of the still

object.

- D The *haptic device frame*, which corresponds to the end-effector of the device moved by the user.
- S The *proxy, god or surface frame*, which corresponds to the object that remains on the surface. It is important to note, however, that a small penetration can occur due to the error committed in the previous cycle. Therefore, this frame is corrected to solve its penetration p , which leads to S' .

Also, the *body frame* B is used when deducing mass distribution properties of the object; the original geometry is supposed to be defined in this coordinates, located in its center of mass G . In the same line, the *eigen frame* E results from performing a principal component analysis of the body.

The goal of the method is to compute a constrained proxy pose ${}^W\mathbf{H}_S$ of the moved object with respect to the other (still) object given the coordinates of the haptic device's end-effector ${}^W\mathbf{H}_D$. ${}^W\mathbf{H}_D$ is the pose of the moved object commanded by the device, which can penetrate the other object. On the other hand, ${}^W\mathbf{H}_S$ is the pose of the *proxy or god-object* which tries to remain on the surface in case of collision. The computation of the *proxy* pose constrained to the surface results from restricting the transformation from the previous *proxy* pose to the current device pose with the penalty contact forces related to the *proxy*.

Figure 4.2 gives an overview of the whole procedure that is repeated every haptic cycle (1 ms). All eight steps depicted in it are described in detail in their respective subsections. A brief summary is given here to convey a global idea of the procedure:

- #1 Penalty Contacts (Section 4.2.1): Penalty-based collision detection is performed using the previous *god object* pose. An important requirement is that one object is slightly dilated with a *safety margin* d to avoid real penetration between the objects approaching each other and to obtain less noisy, more robust collision forces. The value of d could be optimized online, but it is fixed to $d = 3$ mm in the simulations after empirical trials, since it already produces a stable behavior. This step yields the signed distance or penetration value p_d and the penalty forces \mathbf{f}_P and torques \mathbf{t}_P .
- #2 Correction Step (Section 4.2.2): If the *god object* is penetrating ($p_d \geq d$) in the previous cycle, its corrected non-penetrating pose is computed, minimizing the error introduced in the previous cycle. This step is the one that required the longest section in this work, but, still, it is the one with the fastest computation times, since analytical correction formulae for any geometry and contact configuration are derived.

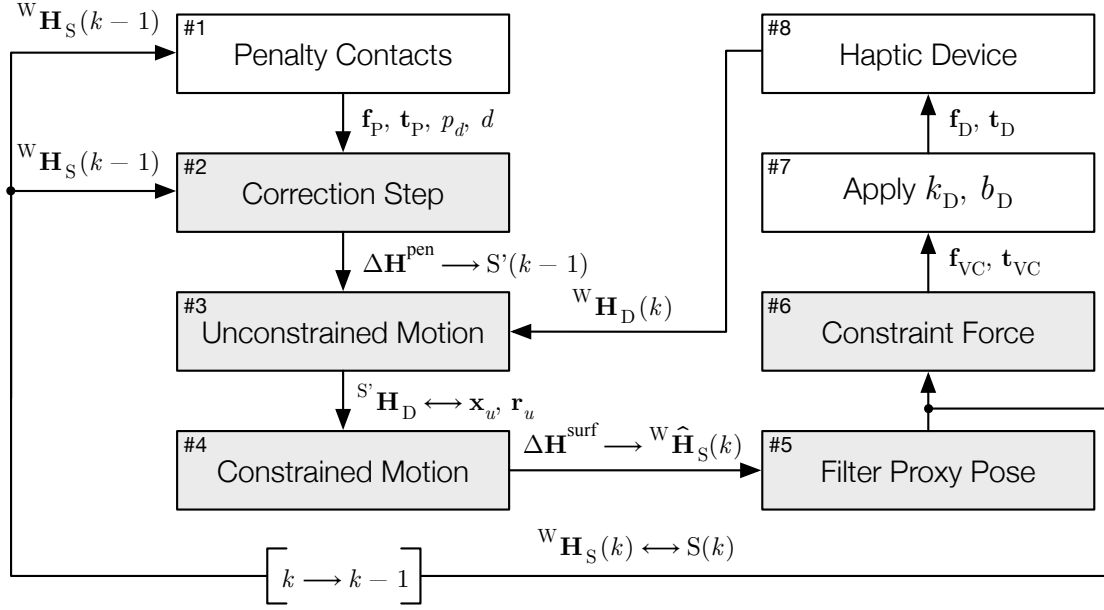


Figure 4.2: Workflow of the *god object* simulation and force rendering method. Shaded boxes #2 – #6 are core steps that define the presented approach; the other steps could be changed without altering considerably the result, particularly the presented approach is suited for other penalty-based contact rendering algorithm (step #1). The whole procedure is repeated every 1 ms, being the contact computation (step #1) the one which lasts longer. Note that the procedure is fed with the *god object* pose of the previous cycle ${}^W\mathbf{H}_S(k-1)$, as well as the current device pose ${}^W\mathbf{H}_D(k)$.

- #3 Unconstrained Motion (Section 4.2.3): The movement of the *god object* is computed as if no collision constraints were present.
- #4 Constrained Motion (Section 4.2.4): The unconstrained motion is corrected with the contacts computed in step #1. The friction is also computed in this step #4 in the object configuration space (Section 4.2.7). The constrained motion leads to the current (unfiltered) *god object* or *proxy* pose.
- #5 Filter Proxy Pose (Section 4.2.5): The current *god object* pose is smoothed with a low pass filter.
- #6 Constraint Force (Section 4.2.6): Constraint forces $\mathbf{f}_{VC}, \mathbf{t}_{VC}$ are proportional to the difference between the current *device* and the *god object* pose, or, in other words, linear to the forbidden movement performed by the *device*. Friction forces are intrinsically considered (Section 4.2.7).
- #7 Apply k_D, b_D (Section 4.2.6): Stiffness and damping factors are multiplied to the constraint forces in order to achieve the desired hard contact on the device while still being stable.

#8 Haptic Device (Section 4.2.6): Device forces \mathbf{f}_D , \mathbf{t}_D are commanded to the haptic device and the pose of the end-effector is read (every 1 ms).

The coordinates of the deepest colliding point Q are unknown but its penetration p_d is provided by the penalty-based collision computation. Note in Figure 4.1 that three regions are distinguished for its value:

- A, $p_d \geq d$: There is overlap between the objects. If the *unconstrained* motion of the *device* frame moves in opposite direction of the penalty forces \mathbf{f}_P and torques \mathbf{t}_P , it must be *constrained* to the surface.
- B, $0 < p_d \leq d$: There is no overlap between objects but the deepest colliding point Q is inside the *safety layer*, which has a width d over the surface. The approach is similar to the previous case, except for slight modifications in several steps, properly indicated.
- C, $p_d < 0$: There is no overlap between objects. In this case, the *god object* pose is the *device* pose, $S(k) = D(k)$, and therefore, there is no constraint coupling force to display, \mathbf{f}_{VC} , $\mathbf{t}_{VC} = \mathbf{0}$.

The cases in which Q lies on either A or B are considered, since the last case of the region C has the mentioned trivial solution.

4.2.1 Penalty-Based Contact Computation (#1)

The first step is accomplished performing a penalty-based collision computation based on the Voxelman-Pointshell (VPS) algorithm [MPT99] presented in the previous Chapter 3.

Given a point P from the pointshell, its signed distance (or penetration) value is computed according to (3.25):

$$V(P) = V_L(P) = \xi_V s v(P) + \mathbf{n}^T \mathbf{d} + d, \quad (4.1)$$

where $\xi_V = \frac{1}{2}(1 + \sqrt{3})$, s is the voxel edge size (constant in uniform grids) and \mathbf{d} the vector from the point to the center C of the voxel where it is located. The *safety distance* d dilates the voxelmap artificially, as previously introduced. This margin is used to predict collision and restitution constraints with respect to the movement, as explained in later sections.

As explained in Chapter 3, all points with $V(P_i) > 0$ are colliding with the dilated *voxelman*. The deepest colliding point Q has a penetration of $p_d = \max_i \{V(P_i)\}$. Additionally, single penalty forces and torques (expressed in the center of gravity G) associated to each points with $V(P_i) > 0$ are defined as

$$\mathbf{f}_i = k_P V(P_i) \mathbf{n}_i, \quad \mathbf{t}_i = \overrightarrow{GP_i} \times \mathbf{f}_i. \quad (4.2)$$

The total penalty force $\{\mathbf{f}_P, \mathbf{t}_P\}$ is the sum of all single forces $\{\mathbf{f}_i, \mathbf{t}_i\}$. These values are already penalty forces that can be displayed to the user. However, penalty-based haptic rendering has the disadvantages mentioned in Section 4.1.

A constraint-based approach could be developed using the whole contact manifold computed in this step. Nevertheless, the goal is to make the presented *god object* simulation method available for any penalty-based collision computation algorithm other than the one presented in this work; therefore, only the most common four values are used: $\mathbf{f}_P, \mathbf{t}_P, p_d, d$. Any algorithm able to provide them can be used instead of the reimplementation of the VPS from Chapter 3.

4.2.2 Correction of the Previous Proxy Frame (#2)

In this section the step $\Delta \mathbf{H}^{\text{pen}}$ necessary to obtain the corrected *surface* frame $S'(k-1)$ out of the previous *god object* pose ${}^W \mathbf{H}_S(k-1)$ is computed. If the method were perfect, no correction would be necessary. However, since contacts are linearized, the predicted *proxy* might minimally penetrate the surface and it is necessary to resolve this overlap for minimizing the introduced error.

First, several parameters used throughout all sections are defined. Then the generalized mass matrix in the center of mass G is composed of the real mass (\mathbf{M}_B) and the inertia tensors (\mathbf{J}_B) of the *body* with mass m , computed out of the data structures defined in Section 4.2.1:

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_B & \mathbf{0} \\ \mathbf{0} & \mathbf{J}_B \end{bmatrix} = \begin{bmatrix} m\mathbf{I} & \mathbf{0} \\ \mathbf{0} & m\sigma\mathbf{J} \end{bmatrix} \in \mathbb{R}^{6 \times 6}, \quad (4.3)$$

with \mathbf{I} the identity matrix and

$$\sigma = \sqrt[3]{\det\left(\frac{1}{m}\mathbf{J}_B\right)}. \quad (4.4)$$

Preferably, the normalized version of the inertia tensor is used, \mathbf{J} , of which all elements are close to 1. While \mathbf{J} changes the direction of vectors when premultiplied, it is the inertia coefficient σ the factor that mainly changes their length. As the reader will see in the following sections, the mass m drops from the equations. Hence, it has no effect on the *god object* simulation, only σ and \mathbf{J} do.

In the same line, the following normalized directions are defined out of the penalty forces and torques (in object $S(k-1)$ coordinates):

$$\mathbf{u}_x = \mathbf{u}_f = \frac{\mathbf{f}_P}{\|\mathbf{f}_P\|}, \quad \mathbf{u}_t = \frac{\mathbf{t}_P}{\|\mathbf{t}_P\|}, \quad \mathbf{u}_r = \frac{\mathbf{J}^{-1}\mathbf{u}_t}{\|\mathbf{J}^{-1}\mathbf{u}_t\|}. \quad (4.5)$$

Their associated magnitudes are the real effective penetration p and the force-torque lever distance δ :

$$p = p_d - d, \quad \delta = \frac{\|\mathbf{t}_P\|}{\|\mathbf{f}_P\|}. \quad (4.6)$$

At this point, the translation ($\Delta\mathbf{x}_p$) and rotation ($\Delta\mathbf{r}_p$) vectors necessary to solve the penetration p of the *god object* in the previous time stamp are defined:

$$\begin{aligned} \Delta\mathbf{x}_p &= \lambda p \mathbf{u}_x, \\ \Delta\mathbf{r}_p &= \theta(\lambda, p) \mathbf{u}_r. \end{aligned} \quad (4.7)$$

In this last equation (4.7), there are two important (still) unknown parameters associated to the current penalty values: θ is the correction rotation step, whereas $\lambda \in [0, 1]$ is the translation-rotation distribution factor. If $\lambda = 1$, then $\theta = 0$, hence, the frame $S(k-1)$ is only translated a distance p along the \mathbf{u}_x direction to obtain the corrected $S'(k-1)$. On the other hand, if $\lambda = 0$, then $\theta = \theta_{\max}$, hence, the frame $S(k-1)$ is only rotated θ_{\max} units around \mathbf{u}_r to obtain the corrected $S'(k-1)$. Usually, the real values lie somewhere in between. In the next two sections, analytical closed form formulae for $\theta(\lambda, p)$ and λ are provided.

4.2.2.1 Computation of the Correction Rotation (θ)

In this section, the case $\theta = \theta_{\max}$ will be considered, i. e., the penetration p is fully transformed into a rotation. It is assumed that the equivalent system presented here can fully solve the penetration by rotating. The correct optimum expression for $\theta(\lambda, p) \in [0, \theta_{\max}]$ that automatically regulates that assumption is provided at the end of the subsection.

A first approximation of the rotation required for fully solving the penetration could be

$$\theta_{\max} \simeq \frac{p}{\delta} \quad \Rightarrow \quad \theta \simeq \frac{p}{\delta}(1 - \lambda). \quad (4.8)$$

Unfortunately, (4.8) overestimates the necessary rotation in some cases. Therefore, that approximation is improved working on the equivalent system shown in Figure 4.3. This system is built essentially using the eigen ellipsoid of the object and the penalty contact forces and torques from step #1 (Section 4.2.1). This eigen ellipsoid results from the principal axis analysis of the inertia tensor.

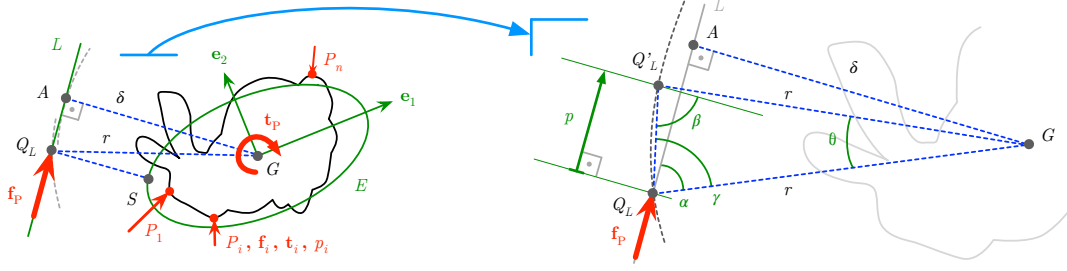


Figure 4.3: Computation of the correction rotation θ . On the left, the equivalent system of a 2D Stanford Bunny is built, consisting of the eigen-ellipsoid $E(\mathbf{J}_B)$ and the force application line $L(\mathbf{f}_P, \mathbf{t}_P)$, on which the equivalent deepest colliding point Q_L with penetration p lies. The distance from Q_L to the center of mass G is the rotation radius r . On the right, a region is zoomed where the point Q_L is rotated around G with its radius r (with exaggerated dimensions for the sake of clarity). The rotation θ brings Q to Q'_L , which is distance p away from Q_L along the force application line L . The rotation $\theta(p, \delta, r)$ should fully solve the penetration p ; its value is deduced in Section 4.2.2.1.

For that purpose, and before starting the simulation, the eigen values s_1, s_2, s_3 and eigen vectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ are computed using the inertia tensor \mathbf{J}_B of the *body* on G . The rotation which brings from the *body* coordinates B to the *eigen* coordinates E is

$${}^B\mathbf{R}_E = [\mathbf{e}_1 \ \mathbf{e}_2 \ \mathbf{e}_3] \in \mathbb{R}^{3 \times 3}. \quad (4.9)$$

On the other hand, the *eigen ellipsoid* E centered in G and expressed in the eigen coordinates $\{x_1, x_2, x_3\}$ is

$$E(\mathbf{J}_B) \equiv \sum_{i=1}^3 \frac{x_i^2}{a_i^2} = \frac{x_1^2}{a_1^2} + \frac{x_2^2}{a_2^2} + \frac{x_3^2}{a_3^2} = 1, \quad (4.10)$$

with the axis lengths a_i computed out of the eigen values s_j

$$a_i^2 = \frac{5}{2} \sum_{j=1}^3 (-1)^\alpha s_j, \quad \alpha = 0 \text{ iff } j \neq i, \ \alpha = 1 \text{ otherwise.} \quad (4.11)$$

Additionally, the force application line L expressed parametrically in the *eigen* coordinates E is

$$\begin{aligned} L(\mathbf{f}_P, \mathbf{t}_P) &\equiv \delta {}^E\mathbf{u}_A + \mu {}^E\mathbf{u}_f = \\ &\quad \underbrace{\delta {}^B\mathbf{R}_E (\mathbf{u}_t \times \mathbf{u}_f)}_{A=G\vec{A}} + \mu {}^B\mathbf{R}_E \mathbf{u}_f = \\ &\quad (\delta_1, \delta_2, \delta_3)^\top + \mu (u_1, u_2, u_3)^\top. \end{aligned} \quad (4.12)$$

The unitary vector ${}^E\mathbf{u}_A$ from (4.12) points from the center of mass G to A , whereas ${}^E\mathbf{u}_f$ is the unitary force direction. The point A is the closest single force application point for the wrench $\{\mathbf{f}_P, \mathbf{t}_P\}$. It is not a *material* point on the *body*, but it is considered to be a material point of the equivalent system, composed by $E(\mathbf{J}_B)$ in (4.10) and $L(\mathbf{f}_P, \mathbf{t}_P)$ in (4.12). Note that in the *eigen* coordinates E , $G = \mathbf{0} = (0, 0, 0)^\top$, however, it is still named for correctness and general validity.

Since E equivalently displays the mass distribution of the body, Q_L is defined to be the closest point on the line L to the ellipsoid E . The distance from Q_L to the center of mass G is the rotation radius r , a key value to define the rotation θ :

$$r = \|\overrightarrow{GQ_L}\|, \quad \text{such that} \quad \min \|L|_{Q_L} - E\|^2. \quad (4.13)$$

As it can be seen in Figure 4.3, $r \geq \delta$. To find Q_L , first, L is substituted in E , which leads to

$$\sum_{i=1}^3 \frac{\delta_i^2 + 2\delta_i u_i \mu + u_i^2 \mu^2}{a_i^2} = 1. \quad (4.14)$$

This expression in (4.14) is a second order equation in μ , with all values δ_i and u_i known. If it has two real roots μ_1, μ_2 , the line L intersects with the ellipsoid E , what leads to two possible force application points $Q_{L,1}$ and $Q_{L,2}$ substituting μ_1 and μ_2 in (4.12), respectively. In case both points are different, Q_L is selected to be the one which satisfies the condition that the force is in opposite direction to the ellipsoid's surface normal:

$$Q_L = Q_{L,i} \text{ s. t. } \nabla E|_{Q_{L,i}} {}^E\mathbf{u}_f \leq 0, \quad i = 1, 2, \quad (4.15)$$

being $\nabla E = (2x_1/a_1^2, 2x_2/a_2^2, 2x_3/a_3^2)^\top$ the gradient of the eigen ellipsoid, thus, its parametrized surface normal.

However, if no real roots exist for (4.14), the line L and the ellipsoid E are disjoint, hence, we are dealing with the case shown in Figure 4.3 (left). Instead of treating the problem as a constrained optimization, it is possible to solve Q_L using projective geometry. Few operations using homogeneous coordinates lead to the point S on E which is closest to L :

$$S = \overrightarrow{GS} = \frac{(a_1^2 \delta_1, a_2^2 \delta_2, a_3^2 \delta_3)^\top}{\sqrt{(a_1^2 \delta_1^2 + a_2^2 \delta_2^2 + a_3^2 \delta_3^2)}}. \quad (4.16)$$

With S known, Q_L can easily be determined:

$$Q_L = \overrightarrow{GQ_L} = \overrightarrow{GA} + ((\overrightarrow{AS})^\top {}^E\mathbf{u}_f) {}^E\mathbf{u}_f. \quad (4.17)$$

The values of the rotation radius r and the force lever δ are enough to estimate the maximum rotation θ the body requires to fully solve a penetration p . The reader is referred to the right part of Figure 4.3, where Q_L is rotated an angle θ around G with a radius r , as if it was a material point. The resulting rotated position Q'_L is a distance p away from Q_L along the direction of the forces ${}^E\mathbf{u}_f$. From the figure, one can deduce the following relationships between angles and distances

$$\gamma = \frac{\pi - \theta}{2}, \quad \beta = \frac{\theta}{2} + \alpha, \quad \sin \alpha = \frac{\delta}{r}. \quad (4.18)$$

Since p is expected to be small with respect to the size of the object, so will be θ ; therefore, it can be considered

$$\alpha \gg \theta \quad \Rightarrow \quad \beta \simeq \alpha \quad \Rightarrow \quad \sin \beta \simeq \sin \alpha. \quad (4.19)$$

Additionally, due to the small value of θ , the arc and the segment joining Q_L and Q'_L will be very similar:

$$\overline{Q_L Q'_L} \simeq \widehat{Q_L Q'_L} \quad \Rightarrow \quad \frac{p}{\sin \beta} \simeq r\theta. \quad (4.20)$$

Therefore, using (4.18) and (4.19) in (4.20) yields

$$\theta \simeq \frac{p}{r}(1 - \lambda) \sin \alpha = \frac{\delta}{r^2} p(1 - \lambda). \quad (4.21)$$

Note that if $r = \delta$, the approximation in (4.8) is obtained, in other words, using the first approximation in (4.8) instead of (4.21) increases the possible correction rotation in a factor of r/δ .

4.2.2.2 Computation of the Correction Translation-Rotation Distribution Factor (λ)

Since in the correction step the object is *moved* from $S(k-1)$ to $S'(k-1)$, the translation and rotation of that movement are distributed (λ) by minimizing the required kinetic energy e_c , computed as

$$e_c = \frac{1}{2} m \|\dot{\mathbf{x}}_G\|^2 + \frac{1}{2} \boldsymbol{\omega}^\top \left(\underbrace{(m\sigma\mathbf{J})}_{\mathbf{J}_B} \boldsymbol{\omega} \right). \quad (4.22)$$

The linear ($\dot{\mathbf{x}}_G$) and angular ($\boldsymbol{\omega}$) velocities required for the correction during time step Δt are

$$\begin{aligned}\dot{\mathbf{x}}_G &= \frac{\Delta \mathbf{x}_p}{\Delta t} = \frac{\lambda p}{\Delta t} \mathbf{u}_x, \\ \boldsymbol{\omega} &= \frac{\Delta \mathbf{r}_p}{\Delta t} = \frac{\theta(\lambda, p)}{\Delta t} \mathbf{u}_r.\end{aligned}\tag{4.23}$$

Introducing $\dot{\mathbf{x}}_G$ and $\boldsymbol{\omega}$ from (4.23) into (4.22), it is obtained

$$e_c = \frac{m}{2\Delta t^2} (\lambda^2 p^2 + \sigma\tau\theta^2),\tag{4.24}$$

with

$$\tau = \frac{\mathbf{u}_r^\top \mathbf{u}_t}{\|\mathbf{J}^{-1} \mathbf{u}_t\|} = \frac{(\mathbf{J}^{-1} \mathbf{u}_t)^\top \mathbf{u}_t}{\|\mathbf{J}^{-1} \mathbf{u}_t\|^2}.\tag{4.25}$$

For a minimum value of kinetic energy e_c on λ , the equation (4.24) must satisfy

$$\frac{\partial e_c}{\partial \lambda} = 0 \quad \Rightarrow \quad \lambda p^2 + \sigma\tau\theta \frac{\partial \theta}{\partial \lambda} = 0.\tag{4.26}$$

If the definition of θ from (4.21) is introduced into (4.26) and solved for λ , the analytical value of the translation-rotation distribution is obtained:

$$\lambda = \frac{1}{1 + \frac{r^4}{\sigma\tau\delta^2}} \in [0, 1].\tag{4.27}$$

4.2.2.3 Assembly of the Final Correction Step

At this point, the correction step transformation matrix $\Delta \mathbf{H}^{\text{pen}}$ which transforms from $S(k-1)$ to $S'(k-1)$ can be assembled using the translation and rotation step vectors from (4.7):

$$\Delta \mathbf{H}^{\text{pen}} \leftarrow \begin{cases} \text{setMatrix}(\xi_p \Delta \mathbf{x}_p, \xi_p \Delta \mathbf{r}_p) & \text{if } Q \in \mathbf{A} \\ \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \in \mathbb{R}^{4 \times 4} & \text{otherwise.} \end{cases}\tag{4.28}$$

The gain $\xi_p = 0.2$ helps regulate the speed with which the object is moved to the surface – instead of doing it suddenly, it is performed exponentially along several haptic cycles. Note that if the deepest point $Q \notin \mathbf{A}$ (i. e., $p_d \leq d \Leftrightarrow p \leq 0$), the corrected previous *god frame* must be the same as the uncorrected one: $S'(k-1) = S(k-1)$. However, the correction step vectors $\Delta \mathbf{x}_p$ and $\Delta \mathbf{r}_p$ from (4.7) still have to be computed, since they are used when computing the unconstrained motion in step #3 (next Section 4.2.3). In that case ($Q \notin \mathbf{A}$), since $p < 0$, their meaning does not refer to the movement required to solve penetration, but to the *movement allowed before collision occurs*.

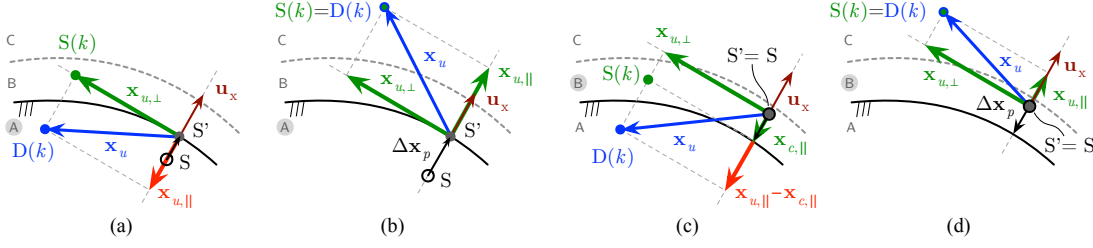


Figure 4.4: Computation of the *constrained* movement \mathbf{x}_c out of the *unconstrained* \mathbf{x}_u using the movement direction constraint \mathbf{u}_x . For simplicity, a 2D version using only translations is shown – the computation of *constrained* rotation is analogous, but using \mathbf{r}_u and \mathbf{u}_r instead. Big dots represent the deepest colliding point Q (with penetration p) of the object in different frames. Blue vectors display the intended *unconstrained* movement from $S'(k-1)$ to $D(k)$. Green vectors are the parallel (\parallel) and orthogonal (\perp) components of \mathbf{x}_u with respect to \mathbf{u}_x which are allowed. Red vectors are the parallel components of \mathbf{x}_u which are not allowed. The constrained motion \mathbf{x}_c is the sum of all allowed components. Subfigures (a) and (b) correspond to the case $Q \in A$, whereas (c) and (d) to the case $Q \in B$, being d the safety distance.

With $\Delta \mathbf{H}^{\text{pen}}$ from (4.28) (in object coordinates $S(k-1)$), the corrected pose of the *god object* in the previous iteration ($k-1$) (in world coordinates W) is

$${}^W \mathbf{H}_{S'}(k-1) = {}^W \mathbf{H}_S(k-1) \Delta \mathbf{H}^{\text{pen}}, \quad (4.29)$$

being ${}^W \mathbf{H}_S(k-1)$ the *god object* pose delivered in the previous iteration ($k-1$).

From here on, the corrected frame $S'(k-1)$ is used instead of $S(k-1)$, and all movement constraint direction vectors are transformed to it:

$$\begin{aligned} \mathbf{u}_x &\leftarrow S' \mathbf{u}_x = \Delta \mathbf{R}^{\text{pen}} \mathbf{u}_x \\ \mathbf{u}_r &\leftarrow S' \mathbf{u}_r = \Delta \mathbf{R}^{\text{pen}} \mathbf{u}_r, \end{aligned} \quad (4.30)$$

being

$$\Delta \mathbf{R}^{\text{pen}} \leftarrow \text{getRotationMatrix}(\Delta \mathbf{H}^{\text{pen}}). \quad (4.31)$$

Although the deduction provided in this subsection might appear relatively long, it is computed in few microseconds, because analytical formulae that provide a correction for any object in any configuration have been derived.

4.2.3 Computation of the Unconstrained Motion (#3)

The *unconstrained* motion of the *god object* is the transformation from the previous corrected *god frame* $S'(k-1)$ to the current *device frame* $D(k)$, as shown in Figure 4.1:

$${}^{S'}\mathbf{H}_D = {}^W\mathbf{H}_{S'}^{-1}{}^W\mathbf{H}_D. \quad (4.32)$$

This *unconstrained* motion is broken down into its translation (\mathbf{x}_u) and rotation (\mathbf{r}_u) parts

$$\begin{aligned} \mathbf{x}_u &\leftarrow \text{getTranslation}({}^{S'}\mathbf{H}_D) \\ \mathbf{r}_u &\leftarrow \text{getRotation}({}^{S'}\mathbf{H}_D) \end{aligned} \quad (4.33)$$

and decompose each of them in parallel (\parallel) and orthogonal (\perp) components with respect to the movement constraint directions (\mathbf{u}_x and \mathbf{u}_r , respectively):

$$\mathbf{x}_u = \mathbf{x}_{u,\parallel} + \mathbf{x}_{u,\perp}; \quad \mathbf{r}_u = \mathbf{r}_{u,\parallel} + \mathbf{r}_{u,\perp}, \quad (4.34)$$

where

$$\begin{aligned} \mathbf{x}_{u,\parallel} &= (\mathbf{x}_u^\top \mathbf{u}_x) \mathbf{u}_x; & \mathbf{x}_{u,\perp} &= \mathbf{x}_u - \mathbf{x}_{u,\parallel}; \\ \mathbf{r}_{u,\parallel} &= (\mathbf{r}_u^\top \mathbf{u}_r) \mathbf{u}_r; & \mathbf{r}_{u,\perp} &= \mathbf{r}_u - \mathbf{r}_{u,\parallel}. \end{aligned} \quad (4.35)$$

4.2.4 Computation of the Constrained Motion (#4)

Figure 4.4 summarizes the computation of the *constrained* motion vectors \mathbf{x}_c and \mathbf{r}_c . Essentially, the parallel components of the *unconstrained* vectors are cancelled or shortened in order to obtain the *constrained* vectors. For the sake of brevity, the procedure with translation vectors (\mathbf{x} , \mathbf{u}_x) is explained only; The computations with rotation vectors are completely analogous, but using rotation constraint direction vector \mathbf{u}_r .

As for the *unconstrained* motion, the *constrained* movement vector is the summation of its parallel and orthogonal components with respect to the motion constraints (\mathbf{u}_x):

$$\mathbf{x}_c = \mathbf{x}_{c,\parallel} + \mathbf{x}_{c,\perp}, \quad (4.36)$$

where *always*

$$\mathbf{x}_{c,\perp} = \mathbf{x}_{u,\perp}. \quad (4.37)$$

On the other hand, the parallel component of the *unconstrained* motion are allowed iff it does not increase penetration in a linearized contact model based on \mathbf{u}_x . If $Q \in \mathbf{A} \Leftrightarrow p \geq 0$ (see Figure 4.4 (a) and (b)), the contact constraint model leads to

$$\mathbf{x}_{c,\parallel} = \begin{cases} \mathbf{x}_{u,\parallel} & \text{if } (\mathbf{x}_{u,\parallel})^\top \mathbf{u}_x \geq 0, \\ \mathbf{0} & \text{otherwise.} \end{cases} \quad (4.38)$$

Otherwise, if $Q \in \mathbf{B} \Leftrightarrow -d < p \leq 0$ (Section 4.4 (c) and (d)):

$$\mathbf{x}_{c,\parallel} = \begin{cases} \mathbf{x}_{u,\parallel} & \text{if } (\mathbf{x}_{u,\parallel})^\top \mathbf{u}_x \geq 0, \\ -\min\{\|\mathbf{x}_{u,\parallel}\|, \|\Delta\mathbf{x}_p\|\} \mathbf{u}_x & \text{else.} \end{cases} \quad (4.39)$$

Recall from Section 4.2.2.3 that the correction vectors $\Delta\mathbf{x}_p, \Delta\mathbf{r}_p$ denote the minimum motion to solve penetration for the case $Q \in \mathbf{A}$ and the minimum motion to reach contact for the case $Q \in \mathbf{B}$.

The difference between the *unconstrained* and *constrained* motion vectors is called *restricted* or *friction* parallel movement:

$$\mathbf{x}_{r,\parallel} = \mathbf{x}_{u,\parallel} - \mathbf{x}_{c,\parallel}. \quad (4.40)$$

This vector, displayed in red in Figure 4.4, is the parallel component which is not allowed and a key value for computing friction in Section 4.2.7.

At this point, the transformation $\Delta\mathbf{H}^{\text{surf}}$ in Figure 4.1 which transforms from $S'(k-1)$ towards $S(k)$ can be assembled:

$$\Delta\mathbf{H}^{\text{surf}} = S'^{(k-1)} \mathbf{H}_{S(k)} \leftarrow \text{setMatrix}(\mathbf{x}_c, \mathbf{r}_c). \quad (4.41)$$

With this step transformation along the surface, the current but still unfiltered *god object or proxy* is computed:

$${}^W \hat{\mathbf{H}}_S(k) = \Delta\mathbf{H}^{\text{surf}} {}^W \mathbf{H}_{S'}(k-1). \quad (4.42)$$

This transformation could be used to define the final *god object* frame, but better results were observed applying some filtering to it, as explained in next Section 4.2.5.

4.2.5 Filtering of the Proxy Pose (#5)

A simple discrete exponential six-DoF low pass filter is applied to ${}^W \hat{\mathbf{H}}_S(k)$ based on the previous *proxy* pose ${}^W \mathbf{H}_S(k-1)$ in order to smoothen the movement of it on the surface. The filter was tested with many complex geometries and several parameter values; with the presented configuration no additional overlaps or artifacts have been observed. For an optimum behavior, the cut-off frequency f_c is linearly evaluated from the difference between ${}^W \hat{\mathbf{H}}_S(k)$ and ${}^W \mathbf{H}_D(k)$:

$${}^W \mathbf{H}_S(k) \leftarrow \text{LowPass}({}^W \mathbf{H}_S(k-1), \underbrace{{}^W \hat{\mathbf{H}}_S(k), {}^W \mathbf{H}_D(k)}_{\Delta \rightarrow f_c \in [1,60] \text{ Hz}}). \quad (4.43)$$

The value taken by f_c depends on the stiffness of the haptic device and the contact configuration, but it is scaled to the range $[1, 60]$ Hz. A typical contact situation with a distance from $S(k)$ to $D(k)$ of about 7 mm can reach a value of $f_c \simeq 20$ Hz.

This filtered *god object* pose ${}^W\mathbf{H}_S(k)$ describes the *proxy frame* $S(k)$. It is used

- (i) to visualize the non-penetrating object,
- (ii) to compute the coupling constraint force as explained in the next Section 4.2.6,
- (iii) and to obtain the penalty contact manifold in the next cycle ($k + 1$), as explained in Section 4.2.1.

4.2.6 Coupling Forces Applied to the Haptic Device (#6, #7, #8)

Finally, the constraint coupling forces are computed in the step # 6 (see Figure 4.2). For that, the difference between the haptic *device* pose and the virtual *god-object* pose is computed first:

$${}^D\mathbf{H}_S(k) = {}^W\mathbf{H}_D^{-1}(k) \cdot {}^W\mathbf{H}_S(k). \quad (4.44)$$

Then, after obtaining the vector representation of (4.44),

$$\begin{aligned} \mathbf{x}_{VC} &\leftarrow \text{getTranslation}({}^D\mathbf{H}_S(k)), \\ \mathbf{r}_{VC} &\leftarrow \text{getRotation}({}^D\mathbf{H}_S(k)), \end{aligned} \quad (4.45)$$

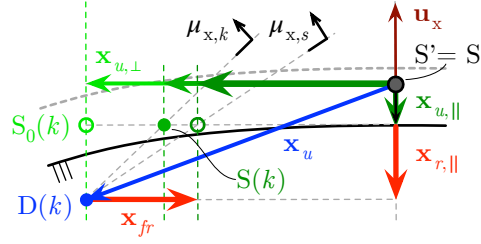
virtual coupling stiffness constants are applied to it:

$$\mathbf{f}_{VC} = k_{VC,x} \mathbf{x}_{VC}; \quad \mathbf{t}_{VC} = k_{VC,r} \mathbf{r}_{VC}. \quad (4.46)$$

The stiffness constants were experimentally set to be $k_{VC,x} = 1$ and $k_{VC,r} = 0.025$ (unit-less factors).

As a last step, the maximum virtual stiffness (k_D) and the corresponding damping (b_D) of the haptic device are applied to $\mathbf{f}_{VC}, \mathbf{t}_{VC}$ before displaying them to the user as $\mathbf{f}_D, \mathbf{t}_D$. In the experiments (see Section 4.4), the HUG was used, consisting of two DLR/KUKA Light Weight Robot transformed into a haptic device [HSA⁺08], as already explained and further described in Appendix A. The usual constant values for the haptic device are $k_D = 4000$ N/m and $b_D = 20$ Ns/m. However, in the simulations, moderate ($k_D = 2000$ N/m) and low ($k_D = 200$ N/m) stiffness values are used. Lower constants allow deeper penetrations of the device, which is more challenging, and it also becomes visually more noticeable how the *god object* remains on the surface.

Figure 4.5: Friction model which operates in the pose configuration space of the object (case $Q \in A$ displayed, (c) from Figure 4.4). The apex of the friction cone is placed in $D(k)$ and its axis is the opposite of the restricted parallel movement defined in (4.40): $-\mathbf{x}_{r,\parallel}$. The angle of the cone is defined with the static ($\mu_{x,s}$) and/or kinetic ($\mu_{x,k}$) friction coefficients. The current *proxy frame* $S(k)$ is moved to the boundary of the cone using \mathbf{x}_{fr} . Static, kinetic and viscous friction are possible for both translations and rotations. In this figure, the final pose of $S(k)$ due to kinetic friction is displayed with a filled green dot.



4.2.7 Six-DoF Friction (#4)

In order to compute friction forces, it is operated in the object configuration space, restricting the *constrained* movement of the *proxy* \mathbf{x}_c with a *friction restriction* movement \mathbf{x}_{fr} :

$$\mathbf{x}_c = \mathbf{x}_{c,\parallel} + \mathbf{x}_{c,\perp} + \mathbf{x}_{fr}. \quad (4.47)$$

This *friction restriction* movement is computed using the allowed perpendicular motion $\mathbf{x}_{u,\perp}$ and the forbidden parallel motion $\mathbf{x}_{r,\parallel}$ introduced in (4.40):

$$\mathbf{x}_{fr} \leftarrow \text{computeFriction}(\mathbf{x}_{r,\parallel}, \mathbf{x}_{u,\perp}). \quad (4.48)$$

Algorithm 4.1 summarizes the computation of \mathbf{x}_{fr} and Figure 4.5 illustrates it for translations. The presented method is similar to approaches presented by [ST97], [HM02], [KOKM11], all introduced in Section 4.1.1. The basic idea consists in, first, computing friction cones with apex in $D(k)$ and axis parallel to \mathbf{u}_x , and then, sliding $S(k)$ to the cone boundaries, achieved by adding \mathbf{x}_{fr} to the *constrained* movement \mathbf{x}_c . As shown in Algorithm 4.1, there can be friction only if there is forbidden parallel motion ($\|\mathbf{x}_{r,\parallel}\| > 0$), i. e., the user is applying a force/movement in opposite direction to the surface. In that case, the friction vector \mathbf{x}_{fr} will always point in the opposite direction to $\mathbf{x}_{u,\perp}$ and its length will depend on the type of movement and friction associated to it:

- (i) Static friction – If the perpendicular movement is fully contained in the static friction cone, the object will not move perpendicularly, i. e. $\mathbf{x}_{fr} = -\mathbf{x}_{u,\perp}$. The aperture of the static cone is defined with the static friction coefficient $\mu_{x,s}$.
- (ii) Kinetic friction – If, on the contrary, the perpendicular movement is outside of the friction cone, a new (smaller) kinetic cone is computed and the length of \mathbf{x}_{fr} is set to reach the boundary of the cone. Hence, the object will move, but less than

in the frictionless case and tangential forces proportional to the length of \mathbf{x}_{fr} are going to be present. The aperture of the kinetic cone is defined with the kinetic friction coefficient $\mu_{x,k}$.

- (iii) Viscous friction – In the case kinetic friction occurs, the length of \mathbf{x}_{fr} is additionally increased proportionally to the perpendicular velocity ($\mathbf{x}_{u,\perp}$) using the factor $\mu_{x,v}$.

Note that the presented friction model works also with rotations with the same algorithm but using the rotation vectors ($\mathbf{r}_c, \mathbf{r}_{r,\parallel}, \mathbf{r}_{u,\perp}, \mathbf{x}_{fr}$) and friction coefficients ($\mu_{r,s}, \mu_{r,k}, \mu_{r,v}$) instead. In the case of rotations, the computations are more difficult to illustrate, but, when applying friction, essentially, the amount of rotation from $S'(k-1)$ to $S(k)$ is decreased without altering the axis of rotation.

Friction coefficients can be measured or looked up in tables. In the experiments, the most stable behaviors were achieved by choosing first the static friction coefficients $\mu_{x,s}$ and $\mu_{r,s}$ (≈ 0.2) and then fixing $\mu_k \approx 0.9\mu_s$ and $\mu_v \approx 0.01\mu_s$.

Algorithm 4.1: $\mathbf{x}_{fr} = \text{computeFriction}(\mathbf{x}_{r,\parallel}, \mathbf{x}_{u,\perp})$

Data: Allowed perpendicular motion $\mathbf{x}_{u,\perp}$ and not allowed parallel motion, $\mathbf{x}_{r,\parallel}$.

Result: Motion restriction due to friction, \mathbf{x}_{fr} .

```

// Initialize default friction restriction
1  $\mathbf{x}_{fr} = \mathbf{0}$ 
2 if  $\|\mathbf{x}_{r,\parallel}\| > 0$  then
3   if  $\|\mathbf{x}_{u,\perp}\| < \mu_{x,s}\|\mathbf{x}_{r,\parallel}\|$  then
4     // Static friction
5      $\mathbf{x}_{fr} = -\mathbf{x}_{u,\perp}$ 
6   else
7     // Kinetic friction
8      $\mathbf{x}_{fr} = -\mu_{x,k}\|\mathbf{x}_{r,\parallel}\| \frac{\mathbf{x}_{u,\perp}}{\|\mathbf{x}_{u,\perp}\|}$ 
9     // Viscous friction
10     $\mathbf{x}_{fr} \leftarrow \mathbf{x}_{fr} - \mu_{x,v}\mathbf{x}_{u,\perp}$ 
11 return  $\mathbf{x}_{fr}$ 

```

4.3 Theoretical Discussion of Methods

In this section, *god object* approaches based on the Gauss' least constraint principle [Gau29] are discussed. The work presented by Ortega et al. [ORC07] is taken as a reference, since it is considered to be the main contribution on which many others are based. Additionally, this method is compared with the heuristics presented in the previous Section 4.2. It is worth to mention that the solution by Ortega et al. has not been implemented

within this work, hence, the matter is studied theoretically and rather analogies between methods are pointed out, for finally giving some insights on why the heuristics introduced in this work performs as good as shown in Section 4.4, despite of the approximations. Thorough formal, experimental and quantitative comparisons are left for future work.

First, the 6D contact wrench corresponding to one colliding point P_i is defined concatenating its penalty forces and torques from (4.2):

$$\begin{aligned} \mathbf{c}_i(V(P_i)) &= (\mathbf{f}_i^\top, \mathbf{t}_{G,i}^\top)^\top \\ &= (V(P_i) \mathbf{n}_i^\top, (\overrightarrow{GP_i} \times \mathbf{f}_i)^\top)^\top \\ &= V(P_i) (\mathbf{n}_i^\top, (\overrightarrow{GP_i} \times \mathbf{n}_i)^\top)^\top \in \mathbb{R}^{3+3}. \end{aligned} \quad (4.49)$$

Note that the wrench \mathbf{c}_i is attached to the center of inertia G and that, unlike for the approach proposed by Ortega et al., its length is proportional to the penetration ($V(P_i)$) of the colliding point. Section 4.2 works with two separate 3D vectors instead of generalized 6D pose vectors for clarity, and it operates with translations and rotations instead of accelerations to avoid integrating, which can lead to numerical errors. However, to facilitate comparisons, 6D generalized acceleration vectors will be employed in this section (always on G), defined as $\mathbf{a} = (\ddot{\mathbf{x}}^\top, \ddot{\mathbf{r}}^\top)^\top \in \mathbb{R}^{3+3}$.

The acceleration is called *unconstrained*, \mathbf{a}_u , when no contact wrenches \mathbf{c}_i are considered. On the other hand, it will be *constrained*, \mathbf{a}_c , if collisions are considered. In other words, the *unconstrained* acceleration describes the movement of a free body, whereas the *constrained* acceleration is related to the movement of the same body but under certain contacts. Making the assumption of *quasi-statics* (i. e., velocity is zero at the beginning of every cycle), Ortega et al. define the *unconstrained* acceleration to be proportional to the movement step from the previous *god object* pose (S) to the current *device* pose (D):

$$\mathbf{a}_u = k_S \left((\mathbf{x}_D^\top, \mathbf{r}_D^\top) - (\mathbf{x}_S^\top, \mathbf{r}_S^\top) \right)^\top, \quad (4.50)$$

where the gain k_S is empirically chosen ($k_S = 0.5$ for Ortega et al.).

One of the main goals of the *god object* method consists in determining the *constrained* acceleration related to this *unconstrained* acceleration; integrating it, the pose of the *god object* is obtained. For that, Gauss' least constraint principle is applied. This fundamental principle states that the *constrained* acceleration state of a body minimizes a *kinetic distance* function $g(\mathbf{a}) \in \mathbb{R}$ subject to all contact constraints $h_i(\mathbf{a}) \in \mathbb{R}, \forall i = 1, \dots, N$:

$$\begin{aligned} \min g(\mathbf{a}) &= \frac{1}{2} (\mathbf{a} - \mathbf{a}_u)^\top \mathbf{M} (\mathbf{a} - \mathbf{a}_u) \\ \text{s.t. } h_i(\mathbf{a}) &= \mathbf{c}_i^\top \mathbf{a} \geq 0. \end{aligned} \quad (4.51)$$

The mass and inertia matrix $\mathbf{M}_B \in \mathbb{R}^{6 \times 6}$ has already been introduced in (4.3). The constraint functions $h_i(\mathbf{a})$ represent unilateral contacts that do not allow the object to

accelerate in the opposite direction of the contact forces and torques. Obviously, all penalty forces that satisfy this condition have no effect in the computation of the *god object* pose.

Ortega et al. use Wilhemsen's nearest point algorithm [Wil76] to solve this quadratic programming problem in (4.51). Instead, the algebraic solution is provided here for discussion. First, the variable change $\mathbf{z} = \mathbf{a} - \mathbf{a}_u$ is applied to (4.51), which yields

$$\begin{aligned} \min g(\mathbf{z}) &= \frac{1}{2} \mathbf{z}^\top \mathbf{M} \mathbf{z} \\ \text{s.t. } h_i(\mathbf{z}) &= \mathbf{c}_i^\top \mathbf{z} + \mathbf{c}_i^\top \mathbf{a}_u = \mathbf{c}_i^\top \mathbf{z} + b_i \geq 0, \end{aligned} \quad (4.52)$$

then, Lagrange multipliers are introduced and the Karush-Kuhn-Tucker conditions are used to obtain the following linear system with $6 + N$ scalar unknowns (\mathbf{z} and h_i) and equations:

$$\begin{cases} \nabla g(\mathbf{z}) + \sum_{i=1}^N \lambda_i \nabla h_i(\mathbf{z}) = 0 \text{ (6 eqs.)} \\ \lambda_i h_i(\mathbf{z}) = 0 \text{ (} N \text{ eqs.).} \end{cases} \quad (4.53)$$

After some operations in (4.53), the optimum \mathbf{z} which leads to the *constrained* acceleration can be obtained:

$$\mathbf{z}_{\min} = \mathbf{a}_c - \mathbf{a}_u = -\mathbf{M}^{-1} \sum_{i=1}^N \lambda_i \mathbf{c}_i, \quad (4.54)$$

with

$$\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_i, \dots, \lambda_N)^\top = \mathbf{F}^{-1} \mathbf{b}, \quad (4.55)$$

and being

$$\begin{aligned} \mathbf{b} &= (\mathbf{c}_1^\top \mathbf{a}_u, \dots, \mathbf{c}_i^\top \mathbf{a}_u, \dots, \mathbf{c}_N^\top \mathbf{a}_u)^\top \in \mathbb{R}^N, \\ \mathbf{F} &= [\mathbf{c}_i^\top \mathbf{M} \mathbf{c}_j] \in \mathbb{R}^{N \times N}, \quad \forall i, j = 1, \dots, N. \end{aligned} \quad (4.56)$$

Equations (4.54) and (4.55), and parameters in (4.56) give the solution of the *constrained* acceleration \mathbf{a}_c for a rigid body with mass matrix \mathbf{M} and a set of penalty contacts $\{\mathbf{c}_i\}$. The pose of the *god object* can be obtained integrating this \mathbf{a}_c in two steps. The contact matrix \mathbf{F} is symmetric and assumed to be invertible in general. The values $b_i = \mathbf{c}_i^\top \mathbf{a}_u$ must be negative so that their corresponding Lagrange multipliers λ_i are positive; if not, $\lambda_i = 0$, i. e., the direction of contact wrench \mathbf{c}_i would be conform to the *unconstrained* acceleration, thus, it would have no constraining effect.

As stated by Ortega et al., "the constrained acceleration is the non-euclidean projection of the unconstrained on the set of all possible accelerations". The value of \mathbf{z} is

related to the difference between the accelerations caused by this projection and it is interestingly the weighted sum of all contact wrenches, being the Lagrange multipliers the weighting factors. In fact, in the heuristics presented in this work, the component which is parallel to the contact wrench is also removed from the *unconstrained* displacement vector if it is pointing in the opposite direction (see Section 4.2.4).

However, there are two fundamental differences in the presented approach with respect to the method by Ortega et al.:

- (i) In their work, the authors perform continuous collision detection along the *unconstrained* movement path, and the set of contact wrenches corresponds to a surface manifold in which all points have the same relevance; in contrast, the contact set in the presented heuristics corresponds to the error of the previous *god object* pose, and *each contact is weighted with its penetration*, with deeper contact points being more relevant.
- (ii) Their contact set is computed asynchronously in a separate thread and its update rate is around one order of magnitude slower than the force display frequency (1 kHz); on the other hand, in this work, the *god object* simulation and force display are carried out sequentially and below the 1 ms computation time (see Section 4.4), which helps to distribute any error caused by approximations or the method itself along more computation cycles. In other words, *the god pose computation and error compensation is virtually distributed onto several cycles that occur faster*.

Therefore, since all contact wrenches are already weighted with their corresponding penetrations, one could assume that all effective Lagrange multipliers are similar ($\tilde{\lambda} \simeq \lambda_i \forall i$), which would simplify (4.54) considerably. This can be realized in practice by considering the sum of all (weighted) contact wrenches $\tilde{\mathbf{c}} = \sum \mathbf{c}_i(V(P_i)) = (\mathbf{f}_P^\top, \mathbf{t}_P^\top)^\top$ to be the only contact wrench, which leads to an approximation of the *constrained* acceleration $\tilde{\mathbf{a}}_c \simeq \mathbf{a}_c$ in (4.54):

$$\tilde{\mathbf{a}}_c = \begin{cases} \mathbf{a}_u - \frac{\tilde{\mathbf{c}}^\top \mathbf{a}_u}{\underbrace{\tilde{\mathbf{c}}^\top \mathbf{M}^{-1} \tilde{\mathbf{c}}}_{\tilde{\lambda}}} \mathbf{M}^{-1} \tilde{\mathbf{c}} & \text{if } \tilde{\mathbf{c}}^\top \mathbf{a}_u < 0, \tilde{\mathbf{c}}^\top \mathbf{M}^{-1} \tilde{\mathbf{c}} \neq 0 \\ \mathbf{a}_u & \text{else.} \end{cases} \quad (4.57)$$

The formula for the *constrained* acceleration in (4.57) would be the equivalent to the corrections performed to the *unconstrained* displacement vector \mathbf{x}_u in the presented heuristics (see Section 4.2.4). If the assumption made (i. e., the Lagrange multipliers can be considered similar in case contact wrenches are weighted with penetration) is not good enough, the approximative contact wrench $\tilde{\mathbf{c}}$ does not solve the overlap of the *god*

object on the direction related to the biggest and most significant multipliers properly; as a result, the overall approximate contact wrench becomes bigger in the directions of the unsolved contacts in the next cycle, and hence, it is resolved with higher priority. In other words, the algorithm regulates in several cycles the errors introduced due to simplifications, given its fast computational velocity. No case could be found where the heuristics fail to work.

Overall, the heuristics presented in this work are considered to be much easier to understand and implement, since it is based on a very intuitive idea: restriction of movements to the contact surface, using for that the desired *unconstrained* movement step and the penalty forces as averaged surface normal. Additionally, the quadratic programming approach is avoided; quadratic programming is essentially a more general problem that increases implementation and computational complexity. Besides that, it is directly operated on the configuration space of the object, computing translations and rotations, which spares integrating accelerations. Therefore, in its simplicity, the presented heuristic method is very fast and robust and it is computed right after the contact computation step avoiding asynchronous parallel loops which could lead to sudden force artifacts.

4.4 Experiments and Results

Several benchmarking experiments were performed, three of them shown in Figure 4.6 and in the video accessible under

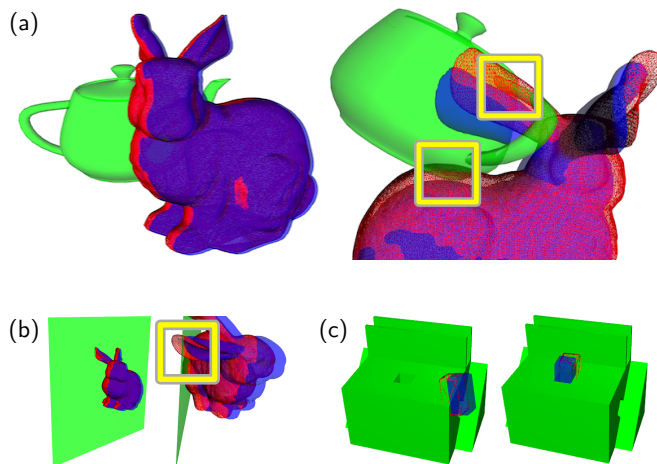
Fast and Robust Six-DoF God Object Heuristic for Haptic Rendering of Complex Models with Friction @ Vimeo

<https://vimeo.com/190217647>

As mentioned in Section 4.2.6, the HUG haptic device was used (see Section A) with moderate ($k_D = 2000$ N/m) and low ($k_D = 200$ N/m) stiffness values. The first scenario in Figure 4.6(a) consists in interacting with the Stanford Bunny and the Utah Teapot (both desktop sized). Second, the same bunny hits a thin surface to check that it does not pop through (b). And finally, a peg object is introduced into a hole, in such a way that roughly 90% of all its points collide (c). Figure 4.7 shows the results of the first bunny-teapot scenario and Figure 4.8 of the second bunny-plane environment. Contact and computation time values are plotted for hitting, sliding and scenario specific tasks. The third scenario is portrayed in the aforementioned video and extensively analyzed in Chapter 6. Further realistic assembly simulations are also featured in the following Chapter 5.

In all tested scenarios, stable, stiff and realistic forces were always generated, and the *god object* proxy remained visually on the surface. The penetration (p_d , Section 4.2.2)

Figure 4.6: Three benchmarking scenarios: (a) the Stanford Bunny collides against The Utah Teapot and the bunny’s ear is introduced into the handle of the teapot, (b) The Stanford Bunny collides against a thin plane, (c) A classical peg-in-hole benchmark. Green objects are the ones voxelized. The red mesh is the representation of the (unconstrained) object moved by the user, whereas the blue one is the god object constrained to the surface of the green object. Yellow boxes highlight challenging areas where the computation of the constrained object is successfully achieved.



stayed below the used voxel size, which defines the maximum resolution. Moreover, in the case of the bunny-plane benchmark (Figure 4.8), that error was below the voxel half size threshold most of the time. Additionally, the total computation time (steps #1 – #7) stayed always easily below the 1 ms convention even when several thousands of colliding points were reached. Besides that, the results in Figure 4.7 and Figure 4.8 show that the presented *god object* method requires only around $5 \mu\text{s}$ for collision cases (see computer specifications in the captions of the figures).

The applied forces in the plots (\mathbf{f}_D , \mathbf{t}_D) are the values sent to the haptic device after multiplying the stiffness and damping constants to the constraint or virtual coupling forces and torques (\mathbf{f}_{VC} , \mathbf{t}_{VC}), as explained in Section 4.2.6. Thus, curve shapes are similar. On the other hand, the penalty forces and torques (\mathbf{f}_P , \mathbf{t}_P) correspond to the VPS forces (step #1, Chapter 3) obtained with the *god object* proxy constrained but slightly penetrating the surface. As small and differently shaped as they seem, they are the very first value necessary for computing the other plotted force values. Since the god object algorithm works in the object configuration space, friction is represented as the friction restriction movement \mathbf{x}_{fr} shown in Figure 4.5 and computed in Algorithm 4.1.

In both scenarios, the applied force values increase when objects are pushed against each other harder (segments A–B). The insertion of the bunny ear into the teapot handle (from instant C on) in Figure 4.7 is a notable *non-convex* task. As expected, the force-torque ratio decreases in favor of bigger torques than usual, since the user is applying a lever force once the ear has been introduced through the handle. A similar behavior can be observed between the translational and rotational friction components.

It is worth to mention that the two experiments in Figure 4.7 and Figure 4.8 were

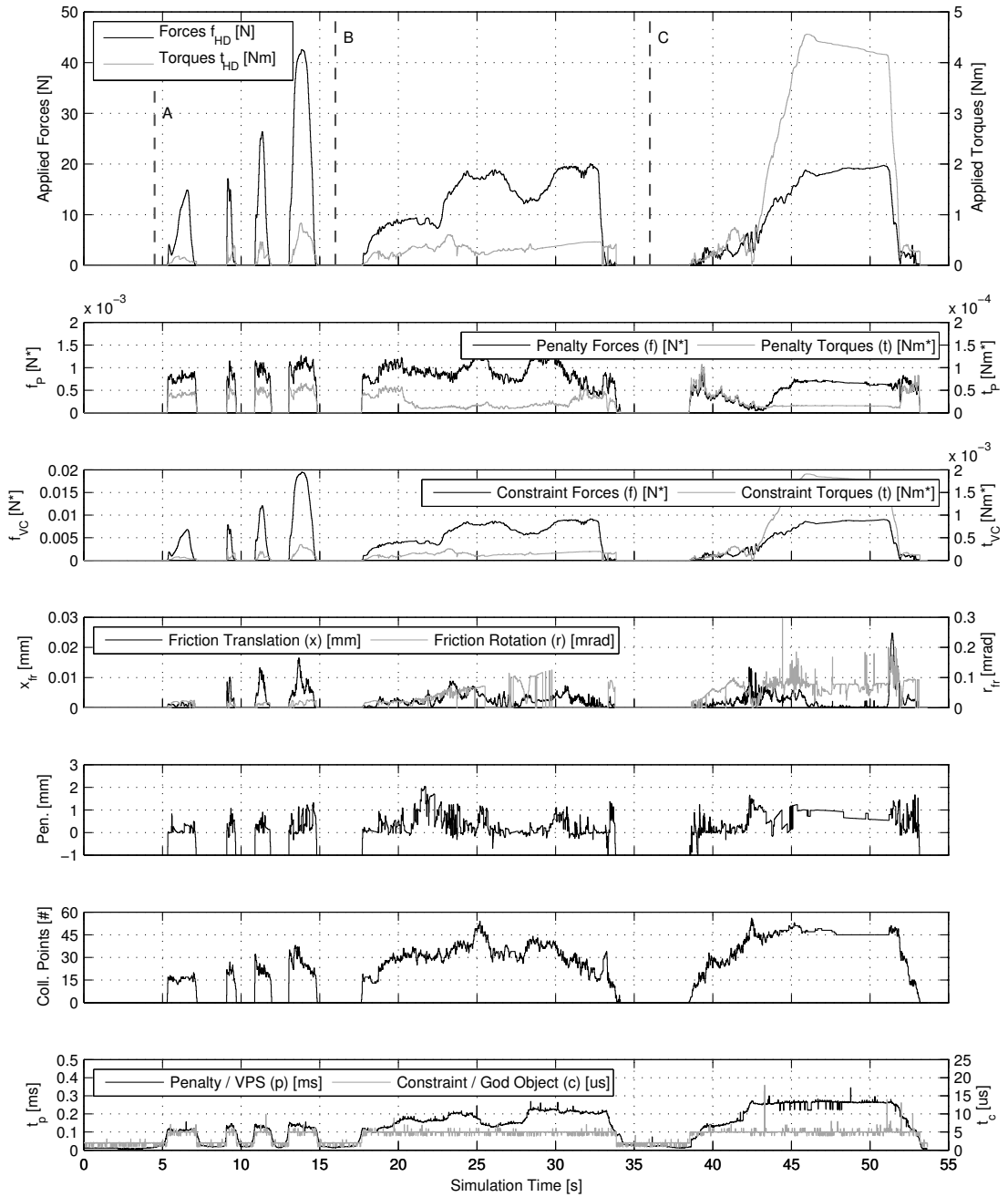


Figure 4.7: Experimental results of the presented method for the scenario displayed in Figure 4.6(a), where the Stanford Bunny (point-cloud, 5587 points in 7 levels) collides against the Utah Teapot (voxelized distance field, $360 \times 299 \times 315$ voxels with 2 mm voxel edge length). Computed force and torque values (applied, penalty, and constraint), friction movement corrections ($\mu_s = 0.2$, $\mu_k = 0.9\mu_s$, $\mu_v = 0.01\mu_s$), penetration, colliding points, and computation time are shown. The device constants were $k_D = 2000 \text{ N/m}$ and $b_D = 17 \text{ Ns/m}$. Units marked with a * (penalty and constraint forces) are not the real units, since their respective values are multiplied by additional gains to obtain them. The computation time is divided into the one related to the penalty-based method (step #1) and the one of the god object method (steps #2 – #7). From start time to marker A objects are separated; Between A – B the bunny knocks the teapot with increasing forces; Between B – C the bunny slides around the teapot; From C until the end of the simulation one ear of the bunny is introduced into the handle of the teapot. The used computer was an Intel(R) Core(TM) 2 Quad with CPUs at 2.66GHz and running Suse SLED 11 32B (not realtime).

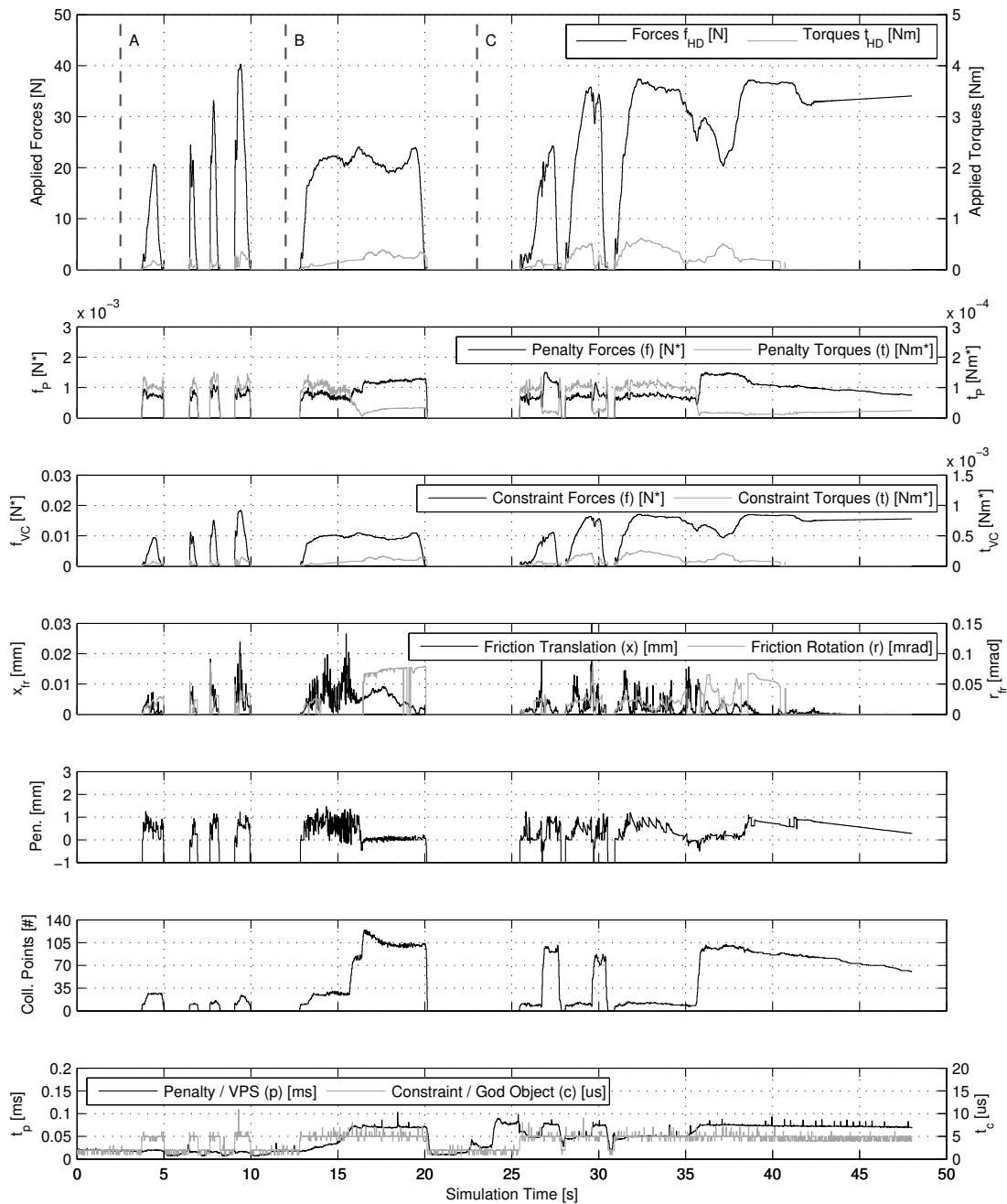


Figure 4.8: Experimental results of the presented method for the scenario displayed in Figure 4.6(b), where the Stanford Bunny (point-cloud, 5587 points in 7 levels) collides against a thin plane (voxelmap, $250 \times 250 \times 1$ solid voxels with 2 mm voxel edge length). Computed force and torque values (applied, penalty, and constraint), friction movement corrections ($\mu_s = 0.2$, $\mu_k = 0.9\mu_s$, $\mu_v = 0.01\mu_s$), penetration, colliding points, and computation time are shown. The device constants were $k_D = 2000 \text{ N/m}$ and $b_D = 17 \text{ Ns/m}$. Units marked with a * (penalty and constraint forces) are not the real units, since their respective values are multiplied by additional gains to obtain them. The computation time is divided into the one related to the penalty-based method (step #1) and the one of the god object method (steps #2 – #7). From start time to marker A objects are separated; Between A – B the bunny knocks the plane with increasing forces; Between B – C the bunny slides on the thin plane; From C until the end of the simulation the bunny is placed on the edge of the plane and then pushed against it. The used computer was an Intel(R) Core(TM) 2 Quad with CPUs at 2.66GHz and running Open Suse Leap 42.2 64B (not realtime). Note that the operating system is different compared to the previous experiment in Figure 4.7.

carried out with the same computer but different operating systems (and architectures), as mentioned in the captions. This should barely affect the plotted contact magnitudes, but it has an effect on the computation time.

4.5 Summary, Conclusions, and Perspectives

This chapter presented an easy-to-implement *constraint-based* force computation method that can be applied to *penalty-based* haptic rendering algorithms. The method is used to extend the VPS re-implementation introduced in Chapter 3.

Constraint-based or *god object* methods usually simulate and visualize a *god object* or proxy constrained to the contact surface; the users move with the device the penetrating virtual probe (not visualized) and the proxy tries to follow it avoiding penetration. In the presented method, the penalty forces \mathbf{f}_P and torques \mathbf{t}_P of the god object or surface proxy in the previous time step are computed. These six-DoF forces are used to correct with a heuristic the previous surface proxy pose and to define the movement constraint applied to it. The god object tries to follow the movement of the device or probe in the next time step, but only the translations and orientations that are not opposite to the directions defined by \mathbf{f}_P and \mathbf{t}_P are allowed – this yields the next surface proxy pose. The constraint force is the virtual coupling force resulting from the difference between the pose of the device and the god object pose constrained to the surface.

As shown in the results, the method simulates the surface proxy and computes six-DoF collision forces with the advantages of both paradigms (penalty- and constraint-based) even with complex objects (e.g., non-convex and non-watertight): computational speed (only around $5\ \mu\text{s}$) and accuracy (error bounded by the resolution of data structures). Additionally, the algorithm computes six-DoF static and dynamic frictional forces.

A theoretical comparison of the presented method to the well-known god object method from Ortega et al. [ORC07] is also discussed. Future work will cover the experimental comparison to that method and the extension of the presented algorithm to multi-body scenarios.

The *constraint-based* haptic rendering algorithm presented in the current chapter is integrated into a virtual assembly simulation platform, as explained in Chapter 5, and evaluated in Chapter 6 in comparison to the *penalty-based* algorithm from Chapter 3.

Chapter 5

Integration, Applications, and Interaction Techniques

This chapter builds upon the haptic rendering algorithms introduced in Chapter 3 and Chapter 4, and presents the definition and implementation of the following application frameworks:

- (i) Section 5.2: A platform for bi-manual virtual assembly simulations with haptic feedback in large environments, supporting multiple complex objects directly imported from CAD tools. Performance results are provided for an exemplary car assembly sequence.
- (ii) Section 5.3: The integration (in form of a plug-in) of the presented collision detection methods into the Physics Engine Bullet [Cou03]. Benchmarking experiments with Bullet's three native algorithms show the outperforming efficiency of the methods introduced in this work, especially when complex objects are handled.

Additionally, Section 5.4 briefly presents five more applications in which the methods from the previous chapters have been integrated successfully, covering, amongst others, robotic planing and manipulation, and maintenance or rehabilitation simulations.

Section 5.3 and Section 5.2 in this chapter use parts from the following peer-reviewed publications written by the author of this work: [SSeS14a], [SSeS14b], and [SHH⁺16]. Besides, additional material from publications (co-) authored by the author of this work or colleagues appears properly cited in different parts from Section 5.4.

5.1 Introduction

For a realistic and immersive interaction in virtual assembly environments with haptic feedback, (i) **collisions** between objects and assembly features must be similar to the ones in the **physical** reality, (ii) the employed user **interfaces** must operate synchronized and according to the capabilities of the human senses, and (iii) natural or at least intuitive **interaction** techniques must be provided. Figure 1.1 from Chapter 1 illustrates these components and their interplay, and Section 2.2 from Chapter 2 thoroughly describes them, considering all major technical and human factors. The reader might revisit those contents for refreshing the picture of the global state-of-the-art; in the following, concrete immersive simulators and methods are briefly reviewed.

5.1.1 Related Work

Related work has been split into two subsections for easier reading, each of them related to one of the main sections in this chapters: Virtual Assembly (VA) Systems (Section 5.1.1.1) and Physics Simulators (Section 5.1.1.2).

5.1.1.1 Virtual Assembly (VA) Systems

Several non-commercial **VA systems with haptic feedback** have been presented in the past years. Seth and Vance [SVO11], and more recently Liu et al. [LYFH15] have surveyed the field, the latter focusing on physically-based interactions. In the following, selected VA platforms and applications that appeared in the last decade are described. Most of them are desktop-based and many use different versions of the Phantom* device for force feedback. Additionally, in many of them, conventional physics engines are used for collision computation, which often force to simplify geometries or keep a moderate complexity in the scene. In the system presented in this work, on the other hand, upper body movements are supported and the collision detection engine allows for large-scale scenarios with several complex objects (non-convex) composed of millions of triangles.

The MIVAS platform [WGP⁺04] is a multimodal VA system with which the users can manipulate complex objects in a CAVE (cave automatic virtual environment) with a virtual hand avatar moved through a CyberGrasp[†] hand exo-skeleton. Collision forces are displayed at the fingers. Manipulations with a virtual hand can lead to very natural interactions, but can also become uncomfortable due to the complexity of the interface. Additionally, the system receives voice inputs.

*<http://www.geomagic.com/en/>

†<http://www.cyberglovesystems.com/cybergasp/>

SHARP [SSV06], on the other hand, is a bimanual virtual assembly simulator that uses two Phantoms and physically-based modeling with the Boeing VPS collision computation algorithm. Their system supports head-mounted displays (HMD) and also a CAVE, and they compute swept volumes for later assembly analysis.

Similarly, the work in [HV07] presents a system where complex parts (which are simplified for collision detection) can be bimanually manipulated with Phantom Omni devices. The authors test the performance of the system while handling some motor parts and in a drop-peg-in-hole scenario when using low- and high-end desktop computers.

Also bimanual, IMA-VR [GRV⁺10] is a multimodal platform that focuses on training assembly skills. The dominant hand controls the haptic device (their own developed LIFhAM or a Phantom) and the motion of the non-dominant hand is captured for gesture commands. Diverse direct and indirect aids are implemented in the simulation to accelerate the learning of the trainee, including tele-mentoring, in which trainers can guide the movement of the apprentice.

In [XLR11], HVAS is described, a virtual assembly system that implements a hierarchical scene graph divided into several layers, from the assembled product itself to polygons. A Phantom device is used and the authors prove in a user study that their geometry constraints and guidance forces improve performance.

HAMS [GBMCL⁺14] is another example of a bimanual haptic assembly and manufacturing system that is able to handle manipulation with two Phantoms. The authors perform collision feedback with a mixed approach in which part collision detection and assembly constraints are displayed. Trajectories are visualized with colored spheres which encode movement properties for later analysis. The system is additionally validated with questionnaires after a user study where assemblies of realistic objects such as a bearing puller or a gear oil pump are performed.

Bimanual and covering upper body movements, VR-OOS [SHH⁺15] is a virtual reality system for satellite on-orbit servicing simulations based on the previous version of the setup presented in this work. Interactions are possible with two DLR/KUKA Light Weight Robots used as haptic devices. The system focuses on collision detection and physical motion simulation of parts in a space environment. The goal of the authors is to research on systems able to test maintenance scenarios in space and eventually generate (virtual) experience data for astronauts or robotic systems.

More recently, VMASS [AAAAD16] was presented, a virtual manufacturing assembly simulation system which focuses on the integration of several interfaces and software modules with the goal of providing the most adequate feedback to trainees. The system consists of a powerwall (but supports also HMDs) and objects are manipulated with a

Phantom Desktop and a 5DT* data glove with vibrotactile feedback.

5.1.1.2 Physics Simulators

Important research has been conducted in the past years regarding multibody dynamics simulation; for instance, some of the big steps and compilations can be found chronologically in [Bar92], [Erl05], and [BETC14]. Since this work focuses on contact rendering, it is beyond its scope analyzing different movement simulation methods and engines; instead, evaluation works that compare off-the-shelf frameworks were considered, such as [SR06] or [BB07].

Along these lines, a relatively recent evaluation [HWS⁺12] showed after performing exhaustive benchmarking tests amongst five publicly available engines that Bullet [Cou03] behaves indeed robustly and ranks always in the best positions – except when restitution scenarios are tested. Due to these good results, and also due to the fact that it is a popular, actively maintained engine with a clear nice-to-use open source code, this engine was selected for the present work.

Bullet has several collision computation modules that detect contact manifolds for objects with complexities beyond basic shapes (i. e., spheres, boxes, or cones). In particular, the VPS re-implementation presented in Chapter 3 is benchmarked in Section 5.3.3 alongside with the following native methods in Bullet:

- The well-known Gilbert-Johnson-Keerthi (GJK) Algorithm [GJK88], which computes distances between convex shapes using their Minkowski sums. This algorithm was also extended to compute penetration values [vdB01].
- The Hierarchical Approximate Convex Decomposition (HACD) presented by Mamou and Ghorbel [MG09], which finds convex patches in an initial non-convex object by hierarchically clustering and decimating the mesh. The result is a convex decomposition to which the GJK algorithm can be applied.
- The GImpact [Leo07] algorithm, a collision detection method that handles arbitrary meshes.

5.1.2 Contributions

As mentioned, most of the VA systems are desktop-based and use physics engines that simplify geometries for collision rendering. Yet, real assembly scenarios can be large and consist of multiple complex objects. In that sense, this chapter brings up a novel

*<http://www.5dt.com>

system to the playground and enhances a commonly used physics engine, tackling those aforementioned shortcomings.

Concretely, Section 5.2 introduces:

- The definition and implementation of a bi-manual and large-scale virtual assembly platform with haptic feedback.
- A multi-body collision detection and force computation framework for complex geometries which builds on the methods explained in Chapter 3 and Chapter 4.
- Intuitive navigation and selection techniques suited for large-scale environments.
- User collaboration methods allowing for several haptic devices to interact with the scenario.
- The extension of collision cues to the arm/elbow with vibrotactile feedback (beyond hands/palms), interesting for large-scale environments.
- Exemplary experiments with an interactive assembly sequence of real car parts (courtesy of Volkswagen AG).

Additionally, Section 5.3 describes:

- The definition and implementation of a plug-in for the physics engine Bullet [Cou03] which seamlessly integrates the collision computation algorithm from Chapter 3. Thanks to the plug-in, developers can use the collision methods presented in this work through the widespread interfaces from Bullet.
- Benchmarking experiments in which three collision computation algorithms integrated in Bullet (GJK, GJK with convex decomposition, and GImpact) are compared to the methods from the plug-in; the approaches from this work outperform the ones available in Bullet, especially for increasingly complex geometries.

As far as the VA platform is concerned, analyzing the effect of using either a desktop-sized or a larger interface for the presented scenario is certainly an important step, which is in part researched in Section 6.2 following a wider focus.

As mentioned, Section 5.4 presents five additional frameworks featuring the methods presented so far. However, these are concisely described, since they are considered to be further from the common theme of the thesis.

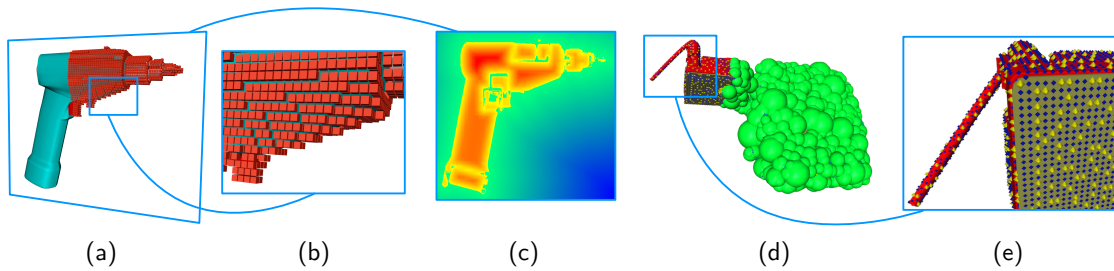


Figure 5.1: Point sampled and voxelized representations of a virtual electronic box and a screw driver. (a) Partially voxelized screw driver ($s = 3$ mm, $83 \times 77 \times 21$ voxels). (b) Close up of the voxelized screw driver. (c) Sagittal section of the voxelized screw driver: distance (turquoise-blue) and penetration (yellow-red) values embedded in the voxelized structure. (d) Point-sphere tree of the electronic control box: one sphere level in green and two successive point levels ($s = 4$ mm, 10507 points, 3506 clusters in $N_L = 7$ levels). (e) Close up of the two last point levels: the blue set contains on average $K = 4$ times more points than the yellow set, which is also contained in the blue set.

5.2 Virtual Assembly with Haptic Feedback

This section presents a virtual reality platform which addresses and integrates some of the currently challenging research topics in the field of virtual assembly: realistic and practical scenarios with several complex geometries, bimanual six-DoF haptic interaction for hands and arms, and intuitive navigation in large workspaces. Special focus is put on the collision computation framework (Section 5.2.1), which is able to display stiff and stable forces in 1 kHz using a combination of penalty- and constraint-based haptic rendering methods from Chapter 3 and Chapter 4, respectively. Interaction with multiple arbitrary geometries is supported in realtime simulations (Section 5.2.2), as well as several interfaces, allowing for collaborative training experiences. Performance results for an exemplary car assembly sequence which show the readiness of the system are provided (Section 5.2.3).

5.2.1 Simulation Framework

This section deals with the different software engines used in order to create the realtime multi-body assembly simulation.

5.2.1.1 Multibody Collision Computation Module

Given the critical rendering frequency of 1 kHz required in haptics, building a multibody environment that dynamically handles several complex objects is not straightforward. In the following, the general structure and the workflow of the library are introduced, as a

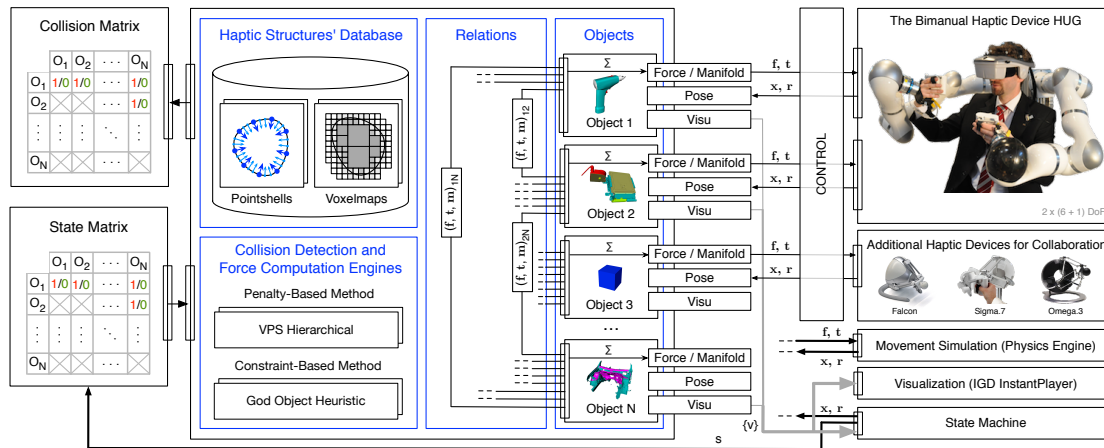


Figure 5.2: Overview of the multi-body simulation framework focusing on collision computation. The collision detection module contains a data base of all object in the scene. Several haptic devices or other modules can be connected to the objects or other appropriate input ports of it; for instance, a physics engine or a game state machine can define the movement of an object. The framework presented in this work uses InstantPlayer* as visualization tool, but other scene graphs could be connected as well. The system is easily scalable. The images of the Omega.3 and the Sigma.7 are courtesy of Force Dimension, Switzerland.

basis for later explanations about the methods used to alleviate the computational effort and deliver contact information for each object in 1 msec.

Collision forces between object pairs are solved using the penalty- and constraint-based methods introduced in Chapter 3 and Chapter 4, respectively. These are implemented as C++ shared libraries, dynamically loaded by the multibody framework. The forces sent to the user are rendered by the constraint-based god object method by default, whereas the re-implementation of VPS is used for detecting collisions between all objects. Figure 5.1 shows some snapshots of the used data structures and objects.

An overview of the multibody library architecture which works beyond the object pair is visualized in Figure 5.2. The scenario is described to the system in a configuration file in which, basically, a list of all objects and their properties is specified, such as name, stiffness, haptic data structure filenames, etc. The parser processes the configuration file and loads all necessary entities into: a database with all haptic structures, object nodes within the framework and the relation links between the objects. Each object node has its own Input/Output (I/O) ports for forces, poses and visualization data. The module receives the poses of the objects and writes their corresponding forces into them. Additionally, there are a collision matrix port and configuration state matrix port. The first summarizes in an object vs object table if an object pair is colliding. The second is a user interface for enabling/disabling collision detection between a specific object pair, also implemented in an object vs object table. Objects can be attached to user interfaces,

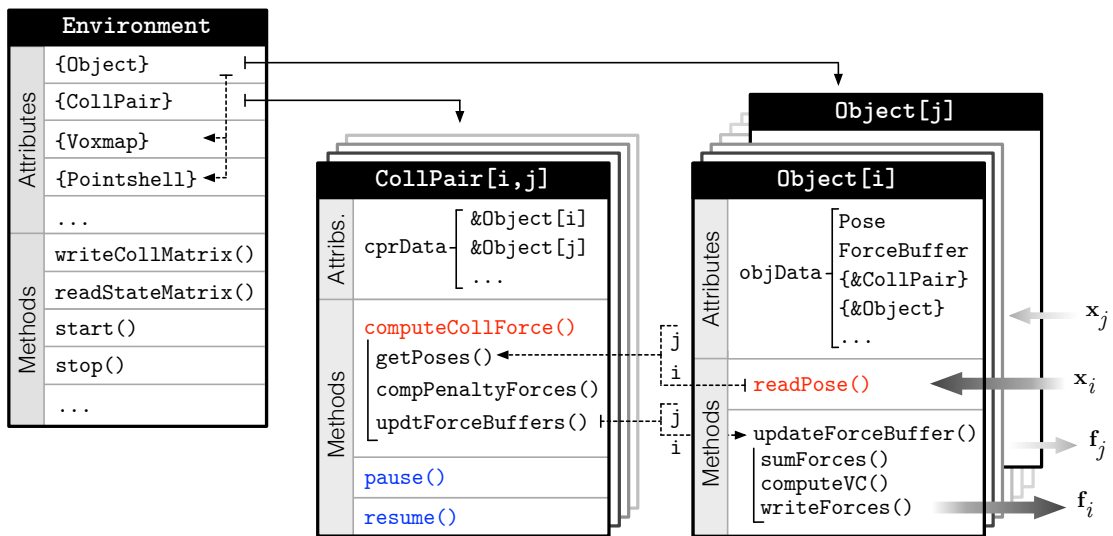


Figure 5.3: Multibody library architecture. The class `Environment` contains and supervises the states of all `Objects` and `CollPairs`. Each `Object` contains both haptic structures that represent it. `CollPairs` call the collision computation method that checks the contacts between the `Objects` they relate.

e. g., to the bimanual haptic device HUG [HHK⁺11] (see Section A.1), or to a physics engine that integrates the collision forces to obtain object poses. The communication between the collision computation module and the devices is realized using thread-safe shared memories and the UDP communication protocol.

In the following, the architecture of the C++ implementation is described; it consists of three main classes, as shown in Figure 5.3:

1. Instantiations `Object[i]` of class `Object` represent a body i in the scenario. The class contains the ports for pose (\mathbf{x}_i) and force (\mathbf{f}_i) data available to the user and methods (e. g., `readPose()`) associated to them. `Objects[i]` contain both voxelized and point-sampled representations and use the appropriate one for each situation (see Figure 5.1).
2. The instance `CollPair[i,j]` of the class `CollPair` uses the pose information (\mathbf{x}_i and \mathbf{x}_j) from its associated `Objects` in order to compute forces according to the algorithms introduced in Chapter 3 and Chapter 4. Each of the instantiations of `CollPair` is related to a cell in the collision matrix.
3. The class `Environment` contains arrays of `Objects` and `CollPairs`, all interconnected. In `Environment`, the collision matrix is written and the configuration state matrix read. All data structures can be accessed by the user via this class, and therefore, the class also has methods to `start()` and `stop()` the functions of

Objects and CollPairs.

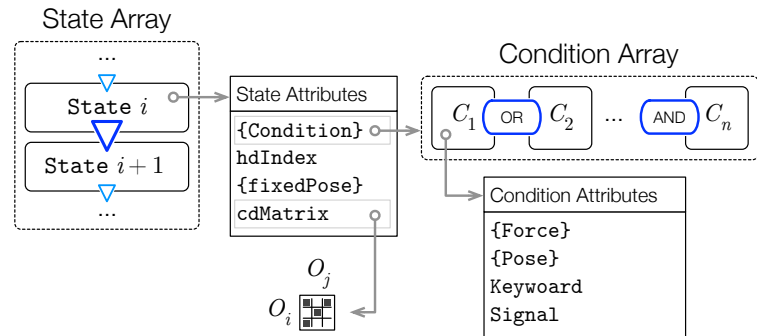
The architecture resembles a graph, where `Objects` are nodes that store poses and forces and the `CollPairs` are links that generate force data in dedicated threads after reading poses. The system is asynchronous and the usual workflow is the following (see Figure 5.3):

- (i) The callback thread function `computeCollForce()` from `CollPair[i,j]` is continuously executed. First, it calls all corresponding `readPose()` in order to update pose data: \mathbf{x}_i from `Object[i]` and \mathbf{x}_j from `Object[j]`.
- (ii) Next, penalty force computation is carried out as explained in Chapter 3, which yields \mathbf{f}_{ij} and its corresponding contact manifold.
- (iii) After that, the thread function passes the penalty force to all related objects via `updateForceBuffer()`. This last method performs several operations on the `ForceBuffer` before sending the total force to the user. The `ForceBuffer` is an array that contains a force cell for each algorithm that is using the `Object`. The total penalty force upon the object is the sum of all forces in the `ForceBuffer`.
- (iv) If the module decides to send the force to the user, first, all penalty forces are summed. Different sending policies are supported: for example, it could be preferred to send total force values every time a new force arrives, or every time the cell with the oldest update time stamp is refreshed. Issues due to lack of synchronization have not been experienced yet. If that were the case, the `ForceBuffer` would have to be extended, for instance, to extrapolate force values with history data every time a force summation needs to be delivered. This analysis is left for future work.
- (v) Next, if the current object is held by the user, the *constraint-based* force rendering explained in Chapter 4 [SH16] is executed with the penalty force summation and the biggest penetration value.
- (vi) Finally, force and visualization data are sent to their corresponding object ports.

In those large-scale environments with multiple objects, one of the main challenges consists in dealing with the quadratic nature of collision detection. Several strategies have been proposed to reduce the computational complexity, such as sweep and prune of pairs using bound boxes [CLMP95]. To cope with multibody environments while keeping the 1 kHz frequency required by haptics, these techniques are exploited in the current work: (i) *object grouping* and *parallelization*, (ii) *spatio-temporal coherence* [MPT06], and (iii) *graceful degradation* [BJ08].

The first one, *object grouping* and *parallelization*, consists in taking advantage of the

Figure 5.4: Game control workflow. Each state dictates which object is moved by the user (`hdIndex`) and specifies the position of the other ones (either free or fixed, `{fixedPose}`). In order to jump to the next state, a series of conditions or tasks related to force or pose values must be fulfilled.



multiple cores of modern CPUs. As previously introduced, `computeCollForce()` (see Figure 5.3) is an interface that can run in a separate thread for each object pair. Higher parallelization degrees are desirable for scenarios with few objects (less than five) and multi-core CPUs, whereas the user should serialize collision computation calls in case the number of geometries increases, due to the thread overload that might be incurred (that degree is currently a parameter in the configuration file). Additionally, objects can be packed in groups in the configuration file so that not all objects from the same group are checked for collision between them. A practical use of that is the kinematic chain formed by the hand and the arm: collisions between the different limbs can be neglected by packing them into the same group.

Secondly, the *spatio-temporal coherence* property of multi-body assembly environments is used before calling `computePenaltyForces()`; the notion behind it is that the distance between two objects is very similar in two consecutive time steps. Since the implemented *penalty-based* approach yields the signed distance p between objects, assuming these are allowed to move no faster than $v_{\max} = 1$ m/s, two objects will not collide during the period p/v_{\max} . This saves in practice hundreds of collision detection calls in a second even in scenarios where objects are relatively close to each other. It is worth to mention that the value of v_{\max} may have to be adjusted for elongated objects. In that case, even small rotations might produce high velocities in the extremities. Nevertheless, the selected value has sufficed in all tested scenarios.

And finally, the third method coping with expensive computations consists in *graceful degradation*, which is possible due to the multi-resolution nature of the point-sphere trees. The volume and surface of all objects is evaluated at the beginning and resource percentages for worst-case scenarios (e. g., when all objects are in contact) are assigned. Additionally, load and efficiency factors are computed online in `computePenaltyForces()`, according to Section 3.3.2. Under worst-case conditions or high loads, pointshells can

automatically limit the number of levels allowed in the hierarchy traverse. Since each level samples the whole object, an approximate but valid result is computed by the algorithm. Force values are scaled with the number of colliding points and point densities, hence, it can be guaranteed that the relative order of magnitude is maintained.

5.2.1.2 Game Control

In order to enable assembly sequences, a finite machine that externally attaches to the multibody collision computation module explained in Section 5.2.1.1 was developed. The user can easily extend the configuration file of the multibody collision detection module specifying which objects have to be mounted in which position and in which order. As shown in Figure 5.4, this is implemented in an array of *states* and, for each *state*, an array of *conditions*.

A state establishes which objects are moved by the user(s) (**hdIndex**). Additionally, objects can have a **fixedPose** (e.g., mounted) or can have no assigned pose, and therefore be waiting for a external pose data. Each state can also update the collision detection state matrix explained in Section 5.2.1.1. This allows for dynamically managing resources; e.g., when an object is mounted, it should not be checked for collision against other mounted pieces, thus, the collision calls between them should be paused.

The condition array is a set of tasks that must be carried out. Usually, each condition is related to target **Pose[j]** and force **Force[j]** values expected on the object *j* that has to be assembled (including tolerances). However, the machine can also receive **Keyboard** commands, such as "fulfilled", or external **Signals** (e.g., from other interfaces, gestures, etc.). Conditions are concatenated with **AND|OR** operators and the user can decide if they need to occur simultaneously or not. When all conditions are fulfilled, an exit event occurs and the machine jumps to the next state.

Simple yet powerful, the game control machine handles the simulation and takes all logic decisions necessary to accomplish assembly sequences.

5.2.1.3 Completing the Jigsaw Puzzle: Communication, Tracking, and Visualization

As shown in Figure 5.2, the **InstantPlayer*** is used as visualization tool. The scene is displayed to the user preferably with an **nVisor SX60†** head-mounted display (HMD), which is optically tracked by a **Vicon Bonita‡** system. Additionally, the **Bullet§** physics

*<http://www.instantreality.org>

†<http://www.nvisinc.com>

‡<http://www.vicon.com/products/camera-systems/bonita>

§<http://bulletphysics.org/wordpress/>

engine has been successfully tested. The framework operates in a distributed manner, having for each module a dedicated desktop computer. The whole system is controlled with an experimental process manager middleware [SBCC17] which is additionally able to communicate in realtime with robotic interfaces.

5.2.2 Interaction Devices and Techniques

This section describes the interaction devices and techniques as key elements of the training platform. The goal is to provide realistic haptic feedback to the hand and the arm while maintaining the safety and increasing the usability (i. e., intuitive experience, minimal restriction of movements with respect to dynamics and workspace).

5.2.2.1 The Bimanual Haptic Device HUG

In order to promote the training effect in assembly simulations and to create an immersive experience for the human operator, a haptic device is required that provides both a large workspace to enable unrestricted movements of the human hand and appropriate force capabilities to generate realistic haptic feedback. For this reason, the bimanual haptic device HUG conceived and built at DLR was chosen in the present application [HHK⁺11].

HUG is equipped with two DLR/KUKA light-weight robot arms* and an additional force-torque sensor at each robot wrist (see Figure 5.2 top right). A full description of the device is given in Appendix A.1, contextualizing it with other commercial devices; the most important features relevant for the uniqueness of the platform are:

- The seven DoF of each robot arm make it possible to interactively optimize their configuration to maximize the overlap between human and robot workspaces, while still providing six-DoF forces – note that commercial devices rarely exceed six DoF.
- Peak forces of 150 N are possible, however, being 50 N the usual maximum collision force for each arm – note this value is considerably higher compared to other devices.

Additionally, HUG supports a variety of hand interfaces that range from passive data gloves to active gripping-force devices, that can be changed depending on the application. In the version presented here, the users operate with the bare hand, but new hand interfaces are contemplated as future work.

5.2.2.2 The Vibrotactile Arm Band VibroTac

With the haptic feedback applied by HUG on the human hand, the user is able to perceive the collision of a manipulated object in a virtual scene. However, in certain situations, it

*<http://www.kuka-lbr-iiwa.com>

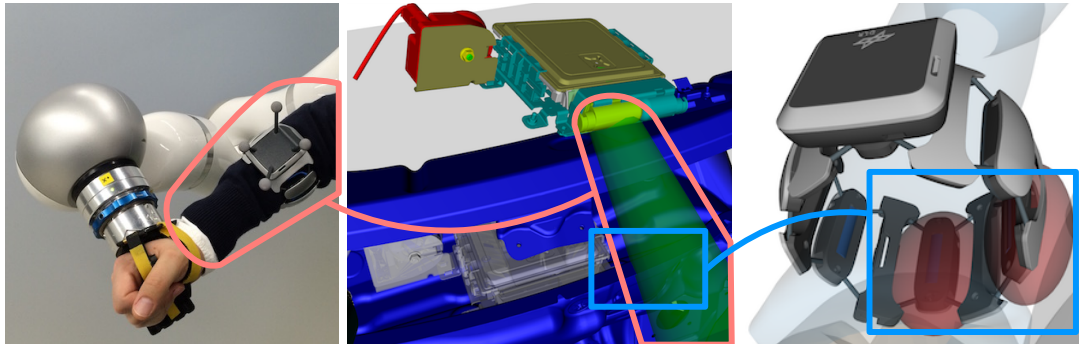


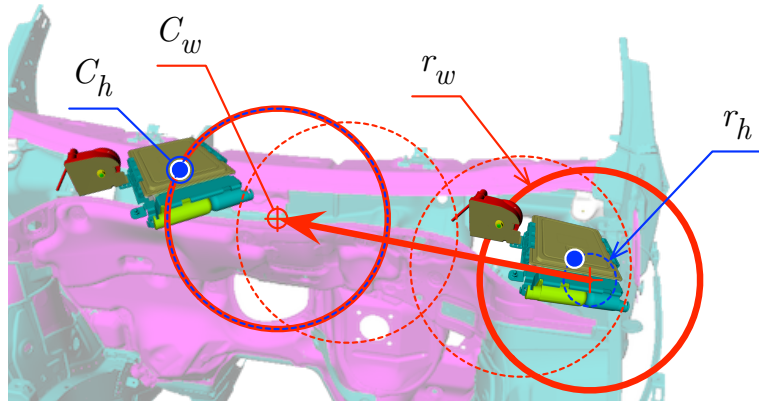
Figure 5.5: The VibroTac provides tactile feedback using six vibration motors equally distributed on the perimeter of the forearm. The markers on the device are optically tracked (left). The movement of the elbow is mapped to its virtual representation (middle, in green). If the virtual forearm collides against the scenario (middle), the corresponding segments of the VibroTac provide vibration feedback to the user (right, in red).

is decisive to know about collisions of the human operator with the virtual environment, e. g., to validate if there is enough space for the human arm inside a motor compartment. This problem can be treated by the usage of additional haptic devices. Arm-exoskeletons may pose the most realistic kind of feedback to the human arm by mechanically coupling the arm with the device.

Due to the lack of available mechanical solutions, however, an alternative solution is used to generate this additional feedback to the operator's forearm. Instead of force feedback, tactile feedback provided by a vibrotactile arm band is employed. The used tactile feedback device VibroTac [SEWP10] is a battery driven and wirelessly controllable bracelet with six vibration motors (so-called tactors). Due to the uniform tactor distribution, the location and direction of an impact can be indicated to the user's forearm. The strength of collision can be displayed by varying the intensity of the stimuli or by generating different vibration patterns, which can be controlled separately for each tactor.

As shown in Figure 5.5, reflective markers were added to the VibroTac in order to track it optically. The user sees a green transparent arm in the virtual scene; the wrist is coupled to the end effector of the HUG, whereas the elbow corresponds to the movement of the VibroTac. The HUG can provide six-DoF collision feedback at the palm and the VibroTac extends the touch sensation to the forearm: when the virtual arm collides with a certain force against the objects in the scenario, the corresponding segments are activated with the intensity suited to this virtual force. As a result, it is possible to evaluate and train assemblies where the whole forearm configuration with respect to the mounted part has to be considered.

Figure 5.6: Scenarios which are bigger than the device workspace require indexing. This is achieved by moving the virtual workspace (red sphere with center C_w and radius r_w) to the grasping point C_h which is outside the boundary.



5.2.2.3 Workspace Navigation

Interacting with virtual environments larger than the workspace of the interface requires movement mapping and control strategies. Conti and Khatib give an overview of those methods [CK05] and propose a *workspace drifting control* scheme with which the virtual workspace is shifted with a velocity controlled by the distance from the device workspace origin to the position of the endeffector. As these authors report, the most common navigation approach consists in performing *position control* with a selected *scaling* factor, or *indexing* the position of the end effector with respect to the moved avatar after the interface has been decoupled from the simulation (also known as *de-/clutching*). It is also possible to carry out *ballistic control* [SCB04], with which the scaling factor depends on the velocity of the end effector, or *rate control*, where the velocity of the avatar or moved object increases linearly with the distance from avatar to workspace origin.

[DABS06] tested some of these methods in a user study and concluded that their *bubble* technique yielded best performance results in a three-DoF painting task that required precise activities in large environments. The *bubble* technique is a hybrid *position/rate control* approach where the optimal workspace around the endeffector (a bubble or sphere) is displayed both visually and haptically to the user. Whenever the boundaries of the bubble are reached, it is moved with a velocity cubically proportional to the trespassed distance and the user feels restoring forces in the direction opposite to the movement.

As reported in [PV11], workspace visualization can become distracting in complex six-DoF assemblies, and restoring forces can lead to the perception of artifacts or they can make the discrimination of collision forces difficult. Therefore, in the implementation presented in this work it was proceeded similarly as in the *bubble* technique, but without displaying the workspace visually or haptically. The user experiences two modi with the

haptic interface: (i) fast controllable *navigation* in order to relocate the workspace (*rate control*) and (ii) *manipulation* with 1:1 mapping (unscaled *position control*).

As shown in Figure 5.6, the workspace is abstracted with a sphere with center C_w and a constant radius $r_w = 0.2$ m. Additionally, r_h is the distance between the grasping point C_h related to the object moved by the haptic interface and the current workspace center. This workspace center C_w is considered the anchor frame with respect to the device movements, i. e., the center of the real workspace of the user is mapped to this point and the moved object is transformed with respect to it.

The workspace shifting works as follows: if the grasping point of the moved object is inside the workspace ($r_h \leq r_w$), the workspace will not move ($C_w(t + \Delta t) = C_w(t)$). Otherwise ($r_h > r_w$), and in absence of collision forces, the workspace is translated towards the grasping point, as summarized in (5.1):

$$C_w(t + \Delta t) = C_w(t) + \lambda_w \Delta t \left(1 - \frac{r_w}{r_h}\right) \underbrace{\left(C_h(t) - C_w(t)\right)}_{\text{vector length } r_h(t)}, \quad (5.1)$$

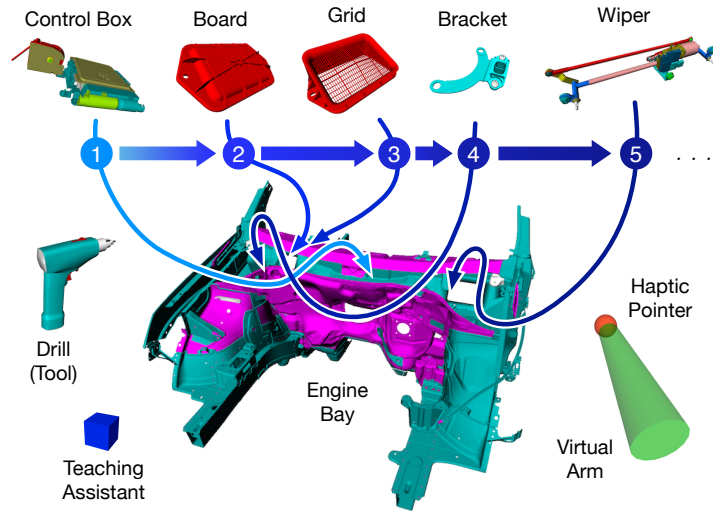
where $\lambda_w = 1.5 \text{ s}^{-1}$ is the gain and Δt the time step between two consecutive cycles. The presented method moves the workspace in the scenario if the grasped object is outside this workspace and the moved objects are not colliding; the velocity of the translation is linear to the distance from the grasping point to the surface of the workspace.

5.2.2.4 Collaboration with Additional Haptic Interfaces

Shared and collaborative virtual environments with haptic feedback are challenging due to the user-user interaction, the required synchronization, and their delay and frequency demands (1 kHz). As mentioned in Section 5.1.1, a collaborative tele-mentoring functionality is presented in [GRV⁺10]. In it, a trainer can teach a trainee how to perform specific assembly tasks: position and force data are sent unidirectionally between them (usually from the tutor to the apprentice) so that skills can be transferred faster and interactively. On the other hand, a peer-to-peer collaborative framework for assembly simulations was presented in [ICG⁺06]. The work focuses on the technical strategies required to maintain synchronicity for each peer simulation. The consistency is achieved basically by keeping track of the last valid (non-colliding) pose of object.

Since the presented framework is modularly built and the objects of the collision computation database transfer pose and contact information independently, the system allows for collaboration between users by simply connecting another haptic (or other) interface to an object node, as it is shown in Figure 5.2. In the current implementation

Figure 5.7: Schematic scenario description and assembly sequence. The user has to mount five parts into a car engine bay in a row; instructions are given in the upper left corner of the display. When no object has been grabbed, the user sees a red sphere at the right hand (haptic pointer); the left hand can carry tools, such as the drill illustrated in the picture. A teaching assistant box can be controlled by another interface.



a Falcon* and a Sigma.7† were tested successfully. The interaction occurs through an extra pointer in the scene which is moved by the second user, called *teaching assistant* (blue cube in Figure 5.7). The teacher can show the trainee how to move to perform the assemblies and correct wrong movements with small pushes. Both users see and feel collisions. Synchronicity or consistency is achieved with the ForceBuffer described in Section 5.2.1.1, as if the *teaching assistant* pointer were another object in the scene.

5.2.3 Exemplary Scenario: Car Assembly Sequence

This section introduces a practical use case in which all methods explained in the previous sections are used. The following video shows the scenario, the interaction techniques and the introduced tasks:

A Platform for Bimanual Virtual Assembly Training with Haptic Feedback in Large Multi-Object Environments @ Vimeo

<https://vimeo.com/190217267>

As shown in Figure 5.7, the virtual scene consists of ten objects:

- (i) a car engine bay where several objects can be mounted,
- (ii) a haptic pointer (red ball) symbolizing the right hand coupled to the right forearm of the HUG (Section 5.2.2.1) with which the user can grab an object to mount it,

*<http://www.novint.com/index.php/novintfalcon>

†<http://www.forcedimension.com/products/sigma-7/overview>

- (iii) a virtual forearm (transparent and green) moved with the tracked VibroTac (Section 5.2.2.2),
- (iv) five selected car parts that have to be assembled in a specified sequence: a control box, a covering board, a grid, a bracket, and a wiper mechanism,
- (v) an electric drill held by the left arm,
- (vi) and a teaching assistant cube moved by another user with a different interface than the HUG. Thanks to this object, the experienced user can haptically correct the trials of novice operators online.

All objects are real CAD parts and the interaction occurs without scaling (e.g., 1:1). It is worth to mention that the user receives only vibrotactile feedback (not force feedback) when the green transparent virtual arm collides. Therefore, the virtual arm might overlap with the other objects if the user does not actively re-configure his arm as the vibrations indicate.

The scenario is easily extensible to different or more objects by modifying the configuration scripts of the multibody framework (see Section 5.2.1.1) and the state machine (see Section 5.2.1.2). The assembly workflow of the control box is shown on the right side of Figure 5.8, and it is similar to all other objects to be assembled. First, the part to be mounted appears on the right and the user has to navigate to it. The navigation takes place thanks to the indexing explained in Section 5.2.2.3. If the user remains in contact 1 s with the object to be mounted, the red haptic pointer is replaced by it. Next, the user can navigate towards the region where a grey transparent replica of the part is shown. The following steps consist in assembling the part into its correct location. If this is achieved within the translation and rotation deviation tolerance specified in the configuration file, the part is fixed in its assembly pose and the red haptic pointer appears again. Then, after testing the space for the tool held with the non-dominant hand, the user has to keep on with the next part. Instructions are displayed in a text box on the left upper corner of the simulation frame.

A previous version of the system has also been tested with a virtual satellite maintenance scenario [SHH⁺15]. Compared to that setting, the use case discussed in the present work focusses on multibody interactions with more complex and realistic objects; additionally, the implementation of techniques that make interacting in large virtual assembly training environments possible (e.g., navigation, collaboration, and game control) is described and validated.

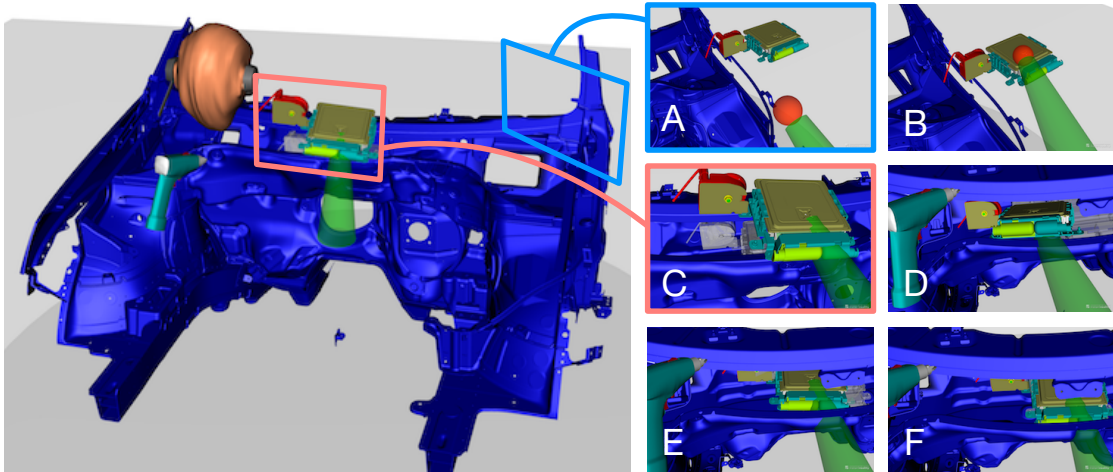


Figure 5.8: Left: Snapshot of the whole scenario; the user head is displayed by an avatar. Right: Assembly steps of the control box as seen by the user through the head mounted display moved by the head and optically tracked: (A) Approach to the control box to be assembled with the haptic pointer; (B) Touch and grab the control box; (C) Move (indexing) close to the place where the object has to be mounted (displayed with a grey transparent replica); (D) and (E) Insert the control box to the engine cavity; (F) Reach target pose. The haptic pointer appears and the user has to move on with the next object.

5.2.3.1 Performance Results

The evaluation of the assembly process of the control box presented in Figure 5.8 is shown in this section. Force, penetration, load and computation time diagrams of the results as well as the details of the used data structures are reported in Figure 5.9.

During the first 12 s, the user is in navigation mode (see Section 5.2.2.3) and shifts the workspace center to the desired position. Around frame D, the first collision occurs and the position is corrected again (13.2 s – 17.1 s), for finally proceeding with the insertion into the assembly cavity (17.1 s – end, frames E and F). As expected, it is in those collision situations when the load and the computation time increase considerably. The load value η is computed every cycle as specified in (3.50) from Section 3.3.2.1 (Chapter 3).

As mentioned in Section 5.2, the number of levels swept in the pointshell can be decreased still checking all collision regions if the *critical load* η_c is reached. This pointshell specific *critical load* can also be updated during runtime depending on the number of collision threads that are active and their respective loads. In the shown assembly experiment part, only three collision threads were active (the ones between the engine bay, the drill and the control box), and the maximum achieved load $\eta_{\max} = 18.89\%$ (at time stamp 25.86 s) was smaller than the critical $\eta_c = 70\%$. Positively and strongly correlated to the load is the required computation time, which achieves a peak of 0.75 msec around

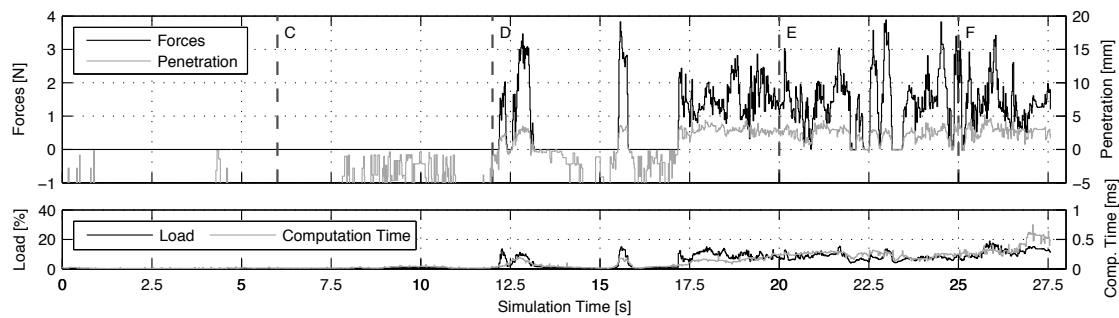


Figure 5.9: Force, penetration, load and computation time values obtained during the assembly of the control box displayed in Figure 5.9. Assembly steps C – F are marked in the upper diagram. Negative penetration values correspond to the approximate distance values used during spatio-temporal coherence computation, which are unsteadier since they correspond to sphere distances. However, these data are scaled to show penetration and force values, because they are considered to be the most relevant in assembly simulations. The car’s engine bay has a voxel edge of 3 mm, resulting in a grid of $1022 \times 788 \times 541$ elements compacted in 286 MB. It was computed from the original model consisting of 1.83M of triangles in few minutes. The assembled control box consists of 10507 points divided in 3506 clusters and 7 levels compacted in 875 KB (see Figure 5.1). It was computed from the original model consisting of 101472 triangles in few seconds. The offline computation time required for data generation depends on the resolution and selected number of layers. The used computer was an Intel(R) Core(TM) 2 Quad with CPUs at 2.66GHz and running Suse SLED 11 (not realtime).

simulation time stamp 27.07 s, below the 1 msec convention for stable and realistic haptic interaction.

Force magnitude oscillates below 4 N (maximum 3.88 N at time stamp 22.97 s) depending on the penetration of the colliding points and the reaction force applied by the user. The values shown in Figure 5.9 are the raw values computed by the collision computation module before sending them to the haptic device. Depending on the interface, filtering or similar control algorithms can be applied. Note, additionally, that the penetration values are smaller than 5 mm or even 4 mm most of the time. This penalty value is the necessary error between the displayed god object and the virtual object attached to the haptic device.

Overall, it can be considered that the results show a system able to render realtime assembly simulations with haptic feedback even in large scenarios with numerous complex geometries exported directly from CAD frameworks.

5.3 Integration into the Physics Engine Bullet

This section presents the integration of the collision computation methods from Chapter 3 into the physics engine Bullet [Cou03]. Selected interfaces provided in that publicly available engine (detailed in Section 5.3.1) were inherited (Section 5.3.2), extending the

physics engine and making the advantages of the VPS re-implementation available to any developer familiar with Bullet. Additionally, the VPS re-implementation was compared with Bullet's native GJK, GJK with convex decomposition, and GImpact, varying the resolution and the scenarios. The experiments (Section 5.3.3) show that the methods from this work perform with similar computation times as the standard collision detection algorithms in Bullet if low resolutions are chosen. With high resolutions, the VPS re-implementation from Chapter 3 outperforms Bullet's native implementations and objects behave realistically.

5.3.1 Data Structures and Workflow in Bullet

A detailed description of the data structures and the pipeline in Bullet is given in its user manual [Cou03]. Fundamental structures necessary for the integration are described here.

Being a physics engine, Bullet ultimately computes the motion of virtual objects. For that, the library builds up on different layers, from basic (math operations) to more specific (dynamics computation). The layer that handles collision detection is separated from the one that deals with dynamics computations. Therefore, it is possible to perform only collision computation with Bullet. Along these lines, the integration essentially consisted in adding and extending interfaces from the collision detection layer, while being still compatible with the layers beneath and above. Note that although Bullet supports deformable objects, only rigid bodies are considered in this work.

The highest control interface that creates the virtual world is `btDiscreteDynamicsWorld`. One can `setGravity()` to it, call `addRigidBody()`, and simulate the next instant in the world with `stepSimulation()`. After each step, it is possible to ask the `btTransform` of each rigid body. Each `btRigidBody` consists of a `btMotionState` and a `btCollisionShape`; this last class contains the geometry that is used for the collision detection.

Collision detection is divided in two phases in Bullet: (i) the broadphase, in which pairs of objects are quickly rejected based on the overlap between their axis aligned bounding boxes and (ii) the narrowphase, in which the selected collision detection algorithm is called. The interfaces `btBroadphaseInterface` and `btCollisionDispatcher` are used respectively for these two steps. The latter contains the selected `btCollisionAlgorithm`, which generates a `btPersistentManifold`, that is, the list of object points that constitute the contact manifold and which is passed to the motion solver.

Therefore, the contact manifold is the ultimate result of the collision detection process, and the rigid body dynamics simulation can work decoupled from the type of algorithm used, provided the list of contact points. Hence, the goal of the integration is to

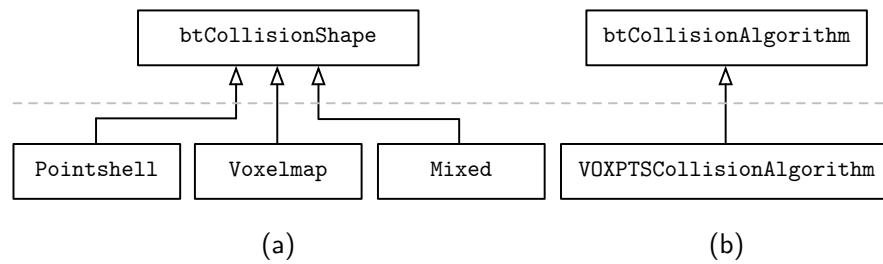


Figure 5.10: (a) Integration of the three new collision shapes into the pool of collision shapes provided by Bullet: `Pointshell`, `Voxelmap`, and `Mixed`, which contains the previous two. (b) Integration of the `VOXPTSCollisionAlgorithm` by inheriting from Bullet’s superclass `btCollisionAlgorithm`.

generate such a manifold as fast as possible, and with the best quality as possible.

5.3.2 Integration Interfaces

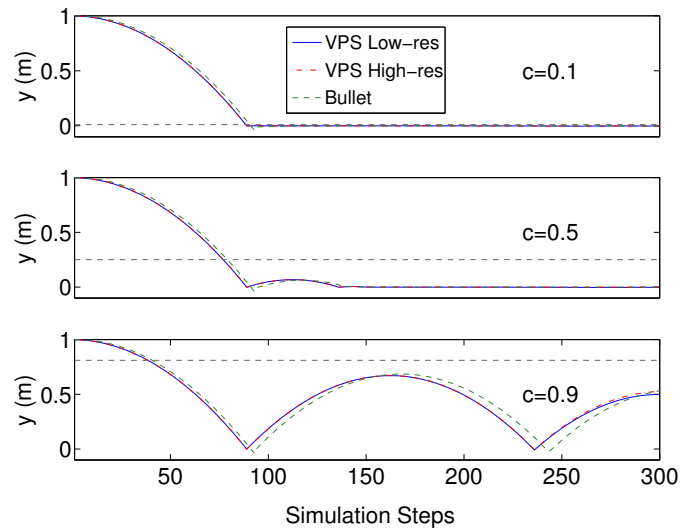
The following structures were modified by adding references to the VPS re-implementation from this work: `btBulletCollisionCommon.h`, `btBroadphaseProxy`, and `btDefaultCollisionConfiguration`. The appropriate algorithm is assigned to each shape type in the last interface.

As shown in Figure 5.10, the abstract collision shape interface was additionally inherited and extended to represent the data structures introduced in Chapter 3. Since the VPS requires two different data structures for each colliding pair, a mixed structure containing both structures for each object was defined. The structure which is used is selected automatically online: the object with less points will be the `Pointshell`. The inertia matrix and some other features required by Bullet, such as bounding volumes, are automatically created with the methods integrated in the data structures from this work.

In the broadphase, bounding spheres of the highest pointshell hierarchy level are checked against the distance fields. Colliding pairs are handled by the dispatcher, which calls the proximity query from Algorithm 3.1 explained in Chapter 3. The result of the query is a contact manifold \mathcal{M} containing colliding points with their respective normal vectors and penetration values (see Section 3.3.1.2).

If the segmented hierarchical traverse is performed (see Section 3.3.1.5), this list is already segmented in m clusters and for each one a `btPersistentManifold` is created with the point in the cluster that has the deepest penetration value. In case the regular hierarchical traverse is carried out, a unique `btPersistentManifold` is filled with the contact points, starting with the deepest point, and adding points so that the manifold area maximizes. Note that the size of the manifold is limited to four points in Bullet, although this constant value can be modified before compilation.

Figure 5.11: Height of the center of mass of the bouncing ball (radius 0.5 m) using different coefficients of restitution c . The pointshell of the sphere is composed of 275 (low resolution) and 25880 (high resolution) points. The plane’s voxelmap has a resolution (voxel edge size) of 5 mm. The black dashed line represents the ideal maximum height after the first collision.



5.3.3 Experiments and Results

In this section, first, the results of simple experiments are shown to prove the validity of the VPS re-implementation, which is compared to the default algorithms in Bullet. Afterwards, the VPS is compared with the fastest collision detection algorithm in Bullet in more challenging scenarios, using the Stanford Bunny as the benchmark object. All tests were carried out using a PC running SUSE Linux Enterprise Edition 11 with an Intel Xeon CPU at 4×2.80 GHz.

The following video shows the benchmarking experiments with the Stanford Bunny:

Integration of a Haptic Rendering Algorithm Based on Voxelized and Point-Sampled Structures into the Physics Engine Bullet @ Vimeo

<https://vimeo.com/89910579>

5.3.3.1 Tests with a Bouncing Ball

In this scenario, the height of a sphere dropped onto a plane as well as its maximum penetration are analyzed. The sphere, with a radius of 0.5 m and a mass of 1 kg, was dropped from a height of 1 m. The gravity was considered to be 10 m/s^2 and the frequency of the simulation was 200 Hz. Given the geometries, the re-implemented VPS delivered at each time step one colliding point. For this type of scenarios, Bullet calls a simple algorithm called `btSphereBoxCollisionAlgorithm`.

The results show that the height profile of the center of mass of the ball for the VPS re-implementation roughly matches the one yielded by using Bullet’s algorithm. The discrepancies between the VPS and Bullet’s values are due to Bullet having higher

Table 5.1: Maximum penetration errors (mm) in the bouncing ball experiment using Bullet and the VPS re-implementation in this work. Two resolutions are considered for the VPS algorithm: low resolution with 275 points and high resolution with 25880 points.

Restitution coefficient [-]	Penetration error [mm]		
	Bullet	VPS low res.	VPS high res.
0.1	42.5	6.8	3.5
0.5	42.5	9.5	4.0
0.9	42.5	20.1	19.2

penetration errors (see Table 5.1), which delay the rebound and increase the period of the bouncing. Having pointshell objects with much higher resolutions seems to provide lower penetration errors, but the benefits are not substantial.

Alongside the bouncing ball experiment, the stacking of similar objects was also tested. Stacking spheres and disabling freezing of the objects will cause them to collapse, eventually. As the resolution of the pointshell representation of the sphere is increased, the stack gets more and more stable. Using a coarser sampling grid to generate the pointshells intrinsically adds some quantization noise to the modeled object; that could explain its apparently more realistic behavior.

5.3.3.2 Tests with the Stanford Bunny

In this section, the VPS re-implementation is compared against other available algorithms in Bullet, varying the resolution of the Stanford Bunny or its segmentation level. Figure 5.14 illustrates some of the used data structures.

General Comparison with Bullet Algorithms Varying Resolution

Figure 5.13 shows computation time and linear velocity diagrams produced by the VPS re-implementation from this work, Bullet’s convex decomposition, and Bullet’s GImpact implementation, which is used with arbitrary non-convex triangle meshes. In the experiment, a Stanford Bunny (35606 vertices) was dropped onto a horizontal plane, as shown in Figure 5.12.

During full operation (Figure 5.13, steps 250 to 600), the VPS algorithm is $137\times$ faster than GImpact and requires on average 0.71 ms for each check using the fine resolution (34892 points). The bunny is simplified to a convex hull composed of only 42 vertices in the Bullet’s GJK implementation and to 8 convex hulls with 100 vertices each in the convex decomposition approach (equivalent to the pointshell in the VPS low resolution

Figure 5.12: Successive frames of a Stanford Bunny dropped onto a plane. This experiment corresponds to Section 5.3.3.2 and Figure 5.13. In the case of Figure 5.13, the frames match with the following steps: (a) Step ~ 50 , (b) Step ~ 175 , (c) Step ~ 200 , and (d) Step ~ 300 .

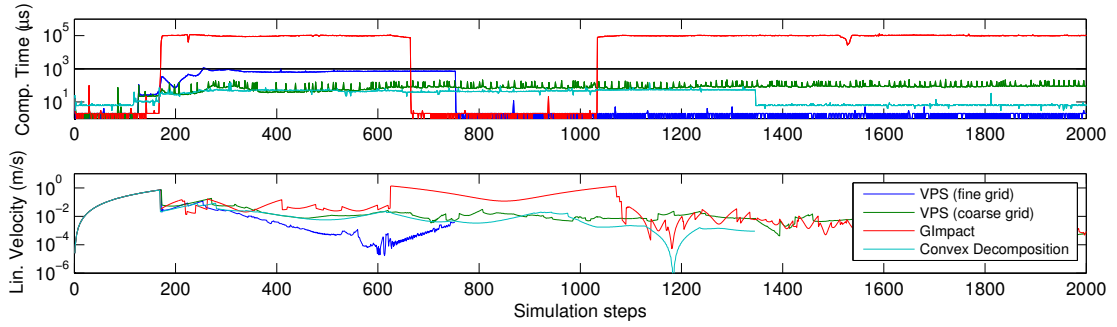
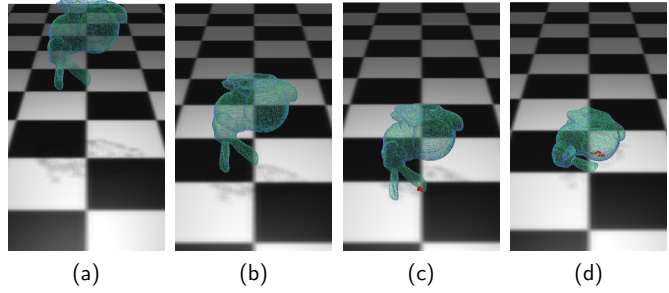


Figure 5.13: Computation time (μsec) and linear velocity (m/sec) curves in logarithmic scale for the testing scenario in which a Stanford Bunny with 35606 vertices is dropped onto a plane (see Figure 5.12). The pointshell of the bunny is composed of 799 (coarse/low res.) and 35596 (fine/high res.) points, and the voxelmap with $306 \times 305 \times 282$ voxels. The decomposed bunny consists of 8 convex hulls with 100 vertices each. Note that Bullet de-activates collision detection under certain kinetic conditions causing sudden steps in the computation time curves.

case). In these conditions, GJK is $339\times$ faster than the VPS with a fine resolution (34892 points), but the convex decomposition is only $1.3\times$ faster than the VPS with a coarse resolution (799 points).

Therefore, the integrated VPS re-implementation leads to similar computation times and higher accuracies compared to the Bullet's tested ones with low resolutions, whereas it outperforms them when the resolution is increased.

Segmented Collision Detection

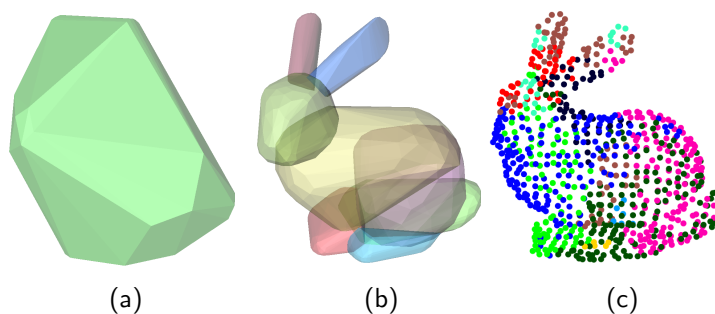
In this section the segmented collision detection method introduced in Section 3.3.1.5 is validated. The experiments used a desktop-sized Stanford Bunny of low (799 points) and high resolutions (35596 points) with a mass of 1 kg and a diagonal inertia tensor $\mathbf{J}_B = (4.79, 4.46, 6.38)10^{-3} \text{ kgm}^2$, which was dropped from a height of 0.5 m onto a plane, as shown in Figure 5.12.

In the experiment, the collision computation time for different segmentation levels (L)

Table 5.2: Computation time (μs) of the VPS when supplying different number of contact points according to the level of the point-sphere tree selected for the segmentation. Low resolution: 799 points; high resolution: 35596 points. Point-sphere hierarchies had a branching factor of $N_K = 4$ children per cluster.

Resolution	Level, L				
	1	2	3	4	5
Low (799 points)	102.2	102.1	104.2	102.3	102.8
High (35596 points)	1387.9	1477.5	1407.4	1449.6	1494.3

Figure 5.14: Different representations of the Stanford Bunny used in the tests of Section 5.3.3.2: (a) Convex hull created with Bullet; (b) Convex decomposition created using Bullet [MG09]; (c) The segmented point representation from Section 3.3.1.5 encoded by colors.



was measured. For that, the bunny was dropped with 10 random orientations, calculating the average time during the interval between steps 200 – 600, i. e, full operation or worst case. Note that for $L > 1$, the number of segments m is close to LN_K , being N_K the branching factor (see Section 3.3.1.5); additionally, the number of segments m equals to the number of possible points in the contact manifold.

The results in Table 5.2 show that the computation times are very similar for all levels. Therefore, collision computation time is barely affected when segmented collision computation is employed.

The same experiment was performed at frequencies lower than 50 Hz using the Stanford Bunny, and the penetration errors were observed. When performing the test for $L = 1$ (one colliding point) the algorithm was able to maintain the bunny above the plane until ~ 30 Hz were reached. Operating at a lower frequency would lead to the bunny falling through the plane after the initial collision was detected. The same experiment was done using $L = 4$ and $L = 5$ and the bunny was maintained above the plane for frequencies of simulation down to ~ 10 Hz, where the penetration errors were minimized if a higher level L was chosen.

5.4 Other Application Environments

The algorithms and frameworks presented so far in the current chapter and previous chapters were successfully integrated in further applications. Five of them are illustrated in Figure 5.15 (including links to videos online) and briefly described in the following sections. Technical details are left for further reading in references.

All the projects were implemented at the German Aerospace Center (DLR), mainly at the Robotics and Mechatronics Center, and in teams; the main tasks performed by the author are listed here:

- Section 5.4.1 and Section 5.4.2: Concept and design of specifications in team, integration of methods from this dissertation, implementation, management in team.
- Section 5.4.4: Concept and design of specifications, integration of methods from this dissertation in team, supervision.
- Section 5.4.3 and Section 5.4.5: Integration of methods from this dissertation, implementation of interfaces upon specifications.

5.4.1 A Virtual Reality Platform for On-Orbit Servicing Simulations

The growth of space debris is becoming a serious problem. There is an urgent need for mitigation measures based on maintenance, repair and de-orbiting technologies. Along these lines, a virtual reality application in which robotic maintenance tasks of satellites can be simulated interactively was developed [SHH⁺13] using the frameworks introduced in previous Section 5.2 and Section 5.3.

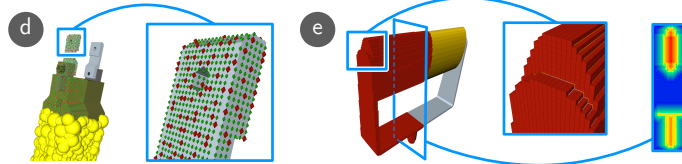
The application features a realistic on-orbit servicing scenario with multiple satellite components in it. Additionally, several common procedures are defined for the user, such as replacing a broken electronic module in the satellite, which requires a pre-defined series of steps controlled by the central logic.

The simulation provides with haptic feedback, using for that all the modules of the multibody haptic rendering framework (Section 5.2.1.1) described for the car assembly scenario. Moreover, motion of objects is simulated with the physics engine Bullet: objects can free-float in low gravity environments or move according to constraints (e. g., hinges).

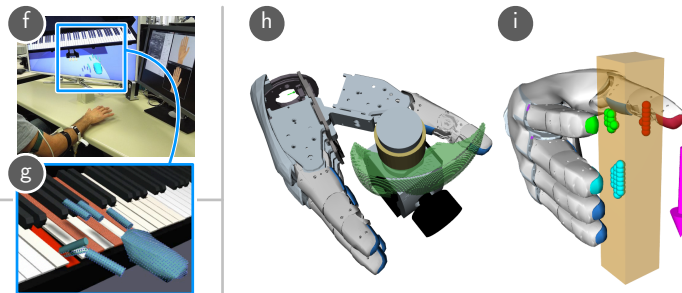
In addition to the immersive devices described in Section 5.2.2 (e. g., HUG), other interfaces have been integrated, such as the electrotactile feedback device for grasping presented in [HDE⁺16].

Altogether, the platform can be used for verification purposes, for user training with robotic systems such as SpaceJustin [KWA⁺09], or it could even be employed as a train-

A Virtual Reality Platform for On-Orbit Servicing Simulations
<https://vimeo.com/64991632>

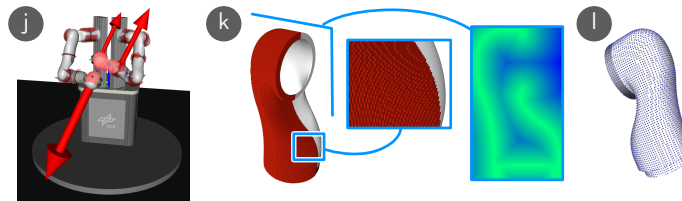


Ultraplano: Playing a Virtual Piano with Ultrasound-Imaging
<https://vimeo.com/97063714>



Shared Grasping: Semi-Autonomous Robotic Grasping Using Virtual Models
<https://vimeo.com/87159074>

Collision Avoidance of Complex Mechanisms with Themselves and the Environment
<https://vimeo.com/260950105>



Robotic Autonomous Assemblies Using Virtual Models
<http://youtu.be/2jYhdmk-pMg>

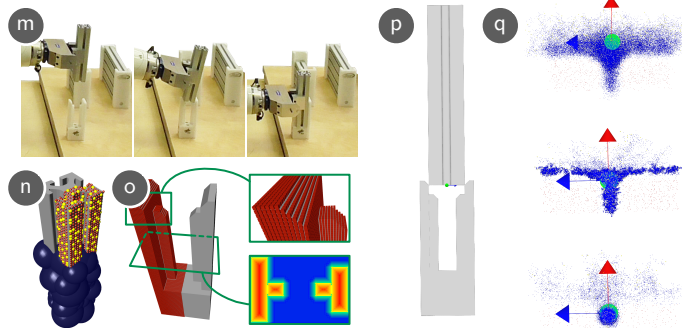


Figure 5.15: Overview of other applications and links to videos showcasing them. (a) Generic virtual satellite model. (b) Space Justin interacting via telepresence with a real replica of the virtual satellite. (c) User interacting with the virtual satellite in front of a power wall. (d) Pointshell model of the virtual gripper (one sphere level and two point levels). (e) Partially voxelized handle and a section of its embedded distance field. (f) Setup of the virtual piano: virtual scene, real hand with ultrasound probe, and finger force prediction display. (g) Virtual scene of the piano with the data structures used during the interactive simulation. (h) Voxelized model of the robotic finger's workspace. (i) Reachable independent contact regions obtained from the pointshell model of the grasped object. (j) Collision avoidance of the HUG applying the generic collision detection module based on the VPS: repulsion forces are applied to close parts (in red). (k) Voxelized model of a robot link and the section of its embedded distance field. (l) Pointshell representation of a robot link. (m) Snapshots of a robot assembly process performed autonomously in realtime. (n) Pointshell representation of the assembled virtual model (one sphere level and two point levels). (o) Voxormap representation of the assembly environment. (p) Estimation of the instantaneous real model configuration represented with virtual models. (q) Three particle samplings for configuration estimation with different convergence rates.

ing platform for autonomous robots. A full description and validation experiments are provided in [SHH⁺15].

5.4.2 Ultrapiano: Playing a Virtual Piano with Ultrasound-Imaging

Ultrapiano is the first integrated application of medical ultrasound imaging to remotely control a virtual hand able to play piano in real-time [SHGC14]. The virtual environment was powered by the engines described in previous Section 5.3 and Section 5.2.

Detecting human finger motions and forces plays an important role in teleoperation and virtual reality. Standard data gloves or optical finger tracking devices can provide reliable finger movement data, but they need to tackle elasticity and occlusion issues, respectively; additionally, both methods might require long and delicate calibration procedures. Within this project, medical ultrasound imaging was implemented as a robust technology for detecting human finger motions by predicting finger forces. These finger forces can be individually predicted using forearm cross section ultrasound images provided by a simple probe, after short and easy calibration procedures. Machine learning methods are employed during the process, in particular, incrementally updated ridge regression between images and finger forces. The system leaves the subject's hand completely free to operate.

The virtual scenario consists of a symbolic hand (palm and five fingers) controlled by the user via the novel human-machine interface (HMI) and an interactive virtual piano with two octaves. The collisions in the multibody environment are handled by the framework from Section 5.2.1.1 and the movement of the keys is simulated with the physics engine Bullet. The virtual forces between fingers and keys are used to modulate the key notes in a sound module.

This integrated system can be used as an entertainment device, for rehabilitation, and for recovery from phantom-limb pain for amputees. Additionally, the HMI has straightforward applications on the non-invasive control of prosthetic limbs. The efficacy of the system was validated in a user study [CHS⁺14].

5.4.3 Robotic Autonomous Assemblies Using Virtual Models

Typical assembly processes consist of sequences of contacts that occur when transferring objects from free configuration spaces to constrained goal configurations. Traditional autonomous robotic assemblies require precisely defined rigid trajectories that minimize those contacts.

However, another more powerful paradigm is also possible with new generation robots that integrate force-torque sensors and have impedance control architectures (e. g., the

DLR/KUKA Light-Weight Robot (LWR) [ASHO⁺07]*). Applications that follow that new robotic assembly paradigm iteratively (i) execute planned assembly steps, (ii) observe through sensors the interactions in the environment (e. g., via force sensors or cameras), (iii) understand the real world states by contrasting processed sensor data with guessed states obtained using virtual models, and (iv) actualize assembly steps according to that contrasting.

In [NSSB16], a framework which followed that new paradigm was presented. The approach proposes a Sequential Monte Carlo (SMC) observation algorithm which uses the VPS re-implementation introduced in Chapter 3 as a reference model for the contacts between complex shaped parts. One of the main contributions of the work is the extension of the classic random motion model in the propagation step with sampling methods known from the domain of probabilistic roadmap planning; that increases the sample density in narrow passages of the configuration space. As a result, the observation performance can be improved and a risk of sample impoverishment reduced.

The framework was experimentally validated with a LWR for a peg-in-hole task with a challenging narrow passage. The scenario was built with standard Item[†] parts, which are highly non-convex; in contrast to most available collision engines, the VPS re-implementation from Chapter 3 is able to handle those scenarios faster than 1 kHz, necessary for robotic applications.

5.4.4 Realtime Collision Avoidance for Mechanisms with Complex Geometries

Complex mechatronic systems need to avoid collisions with their environment and themselves if safe human-robot collaboration is to be guaranteed. In [STH18], a solution based on the multibody collision computation framework explained in Section 5.2.1.1 was presented.

That framework was extended to support generic mechanisms that are easily described in simple configuration files. In them, the roboticist has to specify the Denavit-Hartenberg parameters of the manipulator and the filenames of the data structures representing its links. Additionally, it is possible to add objects to the virtual environment, such as tools, assembly parts, or even generic human models. While the virtual robot configuration is displayed according to the readings from the real robot itself, all additional models can be tracked with markers and infrared cameras (e. g., Vicon[‡]) or depth-sensors (e. g., Kinect[§]). As a result, a virtual replica or the real world is simulated

*<http://www.kuka-lbr-iiwa.com>

†<http://www.item24.de>

‡<https://www.vicon.com>

§<https://developer.microsoft.com/en-us/windows/kinect>

in realtime, and the multibody framework computes the distances and collisions between all of them. Increasing the safety margin (see Chapter 3) of the robot links, predictive repulsion forces between the robots and the environment are computed and overlaid on the links, avoiding real collisions.

This novel collision avoidance engine abstracts generic mechanisms and environments, handling complex geometries without the need of simplification or tedious manual modifications. The framework was validated on the HUG, consisting of 14 links divided into two mechanical chains and two tools optically tracked (see Section A.1).

5.4.5 Shared Grasping: Semi-Autonomous Robotic Grasping Using Virtual Models

Fine tele-manipulations of complex robotic systems (e. g., a robotic hand) are difficult for the operator and error-prone. Therefore, adding a level of autonomy to such telepresence systems and releasing workload from the operator is desirable. Along these lines, a semi-autonomous grasp planer which integrates the collision computation methods from Chapter 3 was presented by [RHBH11] and successfully applied to the SpaceJustin humanoid [KWA⁺09].

The framework follows these steps: (i) the user commands movements with an input interface (e. g., the HUG) to the hand-arm chain of the remote manipulator, approaching the object to grasp; (ii) the object and its configuration are detected using computer vision [BLBH12], provided a data base of virtual geometries; (iii) when the robotic hand is close enough so as to grasp the object, feasible contact points and the grasp quality computed by the grasp planer online are shown to the user; (iv) with that information, the user can decide whether to execute the computed grasp by closing a one-DoF interface at the end-effector of the haptic device.

The basic data structures used by the grasp planer are voxelmaps and pointshells. In the offline phase, the workspaces (or swept volumes) of the hand fingers are voxelized, whereas the objects to grasp from the data base are represented with point clouds. Online, the intersections of both result in the contact points if fingers were closed. Then, a physically feasible subset is computed, considering, among others, the contact forces that the robotic end-effector is able to apply. The fast collision computation provided by the re-implemented VPS is a key factor for the robust and interactive operation characteristic of this planer.

Further user assistance methods were also developed within this project. [Her15] describes the whole framework as well as the user studies which validated it.

5.5 Summary, Conclusions, and Perspectives

This chapter presented the design, implementation, and experiments of application and integration frameworks that contextualize in useful environments the collision computation and force rendering methods from previous chapters. In particular, Section 5.2 introduces a bi-manual and large-scale virtual assembly (VA) platform, whereas Section 5.3 presents a plug-in for the physics engine Bullet [Cou03] which integrates the VPS re-implementation from this work. Section 5.4 briefly depicts additional applications in which the same methods were successfully applied.

In the following, the conclusions of each application are summarized and the future work directions outlined.

Virtual Assembly Platform

The presented virtual assembly training platform supports bimanual haptic interactions with several arbitrarily complex CAD objects simultaneously. Additionally, in contrast to usual desktop systems, unscaled large upper body movements can be performed and force and tactile feedback is provided to the hand and forearm, respectively. Some intuitive navigation methods from the literature based on hybrid position/rate control were adapted for a more efficient interaction in large realistic car assembly scenarios. Performance results that validate the system are also provided.

Future work contemplates the following main topics, based in part on the current limitations of the system:

- More realistic and natural bimanual interaction. Currently, the dominant hand grabs the parts to assemble and the non-dominant one holds the tool in order to check whether there is enough space for it. In the future system, both hands will grab and assemble parts; after that, users will have to fix them with the tool held with the dominant hand and, for example, using the non-dominant one as support.
- Integration of data gloves compatible with the HUG. Current hand and forearm are simplified to two objects. The future plan is to include both complete mechanical chains of the arms and hands, and to provide them with haptic feedback.

Additionally, the following Chapter 6 evaluates with user studies the virtual assembly platform consisting of the LWR. The device is compared against the Sigma.7 and the differences between real and virtual manipulations are studied using more abstract but transferable scenarios. Future work could address user evaluations of customary and realistic scenarios targeting a more practical validation.

Physics Engine Integration

The integration of the VPS re-implementation from Chapter 3 into the physics engine Bullet [Cou03] makes the algorithm available to the developer community through well-known interfaces. Furthermore, the improvements made on the data structures and methods of the VPS allow for faster and more accurate distance, penetration and penalty force computation. Consequently, the integrated algorithms perform better than the three approaches in Bullet that handle complex objects (i. e., GJK, GJK with convex decomposition, and GImpact) when medium to high resolutions are used, as needed in robotics and virtual reality applications. That improvement can be appreciated both in *computation time*, with higher resolutions, up to 137× faster than GImpact, and *quality*, as shown in the related video.

Future work could address, among other topics, modifying the Bullet force constraint solver to handle contact manifolds of variable and larger sizes. Moreover, from a practical point of view, the automatic generation of data structures within Bullet out of the initial meshes would be useful, since currently these have to be created outside the physics engine.

Other Applications

The additional applications briefly described in Section 5.4 are a clear proof of the versatility of the collision computation and force rendering methods proposed in previous chapters. It was shown that collision detection and force computation, being fundamental technologies, are needed beyond interactive virtual reality simulations with (Section 5.4.1) or without haptics (Section 5.4.2) that feature miscellaneous scenarios. Concretely, the re-implemented VPS from Chapter 3 was integrated as an essential piece of robotic applications that achieve task perception (Section 5.4.3), collision awareness (Section 5.4.4) and complex grasp planing (Section 5.4.5).

In light of the need in the community and the readiness demonstrated through the applications, future work could try to make the libraries available for other developers, in a similar direction as the Bullet plug-in. Keeping the interfaces as simple and general as possible would certainly have impact in a broader spectrum of application fields.

Chapter 6

Evaluation of Force Feedback Methods and Systems

This chapter evaluates with user studies the haptic rendering algorithms presented in Chapter 3 and Chapter 4, as well as the fidelity of the basic components of the virtual assembly system in which they are integrated (see Chapter 5).

It has been shown that force feedback is superior to visual [SWH⁺12], and tactile feedback [WSHP13] when displaying collisions. Following upon that point, two independent user studies were performed in order to answer these two main research questions:

- (i) Study 1 (Section 6.2): *Which force rendering paradigm is the most appropriate and how should it be parametrized in order to achieve best performance and contact realism during virtual manipulations?*
- (ii) Study 2 (Section 6.3): *Once the optimum force rendering paradigm is configured, which is the remaining difference between real and virtual manipulations and which components should be targeted in order to minimize that gap most efficiently?*

$N = 24$ subjects participated in each of the two studies; both were implemented following a repeated measures or within-design, and share a very similar experimental scenario, consisting of well-defined tasks related to the classical peg-in-hole benchmark. The effect of the varied factors on user performance (e. g., completion time, contacts, and muscular effort measured as electro-myographical or EMG signals) and subjective contact perception were statistically analyzed. For the first study, the tested factors were the used haptic rendering algorithm, the haptic device, and the contact stiffness. For the second,

a gradual virtualization was applied adding synthetic feedback and systems of different modalities to the environment, yielding a spectrum of exercises varying from purely real to purely virtual. The results answer the two aforementioned research questions, while leading to a guideline for better virtual simulations with haptic feedback.

This chapter uses parts from the following peer-reviewed publications written by the author of this work: [SH17a], [SH17b].

6.1 Introduction

As explained in Chapter 2, virtual manipulations with haptic feedback are appealing to medical [CMJ11] or manufacturing [Xia16] applications, amongst others, since they enable realtime training and verification simulations in a great variety of environments without the need of building the scenarios physically. Yet, virtual simulations with force feedback lead to lower performance compared to the real world experiences (see, for instance, [GSW97] and [GZC07]).

That difference in performance between real and virtual manipulations arises from the lower fidelity provided by the used interfaces and the virtual simulations. The model illustrated in Figure 6.1 helps to understand the major elements that play an important role during the perception of contacts. The internal *perceptual hypothesis* of the world is shaped with the interpretation of the *sensations* caused by the *proximal stimulus* [RA93]. When it comes to the perception of the world during assembly manipulations, the most relevant stimuli can be related to the modalities of (i) *visual*, (ii) *haptic*, and (iii) *acoustic feedback*. The stimuli of these feedback modalities cause sensations that can affect each other during the process of creating the *percept* [EB02], [HR09]; hence, a holistic comprehension of contact perception requires analyzing the feedback modalities, their subsystems, and their interaction.

In general, feedback modalities can be *real* or *synthetic*, depending on whether their origin is in the *physical* or *virtual* world. The two major underlying subsystems in feedback modalities of *synthetic* nature are (i) the *rendering* or *signal generation* and (ii) the *device* or *display medium*. Virtual simulation setups try to replicate the real physical world by means of these two constituent elements (for each feedback modality); in other words, the loss in performance and perception between real and virtual manipulations can be explained by observing the effect of these subsystems and their interplay.

In this work, special focus is set on the *synthetic* haptic modality. As explained in Chapter 2, haptic feedback is often divided in (i) *cutaneous* or *tactile* (related to the skin sensations) and (ii) *kinesthetic* or *proprioceptive* (related to the movements of the the muscles, tendons, and bones) [LK09]. In the current chapter, haptic feedback refers

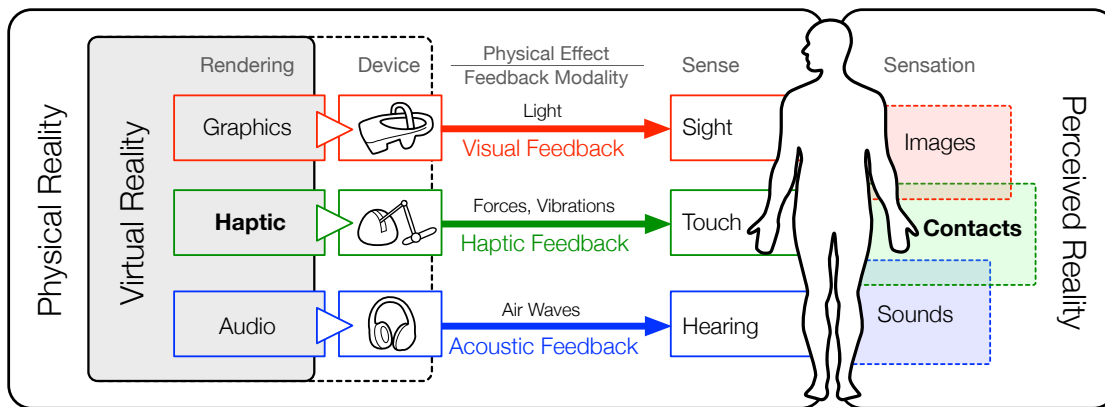


Figure 6.1: Model for virtual contact perception, focusing on the senses of touch, sight, and hearing. Virtual simulations try to replicate real world phenomena by (first) rendering and (then) displaying via devices different feedback modalities, which are sensed by human operators. The sensation of contact, related primarily to the sense of touch, can be influenced by other senses. In this chapter, Study 1 investigates the synthetic haptic modality for optimum human performance and perception during virtual manipulations, and Study 2 compares virtual and real interactions by analyzing the effects of the used rendering and device upon the user performance and perception.

primarily to the second sub-modality, and both *synthetic* aspects of *rendering* and *device* are analyzed.

Whereas Study 1 investigates the *synthetic* haptic modality for optimum human performance and perception during virtual manipulations, Study 2 provides a holistic understanding of those performance and perception constructs when different *real* modalities are replaced by *synthetic* ones. The next subsection puts in context both studies by reviewing the literature on related evaluations.

6.1.1 Related Work

Many haptic rendering methods have been developed the last decades, but the author is not aware of user evaluations in which six-DoF haptic rendering methods based on different contact rendering principles have been compared (in relationship to Study 1). A general analytical benchmark for haptic rendering algorithms was presented by Weller et al. [WSM⁺10], and seven three-DoF haptic rendering methods were compared by Rizzi et al. [RLB12], focusing on medical applications. Some analytical benchmarks for physics engines have also been published, for instance, see the work by Boeing and Bräunl [BB07].

On the other hand, several user evaluations in the literature focus on comparing real manipulation experiences with virtual ones (in relationship to Study 2). Along these lines, a pioneering work was presented by Gupta et al. [GSW97]. The authors compared real and virtual assemblies of a square peg in a multimodal environment. Force

feedback was provided with two Phantom* devices. The trends when varying difficulty (through friction, handling distance, and hole clearance) were similar to those in the real environment; however, users required roughly $2\times$ the time in the virtual environment to accomplish the task. Analog results were obtained by Yoshikawa et al. [YKY03] with a similar scenario and by Garbaya et al. [GZC07]. The latter authors used a Cyber-Grasp† and a cylindrical peg object. A real peg-in-hole task was compared with a virtual one using a magnetic levitation haptic device by Unger et al. [UNT+02]. Although virtual trials required roughly $3\times$ longer completion times, the authors concluded that the haptic display improved significantly the task performance compared to vision feedback alone. Bashir et al. [BBT04] carried out a very interesting user study in which many interaction modes were tested with an exercise that included tasks like pick, place, slide, and insertion. The real mode was compared to virtual modes with varying visual and haptic feedback conditions. They detected that the visual feedback had little effect on the performance results, whereas force feedback increased 45% the required completion time. In another study by Lim et al. [LRD+07], it was shown that design elements such as chamfers can significantly contribute to improve completion time up to 33% in virtual scenarios. Some of the authors in that work concluded together with another research team [GBMCL+14] that the use of haptic geometric constraints in combination with collision feedback helped reduce from $7\times$ to $3\times$ the completion times in virtual assemblies compared to real ones. As the reader might notice, most of the works have focused on comparing completion times and different results have been found, probably because many other factors play an influencing role on it: scenarios, haptic devices, type of visual feedback, haptic aids, etc.

The performance of haptic interfaces has been object of study as well. A 3D assembly of a puzzle was used to test three desktop-size devices by Harders et al. [HBA+06]. No significant differences in terms of completion time were detected, but virtual forces were significantly higher if no force feedback was provided to the user. Samur et al. [SWSB07] presented and validated a testbed for haptic devices. The authors aimed to complement the task taxonomy from Bowman et al. [BH99] for haptic interactions with well defined exercises; these included travel and selection, selection and manipulation, force discrimination, texture discrimination, and shape identification.

Finally, the perception of virtual contacts has also been studied by several researchers. Rosenberg and Adelstein [RA93] proposed (and validated) the decomposition of the contact realism perception of a wall into three salient features: (i) *crispiness* of initial contact with the surface (related to the damping), (ii) *hardness* of rigid surface quasi-static interaction (connected to the stiffness), and (iii) *cleanness* of the final dynamic release

*<http://www.dentsable.com/haptic-phantom-desktop.htm>

†<http://www.cyberglovesystems.com/cybergasp/>

(associated to the stiffness and the directional damping). Not only the parameters of the simulation need to be adjusted to optimize these features, but, as pointed out by the authors, also the haptic devices need to be designed to work with really high dynamic ranges: they have to be as transparent as possible in free movement and as hard and realistic as possible in contact. On the other hand, Kuchenbecker et al. [KFN06] showed that the superimposition of pre-computed high frequency decaying transients triggered by contact events helps to increase contact realism, probably more than object stiffness. In a related sense, the user study by O'Malley and Goldfarb [OG04] suggested that the human perception capability of detail reaches its limit around 400 N/m. Therefore, it seems that the implications of the selected virtual stiffness in human haptic perception are probably not straightforward.

6.1.2 Contributions

The contributions of this chapter are linked to the two research questions posed at the beginning. The two performed user studies answer them by exploring four or more objective variables (related to trajectories and completion time, contacts, and muscular effort) and more than five subjective constructs (related to the perception of contacts, realism, ergonomics, and workload). This is in contrast to most user studies found in the literature, which in general analyze only the differences in completion time and few perception aspects. Additionally, whereas most reviewed works typically collate the data of less than a dozen of participants, in the studies reported in this chapter $N = 24$ subjects performed systematically permuted and well-founded exercises in a within-design experimental setting.

The particular insights of each study are summarized as follows:

1. Study 1 from Section 6.2 presents the evaluation of the haptic rendering algorithms from Chapter 3 and Chapter 4. In addition to those two haptic rendering paradigms, two haptic devices were used, the HUG [HHK⁺11] and a Sigma.7 [THH⁺11] (see Appendix A), and the force stiffness was also varied with maximum and half values possible for each device. The results show that the constraint-based haptic rendering algorithm (Chapter 4) with a lower stiffness than the maximum possible yields the most realistic contact perception, while keeping the visual interpenetration between the objects roughly at around 15% of that caused by penalty-based algorithm (i. e., non perceptible in many cases). This result is even more evident with the HUG, the haptic device with the highest force display capabilities, although user ratings point to the Sigma.7 as the device with highest usability and lowest workload indicators.

2. Study 2 from Section 6.3 analyzes the effects of the following factors on the user performance and perception during virtual assemblies, compared to real assemblies: (i) a visual feedback system consisting of an nVisor head-mounted display, (ii) the haptic device HUG, and (iii) the constraint-based haptic rendering algorithm from Chapter 4. Besides that, the influence of (iv) real collision sounds is also examined to a shorter extent. The mentioned synthetic factors gradually replaced in five degrees or steps the real feedback sources, ending up in completely virtual assembly simulations. In order to explain subjective perception also with objective measures, reaction times of a secondary audio task performed in parallel with the assembly exercises were recorded, too. In general, the haptic feedback modality turned out to have the largest impact on the dependent variables, particularly the HUG interface, whereas audio cues seemed to be less significant. In relationship to the insights from the first study, it seems that virtual manipulations could be closer to real ones using interfaces that display lower stiffness values in favor of lightness.

Those and further qualitative statements are quantified within the domain defined by the used systems and methods. Moreover, the relationship with the insights from related literature are discussed, and their projections are outlined. Altogether, both studies provide guidelines for mapping properties of haptic algorithms and devices to user performance and perception.

6.2 Study 1: Evaluation of Haptic Rendering Methods with Varied Haptic Devices and Stiffness Values

This section presents the first evaluation study previously introduced. In it, the effects on the user performance and perception caused by the penalty-based and the constraint-based haptic rendering algorithms from Chapter 3 and Chapter 4, respectively, are analyzed. Even though both algorithms have been proven to provide perfectly valid analytical forces, early tests seemed to show different effects on user performance in specific contact configurations that needed further research. Given the lack of directly related studies in the literature, the presented user evaluation was accomplished, in which $N = 24$ naïve participants carried out well defined manipulation and assembly tasks in a virtual scenario after performing them as a reference in the real world. In a piloting phase, the additional factors of haptic device, stiffness, scaling, and friction were considered. From them, the haptic device and stiffness were selected for the present study, since they were identified to have the highest effects in combination with the haptic rendering methods. Besides that, it is considered that with these selected factors it is easier to extend and

apply the results and guidelines concluded in this section to other similar interfaces with different force stiffness capabilities.

Objective data were recorded (i. e., trajectories and contact information) to evaluate user and algorithm performance, as well as subjective ratings related to perceived contact realism, ergonomics and workload. The results show which is the relationship between the factors of the triplet *rendering-device-stiffness* and shed light on how to optimize the performance and the contact realism perception in interactions with haptic feedback.

Although in some cases sufficient application effectiveness could be achieved with lower fidelities [BM07], it is considered essential to shape the user response manifold as a function of the selected factors. In this sense, this first study is a preceding but essential step for the overall application of skill transfer in virtual assembly trainings. In particular, it is considered especially interesting to improve contact realism perception, as done by previous researchers (e. g., [RA93] and [KFN06]).

Section 6.2.1 presents the experimental design, elaborating on details related to the properties of the haptic rendering algorithms, the apparatus, and the hypotheses. The statistical analysis of the collected data and its discussion are dealt with in Section 6.2.2. Finally, Section 6.2.3 summarizes the most important insights, relating them to the literature presented in Section 6.1.1.

6.2.1 Experimental Design and Implementation

Figure 6.2 outlines the whole study design and the setup, which are more extensively described throughout this section.

As mentioned, the *penalty-based* haptic rendering algorithm based on VPS presented in Chapter 3 and the *constraint-based* god object heuristic from Chapter 4 were tested in the first study. The hypotheses related to them are discussed in Section 6.2.1.2 and Figure 6.3 shows the used data structures.

The tasks carried out by the participants are presented in the following Section 6.2.1.1 (see Figure 6.4), and Section 6.2.1.3 gives details on the followed procedure.

6.2.1.1 Tested Scenario: Tasks and Exercises

Figure 6.4 shows the real and virtual models used in this study, their measurements, and the tasks performed with them in the virtual scenario. An exercise is defined to be a sequence of three different peg-in-hole tasks that try to abstract common manipulation scenarios and maximize the generalization of the results. Each task had to be started by hitting a yellow box on the right proximal corner of the assembly model; the box blinked green on contact and immediately turned red, remaining so during the task. For finishing each task, the participants had to hit the red box again, which turned back

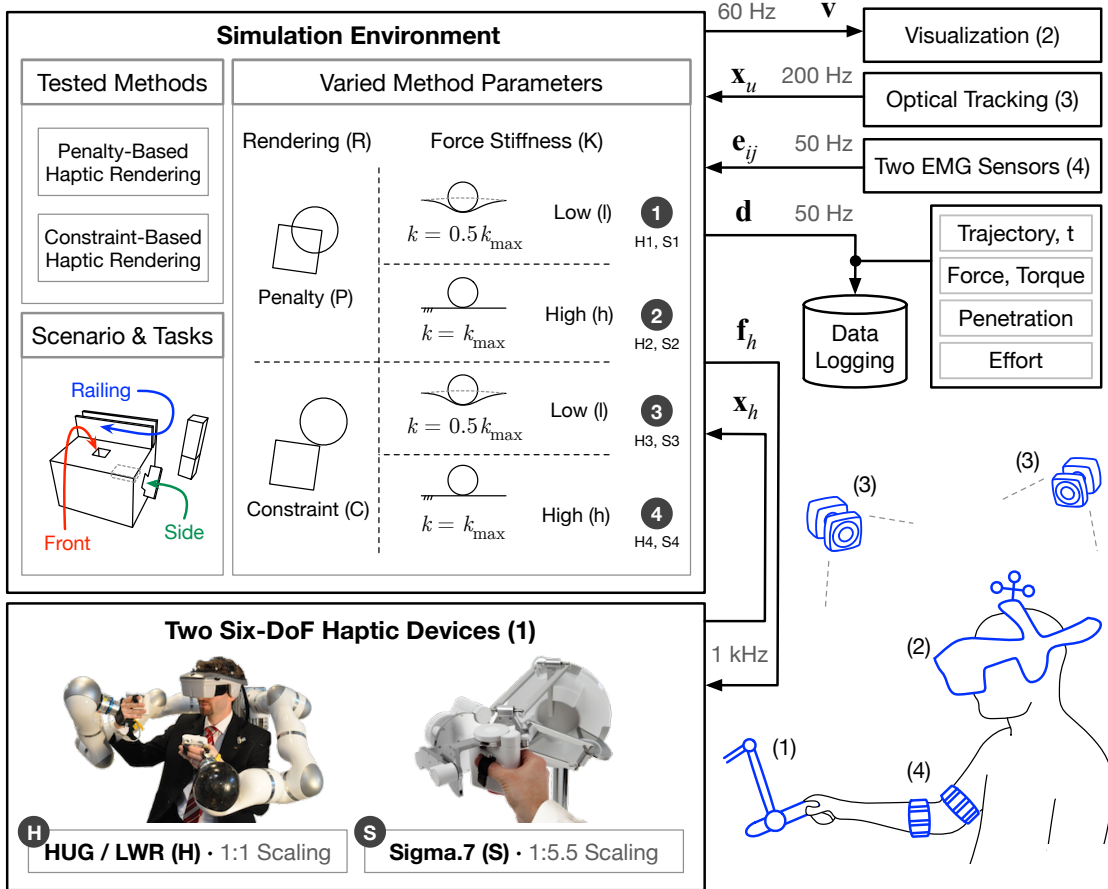
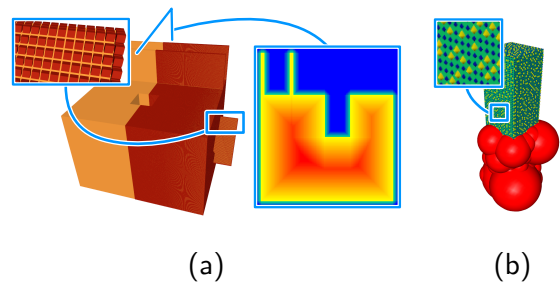


Figure 6.2: General diagram of the setup and the varied factors (device, rendering, and stiffness) in the first user evaluation. The $N = 24$ participants performed with two haptic devices (HUG and Sigma.7) four exercises, each one consisting of the tasks of frontal insertion, side insertion and railing. In each of the exercises, the rendering method (Penalty or Constraint) and the stiffness (Low or High) were varied. The order of all factor levels was systematically permuted. The users controlled with an optically tracked head mounted display the 3D camera view they had. In addition to the trajectory and the generated forces, the muscular EMG signals of the participants were measured with two Myo armbands.

Figure 6.3: Data structures of the virtual models used in the user studies: (a) Partially voxelized representation of the assembly model used in this chapter and its sagittal section with color-coded signed distance values ($s = 1$ mm voxel edge, $253 \times 383 \times 283$ voxels). (b) Point-sphere hierarchy structure of the peg object used in this chapter: it consists of 5513 points distributed into 1842 clusters classified in 8 levels (two consecutive point levels and a sphere level are shown).



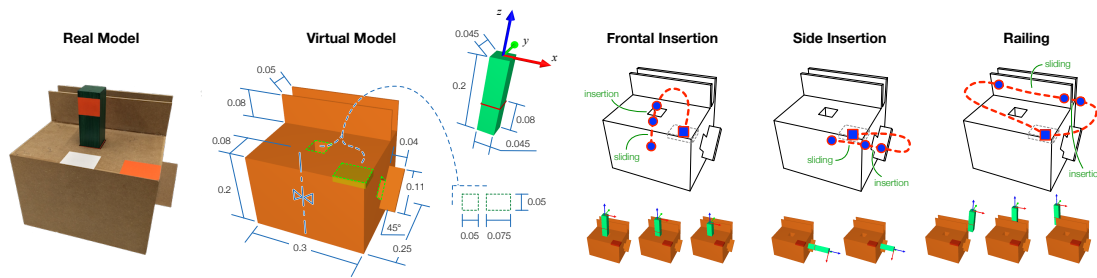


Figure 6.4: Models and tasks in the scenario of the first user study (measurements in meters). The participants had to perform the tasks with an identical real model before interacting with the virtual. Each exercise consisted of a combination of three peg-in-hole tasks: frontal assembly, side assembly, and railing. All holes had a clearance of 5 mm and a depth of 8 cm, marked with a red ring on the peg. The participants had to hit a yellow corner-box of the model for starting and finishing the exercise, and between the tasks; the box turned red during the exercise.

yellow again. Short pauses were allowed between tasks and removed from the evaluation. The participants were instructed to perform the tasks (*first*) with the lowest contact forces possible and (*second*) as fast as possible. In the following, the tasks and their properties are described (see Figure 6.4):

- (i) **Frontal Insertion:** This task is the common peg-in-hole scenario [GSW97]; the green square peg needs to be inserted into the upper hole, until the bottom of the hole is touched (a red ring on the peg shows the depth of the hole). Similar exercises have been long tried by many other authors, for instance, in testbeds for virtual environment applications [LKG⁺94], in benchmarks for human-machine interfaces [SWSB07], or even for measuring human sensory-motor skills [NB04].
- (ii) **Side Insertion:** This task is similar to the previous one, but the hole is located on the right side of the model. Hence, matching the required orientation is more difficult and the visibility is not as good as in the frontal insertion. Analogous exercises have been suggested [PWB197].
- (iii) **Railing:** In this task, the green peg must be translated between two walls from the right to the left. Thus, it consists in a constrained translational task, as proposed in [WAH⁺13].

The peg was coupled to the end-effector of the haptic device at all times; in other words, no additional grabbing/releasing interaction was implemented, and the peg fell when the user hand let the end-effector loose.

The three tasks require *positioning*, *insertion*, and *sliding* in different contact and manipulation configurations. Furthermore, they cover several important tele-manipulation task categories identified by other authors [BDW⁺03], [Dem07]. In general, all tasks in-

volve *manipulations* with *manual dexterity* for controlling *translations* and *orientations*. In particular, the initial positioning requires *gross movements* to bring the peg from the corner to the target hole and the subsequent insertion is more related to *fine motor control*. On the other hand, both insertion and sliding subtasks require a higher *perception* or awareness of the forces that are occurring. It is worth mentioning that all tasks consist of rectangular peg-hole configurations of 8 cm depth and 5 mm clearance (in VR units). This level of difficulty was maintained constant.

6.2.1.2 Apparatus and Varied Factors

As mentioned at the beginning of this section, Figure 6.2 illustrates the main components of the apparatus; their characteristics are described on the following lines.

- (1) Two haptic devices were used in the first study: the HUG [HHK⁺11] and the Sigma.7 [THH⁺11]*, both able to provide the user with six-DoF force feedback. The former consists of two DLR/KUKA Light-Weight Robot (LWR) arms[†], but only one was used during the study. This bimanual device was developed at the DLR and it is characterized by its large workspace that covers the whole upper body (maximum arm span of 0.9 m) and the high forces that it can display (peak values of 150 N). The mass of its moved parts is $m_{\text{HUG}} = 14 \text{ kg}$. The second is commercialized by Force Dimension and has a smaller workspace (a sphere of about 0.12 m diameter) and force capabilities (maximum forces of 20 N); however, since it was designed for medical applications, the fidelity and transparency of the system are significant in the state-of-the-art of haptic devices. The weight of its moved parts is $m_{\text{Sigma}} \simeq 1.55 \text{ kg}$. Since the Sigma is mainly controlled by hand and wrist movements, an arm-pad was disposed for resting the elbow and stabilizing the movements. Both devices were impedance controlled. Although the gravitational forces of the moved parts are compensated during the interaction, the mass is an important indicator for the transparency levels they can achieve. A more detailed description of the devices is provided in Appendix A.
- (2) The visualization was powered by the InstantPlayer[‡] engine from Fraunhofer IGD and displayed with an nVisor SX60[§] from NVIS. The standard eye separation of 63 mm was adjusted for each participant for an optimum 3D vision.
- (3) A Vicon Bonita[¶] optical tracking system was used to track head movements. On

*<http://www.forcedimension.com/products/sigma-7/overview>

†<http://www.kuka-lbr-iiwa.com>

‡<http://www.instantreality.org>

§<http://www.nvisinc.com>

¶<https://www.vicon.com/products/camera-systems/bonita>

account of the tracking, the participants could move the camera view intuitively in the scenario.

- (4) Two Myo armbands* from Thalmic Labs were used for recording the electromyographical (EMG) signals of the upperarm and the forearm. Since each person has a unique pattern, the signal values prior to the exercises with each device were calibrated. To that end, the participants had to relax and leave their arm hanging (minimum reference value) and then lift a 2 kg weight with their hands for about 5 s, stretching out their arm straight with 90° between arm and chest on the frontal or coronal plane (maximum reference value). The scalar effort signal was synthesized as the two-norm of the 2×8 values streamed by the armbands. For the evaluation, the recorded effort signal was divided by the maximum calibration value after subtracting the minimum one.

Varied Factors: Device-Rendering-Stiffness

Three factors or independent variables, with two treatment levels each, were controlled:

- (D) Haptic **Device**: *HUG* (H) and *Sigma* (S). The haptic device HUG was selected because it is particularly well suited for unscaled interactions. The Sigma.7, on the other hand, is a commercial desktop device that ranked as the optimum for many performance criteria in a previous study where five interfaces were compared [Sch15]. This contrast has the additional function of bridging the majority of desktop devices with the HUG haptic device, developed at the DLR institute where this thesis was carried out. It is worth to point out that the HUG had a scaling of 1 : 1, whereas the Sigma, with its smaller workspace, required a scaling of 1 : 5.5 for all tasks to be accomplished, with the displayed forces also being scaled. One could assume that the time to complete decreases inversely to the used scaling and that the matching tasks can be performed better with lower scalings [TSEC94]. In a piloting phase, the HUG was also tried with a 1 : 5.5 scaling with several participants, precisely to evaluate those aforementioned hypotheses more thoroughly; but unfortunately, the disturbing effects of moving such a large robot in such a small workspace were too high and it was decided to remove that configuration. Nonetheless, the comparison is still considered to be fair, since the device workspaces were optimally configured and the difference in scaling size is taken into account in the discussion to avoid misleading conclusions.
- (R) Haptic **Rendering** Method: *Penalty*-based (P) and *Constraint*-based (C). Both algorithms render six-DoF forces with 1 kHz. Although the general force stiffness of

*<https://www.thalmic.com>

both methods was calibrated to be the same, each algorithm has its particularities. The penalty-based approach tends to round sharp edges, whereas the constraint-based method sharpens all edges, sometimes even reducing the slipperiness of smalls corner contacts. Therefore, it was expected that the penalty-based approach facilitates delicate insertion tasks with edge and corner contacts, while increasing the object overlap precisely in those contact configurations. In the piloting phase, it was detected that these effects were sensitive to the used stiffness, therefore, the following third factor was chosen.

- (K) Force **Stiffness**: *High* (h) or the maximum stiffness supported by the device ($k = k_{\max}$), and *Low* (l) or half of the maximum stiffness supported by the device ($k = 0.5k_{\max}$). These maximum (optimal) values were obtained empirically reaching the stability boundary of each device and decreasing that limit stiffness roughly 20% for an optimal performance. The chosen values were: $k_{\max, \text{HUG}} = 3700 \text{ N/m}$ and $k_{\max, \text{Sigma}} = 2700 \text{ N/m}$.

The combination of all levels yields four exercises or conditions for the HUG (H1 – H4) and another four for the Sigma (S1 – S4).

6.2.1.3 Sample, Procedure, and Collected Data

A total of $N = 24$ participants with marginal or no virtual assembly experience were recruited. The statistically standard participant was male (2 female), right-handed (3 left-handed), and on average $M_{\text{age}} = 25.25$ years old (median $Md_{\text{age}} = 25$ years). All participants had a university degree or were undergraduate students. All subjects read and signed a consent form and they were not paid for their participation.

The procedure followed with each participant can be summarized in the following steps:

- #1 Standardized **instructions** were given to the participants in form of slides. The purpose of the study was explained, the procedure, and a video of the tasks. Additionally, the participants learned how to use the devices (~ 15 min).
- #2 The **consent form** and a demographic and experience questionnaire were filled out by the participants (~ 5 min).
- #3 The participants carried out a trial with the **real models** for priming real contact perception as reference (~ 1 min).
- #4 The EMG armband **calibration** was performed (~ 1 min).
- #5 A first **learning test** with the haptic device and the virtual model was performed, not recorded (~ 2 min).

#6 The participants performed $4 \times$ **virtual exercises** with varied conditions. After each exercise, the items of the **perception questionnaire** were orally asked ($\sim 4 \times 1.5$ min).

#7 Finally, the participants were asked to fill out the **haptic device questionnaire** (~ 2 min).

The sequence of steps #3 – #7 was carried out twice, once for each of the haptic devices, which yielded the total eight exercises mentioned beforehand (H1 – H4 and S1 – S4). All factors were systematically permuted between the subjects and each participant had also an own task order or sequence which was maintained constant for all eight exercises.

The perception questionnaire asked in step #6 consisted of two questions explained to the subjects during the instructions (step #1), and a number in a seven-point Likert scale was requested for each:

- **Contact Realism:** “*How realistic were the contacts compared to the real model?*”
- **Penetration:** “*How big was the object overlap between the objects in contact?*”

The haptic device questionnaire in step #7 was analogously explained and consisted in rating three dimensions related to ergonomics and two to the workload:

- Ergonomics [seven-point Likert scale each]: **Usability** of the device (“*comfort and likelihood of frequent use*”), **Movement Restriction** by the device (“*physical constraint or impediment*”), and **Inertia** of the device (“*awareness and weight of the device during free movement*”).
- Workload [1 – 20 scale each]: **Physical** (related to “*the corporal ease when exerting movements*”) and **Mental** (related to “*understanding and planning strategies for optimally fulfilling the exercises*”).

Since the focus of the study does not lie on analyzing the system usability and task overload with each device, it was decided to use these shorter questionnaires instead of standardized but longer ones. This choice shortened the experimental session and facilitated the analysis of the results without loss of generality, since in previous studies at DLR it has been experienced that different dimensions of more elaborate questionnaires are often strongly correlated.

In addition to the subjective ratings, the following objective values were recorded at 50 Hz: trajectory along time, generated virtual forces and torques, contact data (i. e., penetration, number of colliding points, etc.), and muscular effort signals.

Table 6.1: Descriptive data of the subjective dependent variables for each of the devices. Average and standard deviation values are provided, as well as a histogram. Ergonomy variables were coded in a seven-point Likert scale (1: very low, 4: moderate, 7: very high) and workload variables in a 1–20 scale (1: very low, 20: very high). The statistical analysis can be found in Table 6.2.

	Perception of Ergonomy [1 – 7]							Perception of Workload [1 – 20]																											
	Usability	1	2	3	4	5	6	7	Restriction	1	2	3	4	5	6	7	Inertia	1	2	3	4	5	6	7	Physical	Q1	Q2	Q3	Q4	Q5	Mental	Q1	Q2	Q3	Q4
HUG (H)	4.42 (1.25)	2	4	5	8	5	6	4	6	7	1	4.63 (1.24)	1	4	6	5	8	11.79 (2.75)	3	12	8	1	7.21 (3.36)	4	12	5	3								
Sigma (S)	5.38 (1.01)	2	1	9	10	2	5	11	4	3	1	2.42 (0.93)	2	14	5	2	1	5.75 (2.57)	9	12	3	3	7.50 (3.40)	6	11	5	2								

6.2.2 Results and Discussion

In this section, the descriptive results and their inference analysis are provided, in addition to a discussion of their implications. All relevant values are in tables (including means, standard deviations, statistics, significances and effect sizes). For the sake of clarity, it is avoided introducing all these values in the text; the reader is encouraged to look them up in the referenced tables. Additionally, Section 6.2.3 synthesizes the most important take-home messages and their impact is discussed. These conclusions are referenced throughout the text (e. g., →L1.1).

The statistical analysis of the data was performed in R. Due to the lack of packages that perform three-way repeated measures analyses independently of the distribution of the data, parametric ANOVAs (Analysis of Variance) were performed using the Fisher distribution. In that cases, normality and homoscedasticity conditions were assumed after a visual inspection of the plotted data; It is trusted in the robustness of the parametric methods in punctual violations. However, the subjective device ratings were analyzed with the non-parametric Wilcoxon signed-rank test, given that it was available and matched with the nature of the sample. The significance level was set at $\alpha = 0.05$, but p -values $0.05 < p < 0.1$ are considered to be tendencies.

The following video shows the scenario, the exercises, and the interaction with the haptic devices with different configurations:

Evaluation of a Penalty and a Constraint-Based Haptic Rendering Algorithm with Different Haptic Interfaces and Stiffness Values @ Vimeo

<https://vimeo.com/199969376>

The text continues first with the haptic device subjective ratings (Section 6.2.2.1), since they reveal general information. Then, objective and subjective data related to the exercises are presented and commented (Section 6.2.2.2).

6.2.2.1 Haptic Devices: Ergonomy and Workload

In this section, the subjective dependent variables of ergonomy (*Usability*, *Restriction*, *Inertia*) and workload (*Physical W.*, *Mental W.*) related to each of the blocks of four exercises are presented: HUG (H1 – H4) and Sigma (S1 – S4).

Device Descriptives and Analysis (Table 6.1, Table 6.2) ◇ The Sigma device seems to have the highest ergonomy values and the lowest workload ratings. Significant differences ($p < 0.01$) between devices were detected for *Usability*, *Restriction*, *Inertia*, and *Physical Workload*, whereas the *Mental Workload* experienced by the participants seems to be similar for both devices ($p > 0.05$). Concretely, *Restriction*, *Inertia* and *Physical*

Table 6.2: Statistical analysis of the subjective dependent variables to determine the effect of the device on them. Sample size (N), Wilcoxon statistic (V), p -value, Cliff's δ , and the relation (with coded effect size) between treatment levels are provided. The source descriptive data can be found in Table 6.1.

	Ergonomy			Workload	
	Usability	Restriction	Inertia	Physical	Mental
N	24	24	24	24	24
V (Wilcoxon)	19.5	300	224	300	136.5
$p(> V)$	0.00184	1.66e-05	1.54e-04	1.88e-05	0.755
sig.	**	***	***	***	.
δ (Cliff)	0.44	0.83	0.82	0.88	0.07
Relation	S > H	H \ggg S	H \ggg S	H \ggg S	H \approx S
Significance codes (p): 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1					
Effect size codes (δ): 0 ' \approx ' 0.15 ' \gtr ' 0.3 '>' 0.5 ' \gg ' 0.7 ' \ggg ' 1					

Workload have very high effect size values ($\delta > 0.8$). This is reflected in the means of the ratings, which are roughly $2\times$ higher for the HUG in those variables (\rightarrow L1.1). In contrast, the *Usability* of both devices is closer to each other (Sigma +22%) (\rightarrow L1.2).

Device Correlations (Table 6.3) \diamond All device variables are significantly ($p < 0.001$) and strongly correlated ($|\rho| > 0.5$) with each other except for *Mental Workload*. In particular, *Usability* has a negative correlation with all others, as expected. The strong correlation that *Physical Workload* has with *Restriction* ($\rho = 0.73$) and *Inertia* ($\rho = 0.67$) suggests those properties are fundamental if the perceived workload of the user needs to be decreased.

The difference between *Restriction* and *Inertia* to the participants was defined in a standardized manner (see Section 6.2.1.3); the first is related to kinematic *hard* constraints (e. g., workspace limits) and the second to *soft* weight or damping perceptions during free movement. However, the strong correlation ($\rho = 0.76$) between both might point to the fact that probably both perceptions were mixed.

6.2.2.2 Exercises: Performance and Contact Perception

In this section, the results of the objective (performance) and subjective (perception) dependent variables with varied factors of haptic device (**Device**, D), haptic rendering method (**Rendering**, R), and force stiffness (**Stiffness**, K) are presented. The objective variables are the time to complete (*Time*), the average force on contact (*Force*), the average penetration on contact (*Penetration*), and the average effort (*Effort*). On the other hand, the subjective variables consist in the *Perceived Contact Realism* and the *Perceived Penetration*.

Table 6.3: Spearman correlations (ρ) and respective significance values (p) between subjective variables related to the haptic devices (without differentiation between HUG and Sigma).

		Restriction	Inertia	Physical W.	Mental W.
Usability	ρ	-0.58	-0.53	-0.52	-0.27
	p	1.89e-05 (***)	9.53e-05 (***)	1.64e-04 (***)	0.0594 (.)
Restriction	ρ	–	0.76	0.73	0.18
	p		3.35e-10 (***)	4.02e-09 (***)	0.204 (.)
Inertia	ρ		–	0.67	0.12
	p			1.41e-07 (***)	0.435 (.)
Physical W.	ρ			–	0.24
	p				0.105 (.)

Significance codes (p): 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘.’ 1
Correlation codes (ρ): 0 ‘none’ 0.1 ‘small’ 0.3 ‘moderate’ 0.5 ‘strong’ 1.0

Exercise Performance Descriptives and Analysis Overview (Table 6.4, Table 6.5) \diamond The results of the objective performance variables were computed for the total exercise and the task segments (*Frontal Insertion*, *Side Insertion*, and *Railing*). ANOVAs were conducted for each dependent variable of all tasks and concluded that the total exercise suffices for the most important insights. Therefore, the most relevant differences between tasks are briefly commented here, but the rest of the subsections deal with the values related to the total exercise.

Unlike the other performance variables, the total *Time* can be broken down to the summation of the times related to each task. *Side Insertion* is the task which took longer on average, followed by *Railing*, and finally, *Frontal Insertion*. Observing the other variables, *Penetration* seems to present the highest differences between the tasks and *Force* the smallest ones. In the case of that most differentiated variable of *Penetration*, *Railing* has a smallest mean compared to the total average (ratio 0.78), whereas *Side Insertion* has the biggest one (ratio 1.17).

In general, the factors of **Device** and **Rendering** produced significant effects in all performance variables of *Time*, *Force*, *Penetration*, and *Effort* (all $p < 0.05$ or smaller, see Table 6.5) (\rightarrow L1.3). The factor of **Stiffness**, on the other hand, has a significant effect on *Penetration* only, while tendencies for *Force* and *Effort* are still apparent.

Time to Complete (Table 6.5(a)) \diamond Among the **Devices**, the HUG produced significantly higher *Time* means ($p < 0.01$, +22%) compared to the Sigma. Similarly, the Constraint-based **Rendering** yielded higher *Time* means ($p < 0.001$, +39%) compared to the Penalty-based method. There were no significant interactions between factors.

It is interesting to observe that although the HUG has a $5.5\times$ larger workspace (1 : 1 scaling) than the Sigma, on average, it produces (only) a $1.22\times$ larger completion

Table 6.4: Descriptive data of the objective dependent variables for the whole exercise and the tasks. Average and standard deviation values are provided. The ratios between the grand mean of each task and the total exercise are coded in red if the task values are smaller than the exercise values and blue otherwise. The statistical analysis can be found in Table 6.5.

ID	R	K	Total Exercise	Tasks					
				Frontal		Side		Railing	
(a) Time to Complete [s]									
H1	P	l	29.73 (6.80)	7.03 (2.49)	11.50 (2.19)	11.21 (3.19)			
H2	P	h	32.68 (10.29)	8.36 (3.54)	12.31 (3.07)	12.01 (5.12)			
H3	C	l	40.92 (15.86)	10.71 (8.29)	16.59 (7.11)	13.61 (5.34)			
H4	C	h	40.93 (15.30)	10.36 (6.73)	16.14 (8.33)	14.43 (7.44)			
S1	P	l	23.75 (6.40)	4.43 (1.41)	9.93 (2.98)	9.40 (3.44)			
S2	P	h	23.60 (5.28)	5.00 (1.74)	10.49 (2.60)	8.10 (2.14)			
S3	C	l	34.33 (12.66)	7.82 (7.96)	14.70 (5.53)	11.81 (6.98)			
S4	C	h	36.48 (15.74)	7.96 (5.09)	16.65 (9.37)	11.86 (6.40)			
<i>Grand M & SD</i>			32.80 (13.22)	7.71 (5.63)	13.54 (6.26)	11.55 (5.57)			
<i>Ratio</i>				0.24	0.41	0.35			
(b) Average Force on Contact [N]									
H1	P	l	3.60 (0.95)	3.33 (1.38)	3.62 (1.29)	3.77 (1.27)			
H2	P	h	4.44 (1.34)	4.05 (1.40)	4.20 (1.51)	4.79 (1.74)			
H3	C	l	5.68 (2.26)	5.27 (2.77)	5.98 (3.16)	4.71 (2.00)			
H4	C	h	5.08 (1.61)	5.09 (2.35)	5.33 (2.02)	4.39 (1.55)			
S1	P	l	1.04 (0.28)	0.91 (0.39)	1.20 (0.43)	1.02 (0.30)			
S2	P	h	1.41 (0.49)	1.45 (0.84)	1.38 (0.51)	1.44 (0.44)			
S3	C	l	2.10 (1.11)	2.13 (1.59)	2.18 (1.09)	1.99 (1.27)			
S4	C	h	2.43 (0.86)	2.60 (1.52)	2.51 (1.14)	1.98 (0.81)			
<i>Grand M & SD</i>			3.22 (2.04)	3.10 (2.24)	3.30 (2.30)	3.01 (1.94)			
<i>Ratio</i>				0.96	1.02	0.94			
(c) Average Penetration on Contact [mm]									
H1	P	l	4.60 (2.48)	4.98 (2.56)	5.09 (4.28)	3.37 (2.27)			
H2	P	h	3.04 (1.31)	3.29 (1.33)	3.57 (2.03)	2.13 (1.34)			
H3	C	l	0.94 (0.25)	0.58 (0.25)	0.81 (0.25)	1.26 (0.49)			
H4	C	h	0.69 (0.22)	0.52 (0.25)	0.65 (0.25)	0.89 (0.52)			
S1	P	l	7.57 (3.77)	8.13 (5.15)	10.12 (5.93)	5.65 (3.27)			
S2	P	h	5.66 (3.58)	7.67 (6.30)	6.00 (4.52)	4.10 (2.77)			
S3	C	l	0.82 (0.17)	0.68 (0.22)	1.00 (0.37)	0.77 (0.50)			
S4	C	h	0.73 (0.18)	0.71 (0.22)	0.85 (0.27)	0.66 (0.29)			
<i>Grand M & SD</i>			3.01 (3.24)	3.32 (4.28)	3.51 (4.43)	2.35 (2.47)			
<i>Ratio</i>				1.10	1.17	0.78			
(d) Average Effort [0: relaxation, 100: steady max. during calibration]									
H1	P	l	109.45 (35.80)	108.58 (36.98)	118.56 (43.33)	103.85 (33.40)			
H2	P	h	114.55 (37.23)	114.46 (33.59)	117.65 (35.67)	111.20 (41.03)			
H3	C	l	115.13 (38.72)	112.37 (37.94)	114.48 (40.46)	111.69 (38.72)			
H4	C	h	114.83 (34.29)	117.08 (39.45)	117.52 (40.48)	110.02 (36.30)			
S1	P	l	63.84 (27.03)	48.65 (28.18)	86.49 (24.83)	51.87 (27.39)			
S2	P	h	66.58 (27.78)	51.14 (27.56)	87.19 (30.24)	53.20 (27.12)			
S3	C	l	67.73 (24.80)	54.17 (29.52)	82.47 (31.54)	53.02 (26.96)			
S4	C	h	69.55 (28.37)	57.38 (35.10)	84.75 (31.84)	53.54 (29.81)			
<i>Grand M & SD</i>			90.21 (39.31)	82.98 (45.00)	101.14 (38.18)	81.05 (43.02)			
<i>Ratio</i>				0.92	1.12	0.90			

Table 6.5: Statistical analysis of the objective dependent variables (performance) to determine the effect of the varied factors (device, rendering, stiffness) on them. Sample size (N), degrees of freedom (df), Fisher statistic (F), p -value, Cohen's d , and the relation (with coded effect size) between treatment levels are provided. The source descriptive data can be found in Table 6.4.

	N	df	F (Fisher)	$p(> F)$	sig.	d (Cohen)	Relation
(a) Time to Complete (Total Exercise)							
Device (D: H, S)	24	1	9.81	0.00467	**	0.51	H > S
Rendering (R: P, C)	24	1	41.27	1.48e-06	***	0.88	C ≫ P
Stiffness (K: l, h)	24	1	1.21	0.282	.	0.09	h ≈ l
D:R	24	1	0.54	0.472	.	–	–
R:K	24	1	0.02	0.896	.	–	–
D:K	24	1	0.05	0.826	.	–	–
D:R:K	24	1	1.22	0.281	.	–	–
(b) Average Force on Contact (Total Exercise)							
Device (D: H, S)	24	1	257.7	5.48e-14	***	2.09	H ≫≫ S
Rendering (R: P, C)	24	1	54.47	1.66e-07	***	0.61	C > P
Stiffness (K: l, h)	24	1	3.27	0.0837	.	0.11	h ≈ l
D:R	24	1	1.75	0.199	.	–	–
R:K	24	1	9.33	0.00561	**	–	–
D:K	24	1	0.95	0.339	.	–	–
D:R:K	24	1	11.14	0.00286	**	–	–
(c) Average Penetration on Contact (Total Exercise)							
Device (D: H, S)	24	1	36	4.05e-06	***	0.43	S ≧ H
Rendering (R: P, C)	24	1	79.96	6.03e-09	***	1.87	P ≫≫ C
Stiffness (K: l, h)	24	1	37.98	2.75e-06	***	0.30	l ≧ h
D:R	24	1	39.99	1.88e-06	***	–	–
R:K	24	1	24.51	5.26e-05	***	–	–
D:K	24	1	0.087	0.771	.	–	–
D:R:K	24	1	0.60	0.448	.	–	–
(d) Average Effort (Total Exercise)							
Device (D: H, S)	24	1	76.83	8.64e-09	***	1.47	H ≫≫ S
Rendering (R: P, C)	24	1	5.35	0.03	*	0.08	C ≈ P
Stiffness (K: l, h)	24	1	3.42	0.0774	.	0.05	h ≈ l
D:R	24	1	0.04	0.836	.	–	–
R:K	24	1	3.06	0.0934	.	–	–
D:K	24	1	0.003	0.958	.	–	–
D:R:K	24	1	1.36	0.255	.	–	–

Significance codes (p): 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Effect size codes (d): 0 '≈' 0.2 '≧' 0.5 '>' 0.8 '≫' 1.5 '≫≫' ∞

Time. Regarding the differences between **Rendering** methods, these are interpreted as mentioned in Section 6.2.1.2: the Penalty-based approach tends to round sharp corners, which eases the exercises, decreasing the *Time* required to complete them.

Average Force (Table 6.5(b)) \diamond Regarding the **Devices**, the HUG produced significantly higher *Force* means ($p < 0.001$, $2.7\times$) compared to the Sigma. The Constraint-based **Rendering** yielded also higher *Force* means ($p < 0.001$, $+46\%$) compared to the Penalty-based method. A tendency for higher *Force* values ($p < 0.1$, $+7\%$) occurred for the high **Stiffness**. In this case, interactions between **Rendering:Stiffness** (R:K) and **Device:Rendering:Stiffness** (D:R:K) were significant (both $p < 0.01$). The interaction plot of R:K revealed that when using the Constraint-based algorithm, the *Force* values were (unexpectedly) slightly higher with the lower stiffness ($M_{C,l} = 3.89\text{ N} > M_{C,h} = 3.75\text{ N}$). This is not the case for the Penalty-based algorithm ($M_{P,h} = 2.92\text{ N} > M_{P,l} = 2.32\text{ N}$). In the case of the D:R:K interaction, something similar occurred: when using the HUG (and not the Sigma), the *Force* mean decreased if a high **Stiffness** was applied under the Constraint-based method. In both cases, the **Stiffness** factor seems to trigger the interaction; since the differences are relatively small and the **Stiffness** factor is not significant, the commented significant effects of **Device** and **Rendering** are considered to be valid.

With the collected data, it is difficult to fully explain such interactions and, in particular, the factor of **Stiffness** surely needs further investigation.

Average Penetration (Table 6.5(c)) \diamond The *Penetration* is significantly affected by the **Device**, the **Rendering**, and the **Stiffness** (all $p < 0.001$). Using the Sigma has the effect of increasing the overlap between the objects ($+59\%$), compared to the HUG. Employing the Penalty-based approach leads to considerably larger means ($6.53\times$) in contrast to the Constraint-based method. Additionally, lower stiffnesses also are related to higher overlap values ($+38\%$). It is worth to mention that there were interactions between **Device:Rendering** (D:R) and **Rendering:Stiffness** (R:K). The interaction plots showed that the latter is not relevant, whereas the first interaction deserves a closer look. Using the Constraint-based approach decreases the *Penetration* compared to the Penalty-based method. However, when the participants tried the Constraint-based approach, the overlap means were higher for the HUG ($M_{C, \text{HUG}} = 0.82\text{ mm} > M_{C, \text{Sigma}} = 0.77\text{ mm}$), in contrast to what happened when the Penalty-based algorithm was used: $M_{P, \text{Sigma}} = 6.62\text{ mm} > M_{P, \text{HUG}} = 3.82\text{ mm}$. This last phenomenon is expected, since the values are measured in VR units and the Sigma operates with a $1 : 5.5$ scaling. On the other hand, that reversed behavior between the devices when the Constraint-based approach is used has a size of 0.05 mm , i. e., it is so small that it could be contemplated as

negligible. Therefore, the significant effects related to the **Device** and the **Rendering** are still considered to be valid.

The most remarkable result in this section is clearly the fact that the Constraint-based algorithm causes on average only a 15% overlap error of the one produced by the Penalty-based method. The average values are under 1 mm, which is the resolution of the voxelmap structure used during the tests (see Figure 6.3) (\rightarrow L1.4).

Average Effort (Table 6.5(d)) \diamond The factor of **Device** is significant ($p < 0.001$) when it comes to the *Effort*. Interacting with the HUG leads to higher means (+70%) compared to working with the Sigma. In addition, the **Rendering** has a significant ($p < 0.05$) effect too on the *Effort*, being the values with the Constraint-based method slightly higher (+4%) than with the Penalty-based approach. The factor of **Stiffness** shows a tendency to increase the means if the high level is used, but the effect is rather small (+3%). Finally, an interaction tendency occurred between **Rendering:Stiffness**, but the interaction plot and the small effect size revealed it negligible.

Looking at the effect sizes, it is considered that the only relevant factor which affects the *Effort* is the **Device**. As mentioned, the HUG leads to higher means; this was expected, since its workspace is $5.5\times$ bigger and has a considerably larger mass ($\sim 9\times$) than the Sigma (see Section 6.2.1.2). These results are in line with the ergonomics and workload ratings presented in Section 6.2.2.1.

Exercise Perception Descriptives and Analysis Overview (Table 6.6, Table 6.7) \diamond A visual inspection of the color-coded histograms from Table 6.4 shows that, for both devices, as one goes downwards in the table (R: P \rightarrow C, K: low \rightarrow high), the mass of the answers related to the *Perceived Contact Realism* tends to move from left (1: very few realistic) to right (7: very realistic). In the case of the *Perceived Object Inter-Penetration*, it is the other way around: the mass of the answers tends to move to the right (i.e., 1: no penetration) as one descends in the rows (i.e., R: C, K: high). For both variables of *Realism* and *Penetration*, the effect of the **Rendering** is visually more tangible than the one of the **Stiffness**. The statistical analysis revealed the same effects, as discussed in the following subsections.

Perceived Realism (Table 6.7(a)) \diamond When it comes to the factor of **Device**, using the HUG leads to a significantly higher *Perception of Realism* ($p < 0.001$, +29%). The Constraint-based **Rendering** method helps also to significantly increase the means ($p < 0.001$, +65%) compared to the Penalty-based method. Surprisingly, the lower **Stiffness** treatment yields significantly higher ($p < 0.01$, +17%) *Realism* ratings than the stiffer

Table 6.6: Descriptive data of the subjective dependent variables for each of the eight exercises performed by the participants. Average and standard deviation values are provided, as well as a histogram. Both variables were coded in a seven-point Likert scale (1: very low/none, 4: moderate, 7: very high). The statistical analysis can be found in Table 6.7.

ID	R	K	Perception of Contact [1 – 7]															
			Realism							Penetration								
H1	P	low	3.54 (1.69)	2	6	5	4	3	3	1	5.04 (1.60)	2	3	3	5	6	5	
H2	P	high	4.67 (1.34)	1		3	6	7	6	1	3.50 (1.67)	4	2	6	6	3	2	1
H3	C	low	4.92 (1.18)			3	7	4	9	1	2.38 (1.35)	7	9	2	5		1	
H4	C	high	5.17 (1.24)		1	1	5	5	10	2	2.42 (1.69)	12	2	3	3	3	1	
S1	P	low	2.25 (1.29)	7	10	4	1	1	1		6.17 (1.52)		2		1	2	3	16
S2	P	high	3.50 (1.56)	2	5	7	2	5	3		4.63 (1.91)	1	4	3	1	5	6	4
S3	C	low	4.17 (1.20)		2	5	8	5	4		2.29 (1.30)	9	5	6	2	2		
S4	C	high	4.21 (1.10)		1	6	7	7	3		2.67 (1.24)	5	6	7	4	2		

treatment. Finally, there is a significant interaction between **Device:Rendering** ($p < 0.05$), but interaction plots revealed it irrelevant.

Looking at these results, it can be concluded that using the HUG with the Constraint-based method adjusted with a lower force stiffness than the maximum possible will bring the most realistic contacts (\rightarrow L1.5). This last point is a key insight of the first user study: softer contacts in combination with the Constraint-based approach seem to be most realistic. As commented by some participants, increasing the contact stiffness leads probably to situations in which the peg can get more easily jammed in the hole due to a misalignment of the axes, perceived as unrealistic. In addition, it seems that the correct visual display (minimum penetration, achieved with C) plays an important role, apparently dominant over the stiffness of the contact. Therefore, a more fluid (lower K) but visually correct (C) option is the preferred.

Perceived Penetration (Table 6.7(b)) \diamond The interactions present between almost all factors rendered their effect on the subjective *Perception of Penetration* rather inconclusive. The analysis of the *Average Penetration* (Section 6.2.2.2) was not helpful to explain this variable. This is in line with the significant but small correlation between both values discussed in Section 6.2.2.2 (see Table 6.8).

Overall, the **Rendering** seems to be a determinant factor; the Penalty-based approach yields significantly higher means of *Perception of Penetration* ($p < 0.01$, +38%) compared to the Constraint-based method. However, the **Stiffness** significantly interacts with the **Rendering**: higher stiffness values cause the Constraint-based algorithm to produce higher rating means, in contrast to low stiffness values: $M_{C,h} = 4.69 > M_{C,l} = 2.38$, $M_{P,h} = 4.08 < M_{P,l} = 5.60$.

Table 6.7: Statistical analysis of the subjective dependent variables to determine the effect of the varied factors (device, rendering, stiffness) on them. Sample size (N), degrees of freedom (df), Fisher statistic (F), p -value, Cliff's δ , and the relation (with coded effect size) between treatment levels are provided. The source descriptive data can be found in Table 6.6.

	N	df	F (Fisher)	$p(> F)$	sig.	δ (Cliff)	Relation
(a) Perceived Contact Realism (Total Exercise)							
Device (D: H, S)	24	1	18.34	2.78e-04	***	0.31	H > S
Rendering (R: P, C)	24	1	51.76	2.52e-07	***	0.63	C \gg P
Stiffness (K: l, h)	24	1	8.92	0.00661	**	0.18	l \gtrsim h
D:R	24	1	6.10	0.0214	*	–	–
R:K	24	1	0.12	0.734		–	–
D:K	24	1	1.11	0.302		–	–
D:R:K	24	1	0.23	0.639		–	–
(b) Perceived Contact Penetration (Total Exercise)							
Device (D: H, S)	24	1	3.86	0.0616	.	0.08	H \approx S
Rendering (R: P, C)	24	1	82.93	4.33e-09	***	0.41	P > C
Stiffness (K: l, h)	24	1	1.89	0.183		0.11	h \approx l
D:R	24	1	3.45	0.0761	.	–	–
R:K	24	1	66.51	3.08e-08	***	–	–
D:K	24	1	14.75	8.36e-04	***	–	–
D:R:K	24	1	5.43	0.0289	*	–	–
Significance codes (p): 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1							
Effect size codes (δ): 0 ' \approx ' 0.15 ' \gtrsim ' 0.3 '>' 0.5 ' \gg ' 0.7 ' \ggg ' 1							

One possible interpretation to that phenomenon could be connected to the observation introduced in Section 6.2.2.2: despite the fact that the Constraint-based algorithm causes more realistic contacts in general, using higher stiffness values makes the exercise more difficult, since the peg can get jammed more easily; in those situations, it is to expect the user to observe more intensely the details of the contacts in order to figure out how to solve the assembly as fast as possible. Thus, more penetration errors are noticed and probably weighted in a non-linear fashion. In any case, it can be concluded that the subjective perception does not have to correspond to the equivalent objective indicator, at least for small penetrations.

Exercise Correlations (Table 6.8) \diamond Except for *Effort* and *Perception of Penetration*, all other variables of *Time*, *Penetration*, and *Perception of Realism* are significantly ($p < 0.001$) and at least moderately ($|r|, |\rho| > 0.3$) correlated among each other. The strongest value is the one between *Penetration* and *Perception of Realism*, which have a significantly strong and negative correlation ($p < 0.001$, $\rho = -0.54$). Among the variables with a smaller number of proven relationships, the *Effort* has a significant strong and positive correlation with the *Force* ($p < 0.001$, $\rho = 0.57$), and the *Perception*

Table 6.8: Pearson (r) and Spearman (ρ) correlations and their respective significance values (p) related to the independent variables collected after each exercise with varied conditions. Both objective and subjective variables are represented.

	Average Force		Average Penetration		Average Effort		Perceived Realism		Perceived Penetration	
	r	p	r	p	r	p	ρ	p	ρ	p
Average Time	0.32	5.24e-06 (***)	-0.39	2.46e-08 (***)	0.06	0.407 ()	0.34	1.96e-06 (***)	-0.19	0.00831 (**)
Average Force		-	-0.39	2.46e-08 (***)	0.57	<2.2e-16 (***)	0.31	1.15e-05 (***)	-0.08	0.291 ()
Average Penetration				-	-0.12	0.0848 (.)	-0.54	8.91e-16 (***)	0.27	1.55e-04 (***)
Average Effort						-	0.20	0.00603 (**)	0.02	0.745 ()
Perceived Realism								-	-0.32	4.66e-06 (***)

Significance codes (p): 0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation codes (ρ): 0 'none' 0.1 'small' 0.3 'moderate' 0.5 'strong' 1.0

of *Penetration* has a significant moderate and negative correlation with the *Perception of Realism* ($p < 0.001$, $\rho = 0.57$).

In general, no unexpected relevant results related to the objective variables were observed (*Time*, *Force*, *Penetration*, and *Penetration*). Regarding the muscular *Effort*, one could presume that higher virtual *Forces* might be strongly linked to higher *Effort* values that react to them.

As far as the subjective variables are concerned, the *Penetration* value seems to be the objective variable that best explains the *Perception of Realism*, even more than the *Time*, i. e., less overlap increases the perceived realism probably more strongly than easier or faster exercises (\rightarrow L1.6). However, although *Perception of Penetration* and *Realism* have a significant moderate and negative correlation ($p < 0.001$, $\rho = -0.32$), the correlation between the actual *Penetration* and the *Perception of Penetration* is significant but, surprisingly, relatively smaller ($p < 0.001$, $\rho = 0.27$). This phenomenon is also commented in Section 6.2.2.2.

6.2.3 Study 1: Summary of Lessons Learned and Discussion

In this section, the impact of the key take-home messages is discussed. The insights of the results can be summarized as follows:

- L1.1 Compared to the Sigma, the HUG presents considerably higher ($2\times$) values of *Restriction*, *Inertia*, and *Physical Workload* (Section 6.2.2.1).
- L1.2 The Sigma has higher (+22%) *Usability* ratings than the HUG (Section 6.2.2.1).
- L1.3 The choices of the **Device** (HUG or Sigma) and the **Rendering** (Penalty-based or Constraint-based) significantly affect all objective performance values of *Exercise Completion Time*, *Contact Force*, *Object Inter-Penetration*, and *Muscular Effort* (Section 6.2.2.2).
 - In comparison to the Sigma, the HUG leads to higher *Time* (+22%), *Force* ($2.7\times$), and *Effort* (+70%) means, whereas it is the Sigma which increases the average *Penetration* values (+59%).
 - In comparison to the Penalty method, the Constraint-based approach yielded higher *Time* (+39%), *Force* (+46%), and *Effort* (+4%, probably negligible) means; however, the Penalty-based algorithm considerably increased the average values of the *Penetration* ($6.53\times$).
- L1.4 The penetration error caused by the Constraint-based approach seems to be close to the resolution of the underlying signed distance field and it is only 15% of the error produced by the Penalty-based method (Section 6.2.2.2).

- L1.5 The (significantly) most realistic contacts occurred interacting with the HUG (+29%) using the Constraint-based algorithm (+65%) adjusted with a lower force stiffness (+17%) (Section 6.2.2.2).
- L1.6 The *Penetration* or actual overlap between the objects is a strong explanatory variable of the *Perceived Contact Realism*: less penetration is related to more realism (Section 6.2.2.2).

The author is not aware of works similar to the study of this section which report both performance and perception results in such an extent. As exposed in Section 6.1.1, most of the related literature deals with comparing real and virtual interactions with haptic feedback, without varying the rendering algorithms. Among them, few works have compared force values in addition to completion time values, and the author has not found any objective assessment of the muscular effort. Therefore, it is difficult to point out straightforward connections, but there are two relevant insights worth mentioning.

First, although both devices considerably differ in mass (9x) and workspace or scaling (5.5x), the effect of that difference has a smaller magnitude in both subjective (L1.1, L1.2) and objective (L1.3) results. That suggests that, even when the distance between device paradigms (i. e., size, inertia, force capabilities, etc.) is large, the user performance values will be closer to each other, being a rough performance prediction sounder. Along these lines, some works already indicated that there are no significant differences in the performances obtained using haptic devices of the same size or paradigm [HBA⁺06]. Therefore, the presented results could probably be extrapolated to other systems, yet more investigation is required to support that statement. In that sense, this points to an interesting future work direction.

Second, it seems that higher stiffness values do not always improve performance or perception indicators, or at least there might be apparently a threshold [OG04]. Once the minimum necessary stiffness is achieved, contact realism seems to be improved with haptic rendering approaches that provide *coherent visual cues* (i. e., minimum overlap, attained by god object or constraint-based methods) (L1.4, L1.6) and through the use of *light haptic interfaces with bigger workspaces* than desktop-size (L1.5). The values that quantify this qualitative statement have been provided, being also probably extensible to other systems. The remaining question is now, however, which levels of performance and perception in virtual environments are necessary for an optimum skill transfer of tasks to be performed in real environments.

6.3 Study 2: Analysis of the Differences between Real and Virtual Manipulations

This section presents the second evaluation study introduced at the beginning of this chapter. The study tries to isolate the effect of the (i) *synthetic visual feedback*, (ii) the *haptic device* and the (iii) *haptic rendering* on user performance and perception. The influence of (iv) *real acoustic signals* is also analyzed. For that purpose, again, $N = 24$ naïve participants performed in a within-design user study the same three well-defined assembly tasks introduced in the previous Section 6.2. The subjects worked with real models and, gradually, the real subsystems were replaced by their synthetic equivalents, ending up with completely virtual environments. The haptic device tested in the completely virtual environment was the HUG [HHK⁺11] and the *god object* haptic rendering algorithm presented in Chapter 4.

The main results of this work characterize those subsystems. In addition to comparing task completion times and subjective perception ratings, as commonly done in the literature, also force values, muscular effort, and reaction times were collated. This gives rise to a more complete model for virtual assembly performance and perception.

This section is organized as follows: Section 6.3.1 describes the experimental setup and design, including the tasks performed by the participants, the varied factors, and the followed procedure. The results and discussion of the experiments are presented in Section 6.3.2 and, finally, Section 6.3.3 concludes with the most important insights.

6.3.1 Experimental Design and Implementation

Figure 6.5 shows the concept of the study and the used setup. The tasks carried out by the participants are outlined, as well as the varied feedback factors when working with the real and the virtual models. This section elaborates all those points and describes the experimental procedure.

6.3.1.1 Synthetic Haptic Feedback

As explained, the haptic *device* used in the study is again the HUG [HHK⁺11] (see Figure A.1). It is worth to mention that only one arm was used during the user studies, activated by the user with a foot-pedal. Furthermore, although several interfaces (such as data gloves, joysticks, or grippers) can be magnetically coupled to the device, users interacted with a simple handle in order to reduce additional influences and simplify the analysis.

Special attention is given to the used constraint-based haptic *rendering* method from Chapter 4; not only because it is a novel approach, but also because synthetically

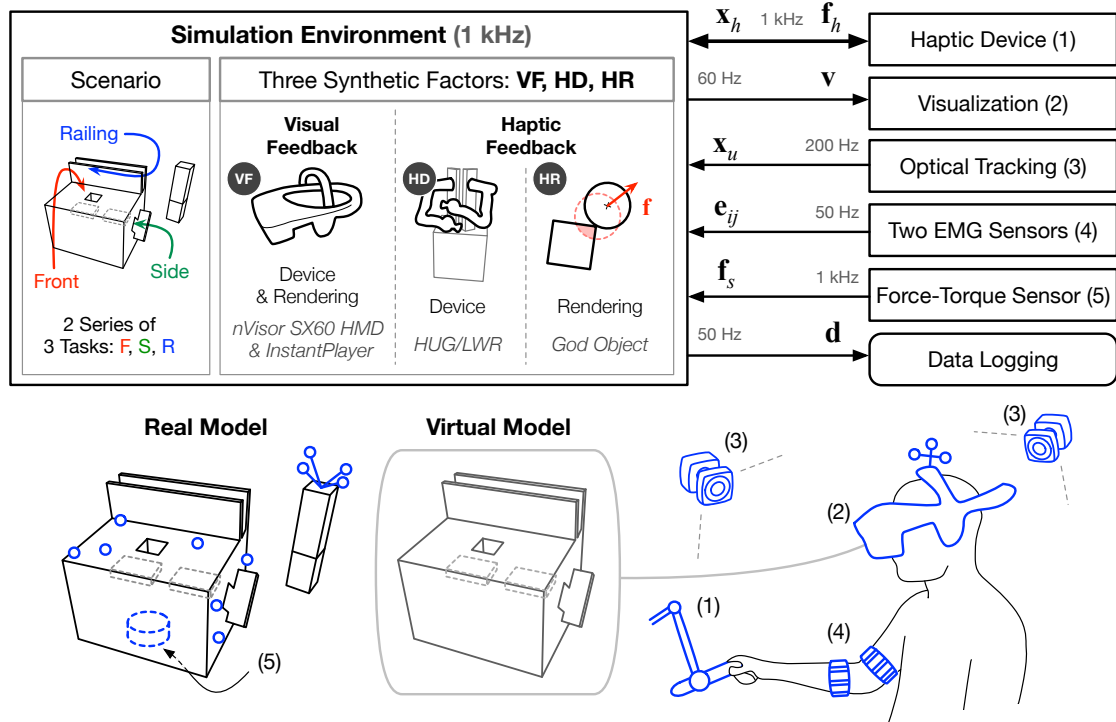


Figure 6.5: General diagram of the setup in the second user evaluation. The $N = 24$ participants performed with real and virtual models several exercises composed of two series of three tasks: frontal insertion, side insertion, and railing. In each of the exercises, the factors of *Visual Feedback* (VF), *Haptic Device* (HD), and *Haptic Rendering* (HR) were varied in two levels: *synthetic* and *real* (i. e., no virtual feedback provided in the *real* treatment). That variation led to five *Degrees of Virtualization* (D, see Figure 6.7), from purely real to purely virtual. The order of all factor levels was systematically permuted. In the exercises where the real models were used, the trajectories were optically tracked and the contact forces registered using a JR3 force-torque sensor. Additionally, the muscular EMG signals of the participants were measured with two Myo armbands during all exercises.

rendered force signals have rarely been contrasted with real contacts while removing the influence of the used mechanical interface. The contact stiffness was reduced to 3000 N/m (instead of 3700 N/m), as suggested by the results from the first study; note that the used stiffness value is also the result of further adjustments performed during the piloting phase with 3 participants.

6.3.1.2 Tested Scenario: Tasks and Exercises

Figure 6.6 shows the real and virtual models used in this study, their dimensions, and the tasks performed with them. The virtual models were the same as in the first study (see Figure 6.3); in the same line, the elementary tasks did not change either, being: (i)

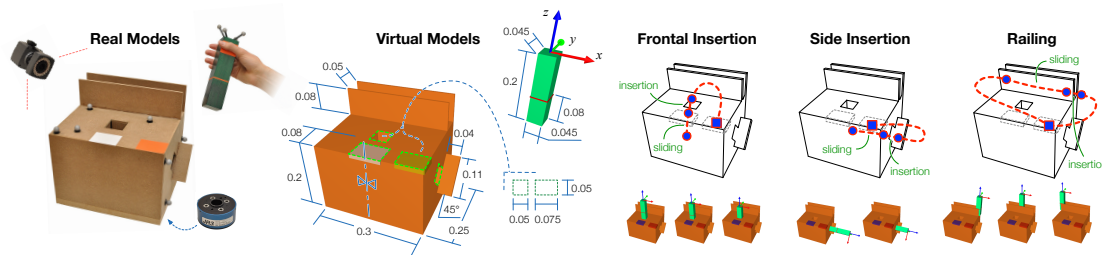


Figure 6.6: Real and virtual models and tasks in the scenario of the second user study (dimensions in meters). Each exercise consisted of two series of a combination of three peg-in-hole tasks: frontal assembly, side assembly, and railing, as in the first study. The grasping regions of the peg were standardized by marking them with red stickers. All holes had a clearance of 5 mm and a depth of 8 cm, marked with a red ring on the peg. In contrast to the first study, the participants had to hit a white middle-box for starting and finishing the exercise; additionally, a yellow corner-box had to be hit between the tasks. The boxes had a different color (blue and red) during each exercise in order to indicate that the process was being recorded. Pauses were not allowed between the tasks.

frontal insertion, (*i*) side insertion, and (*i*) railing. Therefore, the reader is encouraged to re-visit Section 6.2.1.1, if information on properties and motivation related to the tasks is desired.

However, the exercise is defined differently in this second study; it consists of *two* series of a sequence of the three different peg-in-hole tasks, and *no pauses* are allowed during the exercise. Therefore, it is more complex and longer than in Study 1, necessary for the secondary audio task explained in Section 6.3.1.4. The used assembly model had also two reference boxes: a gray one set in the middle for starting and finishing the whole exercise (it turned blue during the exercise); and a yellow one located on the right corner for starting and finishing the tasks (it turned red during the exercise). Right after starting the exercise (collision on the middle box), the users had to carry out the 2×3 tasks in a row (and *without* pauses). The order of the tasks was systematically permuted for each user session (and maintained constraint during it) and the participants had to count aloud the tasks to ensure they did not miss any of them. As in the first study, the participants were instructed to perform the tasks (*first*) with the lowest contact forces possible and (*second*) as fast as possible.

6.3.1.3 Apparatus and Varied Factors

As mentioned at the beginning of this section, Figure 6.5 illustrates the major components of the *apparatus* and the *varied factors*. The main goal of the study was to compare virtual assembly manipulations with real ones in order to identify the effect of each system component. To that end, interfaces and synthetic scenarios were introduced in different degrees, while the exercises to be performed by the subjects (Section 6.3.1.2)

remained constant. The main focus lays on the evaluation of *haptic feedback*, but *visual feedback* was also altered in the process.

In the following, first, all the components of the setup are described, and then, the implementation of the variation is discussed; note that these components are similar to the ones in Study 1 (Section 6.2.1.2):

- (1) The haptic device used in the study is the HUG [HHK⁺11], described in Section A. The used stiffness was $k = 3000 \text{ N/m}$, without added damping. As mentioned, these values were obtained after the first study and during dedicated pilot tests, trying to reproduce the real contacts with the highest fidelity possible.
- (2) The visualization was powered again by the InstantPlayer* engine from Fraunhofer IGD and displayed with an nVisor SX60[†] from NVIS. The standard eye separation of 63 mm was adjusted for each participant for an optimum 3D vision.
- (3) A Vicon Bonita[‡] optical tracking system was used to track the movements of the real models and the head movements. On account of the tracking, the participants could move the camera view intuitively in the scenario.
- (4) Two Myo armbands[§] from Thalmic Labs were used for recording the electro- myographical (EMG) signals of the upperarm and forearm, following the same procedure as in the first study (see Section 6.2.1.2).
- (5) The real assembly model was mounted on a JR3[¶] force-torque sensor with which physical collisions were measured.

Varied Factors: Degrees of Virtualization

Three factors were defined, each one with the two levels of treatment *real* (R) and *synthetic* (S):

VF *Visual Feedback*: the *real* treatment of this factor means the user saw with the bare eyes the scenario, whereas with the *synthetic* treatment virtual images were displayed on the HMD.

HD *Haptic Device*: the user had no haptic device coupled to the hand with the *real* treatment of this factor, i. e., manipulations were done with the bare hand holding

*<http://www.instantreality.org>

†<http://www.nvisinc.com>

‡<https://www.vicon.com/products/camera-systems/bonita>

§<https://www.thalmic.com>

¶<http://www.jr3.com>

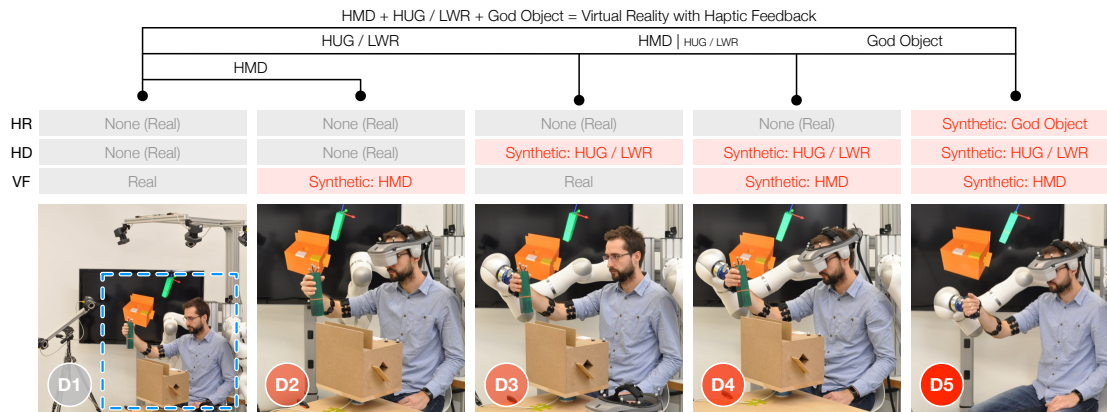


Figure 6.7: Degrees of virtualization, starting from D1 (purely real) to D5 (purely virtual). The whole setup with the tracking cameras and the HUG is also displayed in D1. For each degree, the binary values (*real* or *synthetic*) of their three main factors are specified: *Visual Feedback* (VF), *Haptic Device* (HD), and *Haptic Rendering* (HR). Additionally, the comparisons between degrees are reported; e. g., the collation between D1–D2 isolates the effect of the head-mounted display (HMD).

the physical peg; for the *synthetic* treatment, the HUG was connected to the hand of the user.

HR *Haptic Rendering*: when the *real* treatment of this factor was applied, the users felt the collisions between the physical models; conversely, during the *synthetic* treatment, the virtual forces computed by the haptic rendering (see Section 6.3.1.1) algorithm were displayed to the user via the HUG. That implies that if the HR is *synthetic*, the HD needed to be *synthetic*, too.

As the reader can deduce, VF comprises both the *rendering* and the *device* parts of the *synthetic visual feedback* illustrated in Figure 6.1, whereas the *synthetic haptic feedback* is divided into the other two factors HD (*device*) and HR (*rendering*). This allows for a better analysis of the haptic modality, as sought in the study.

As illustrated in Figure 6.7, the combination of the different treatments of the three factors leads to five *degrees of virtualization*:

- D1 This *degree* can be considered as purely real. The participants performed the exercise in the physical reality without any *synthetic* feedback systems (i. e., VF = R, HD = R, HF = R); however, as in all other degrees, trajectories, collision forces and the muscular effort were measured (see Figure 6.5).
- D2 The participants performed the exercise with the physical models but saw virtual images through the HMD (i. e., VF = S, HD = R, HF = R).

- D3 The participants performed the exercise with the physical models observing them with their bare eyes, but had to carry the HUG arm (i. e., VF = R, HD = S, HF = R). The robot arm was gravity compensated and did not display any virtual forces. Nevertheless, the users could feel the inertia of the system.
- D4 The participants performed the exercise with the physical models but carrying the HUG and watching virtual images through the HMD. This *degree* is the superposition of the previous two (i. e., VF = S, HD = S, HF = R).
- D5 In this *degree* all factors were synthetic, thus, the participants worked with the virtual models instead of the real ones. In contrast to D1, this *degree* is purely virtual.

By comparing the dependent values obtained under each degree, the effects of the synthetic factors over the real level can be determined (see Figure 6.7 above).

6.3.1.4 Secondary Task and Auditory Privation

The effects of two more variations were tested and evaluated also: (i) the influence of a *secondary task* that had to be performed in parallel and (ii) the impact of *auditory privation* during the exercises.

The first consisted in a short (~ 200 ms) and loud horn or “beep” that was systematically played in cycles during the exercises, without perceivable periodicity. The users had to press a pedal with their left foot as soon as they heard the sound and the reaction times were measured with millisecond accuracy. Each *secondary task* cycle lasted 8 s and the sound was played in a random instant during the period of 2–8 s. In addition to the regular degree exercises explained in the previous section, the participants had to carry out all degrees with this *secondary task* too, and, during them, they were told that both assembly exercises and *secondary tasks* performed in parallel had the same priority. Reaction times to audio signals conceived as *secondary tasks* have been used in the literature as objective measures of workload [Her15] or presence [BPW14].

Second, participants had to carry out the exercises wearing ear plugs and headphones with active noise-cancellation; additionally, white noise was played on the headphones in order to maximally obstruct their auditory perception. Exercises with this *auditory privation* were carried out only during the degrees with *real* visual feedback (i. e., D1 and D3), because no full obstruction with headphones was possible in practice if the users wore the HMD. Although it has been suggested that there is no specific dominance in tri-sensory (vision-audio-haptic) tasks [HR09], haptic stiffness perception has been shown to be biased by real [DBS97] and synthetic [AC06] sound cues. Hence, it is relevant to

analyze to what extent *real* audio cues (or their absence) might affect the user interaction with the presented setup.

6.3.1.5 Sample, Procedure, and Collected Data

A total of $N = 24$ participants with marginal or no virtual assembly experience were recruited. Eight of them participated in the first study approximately two months prior to this second one. The statistically standard participant was male (3 female), right-handed (3 left-handed), and $M_{\text{age}} = 28.79$ years old ($Md_{\text{age}} = 27$). All participants had a university degree or (6 of them) were undergraduate students. All subjects read and signed a consent form and they were not paid for their participation.

As in Study 1, all participants tried all conditions (within-design) in one session that roughly lasted 1.5 h (including explanations and pauses). The procedure followed with each participant can be summarized in the following steps:

- #1 Standardized **instructions** were given to the participants in form of slides. The purpose of the study was explained, the procedure, and a video of the exercise. Additionally, the participants learned how to use the devices (~ 15 min).
- #2 The **consent form** and a demographic and experience questionnaire were filled out by the participants (~ 5 min).
- #3 The participants carried out 2 **test trials** with the real models until they felt comfortable with the order of the tasks of the exercise (~ 2 min).
- #4 For each of the five *degrees* (~ 12 min $\times 5$):
 - #4.1 The EMG **calibration** was performed.
 - #4.2 A first **learning test trial** was performed.
 - #4.3 The participants performed the **regular exercise** and the **exercise with the secondary task**. For D1 and D3, the **exercise with the auditory privation** was also carried out. The order of all three exercises was systematically permuted and after each one a **perception questionnaire** was filled out.

The participants performed altogether 5 (regular, D1–D5) + 5 (secondary task, D1–D5) + 2 (auditory privation, D1 and D3) = 12 exercises. As for the perception questionnaire, it comprised items related to the *perception of realism* (seven-point Likert scale) and *workload* (1–20 scale, reported in five quantiles in the results for the sake of clarity: Q1 – Q5).

Regarding the *perception of realism*, the participants had to evaluate

- how realistic the **overall** experience was,
- how realistic the **contacts** felt,
- and how realistic the **manipulation** or movement of the objects in the scene was.

As for the *workload*, the **physical** and the **mental** effort had to be rated, defined and explained to the participants as in the first study. Also in this second study, shorter questionnaires were favored rather than longer standard ones (see Section 6.2.1.3).

In addition to the subjective ratings, the following objective values were recorded at 50 Hz: trajectory along time, real and virtual contact forces and torques, and muscular effort signals; for exercises with the *secondary task*, reaction times were also measured with 1 kHz accuracy, in addition to the missed signals.

6.3.2 Results and Discussion

Results are presented and discussed in two main sections: Section 6.3.2.1 deals with the regular exercises, whereas Section 6.3.2.2 covers the exercises with the *secondary task* and the *auditory privation*. Special focus is put on the set of regular exercises. The most important insights are enumerated and discussed in Section 6.3.3; as in the first study, all items in it are linked throughout the text (e. g., →L2.1).

The following video shows the scenario, the exercises, and the interaction with the different virtualization degrees:

Multimodal Evaluation of the Differences between Real and Virtual Assemblies @ Vimeo

<https://vimeo.com/230945754>

The statistical analysis was performed in R using standard packages and the `ez` library [Law11]. For the objective continuous variables parametric tests were used. All objective variables were normally distributed in most of the cases; it is trusted in the robustness of the parametric methods for the exceptions. Additionally, sphericity corrections were applied when necessary (values given in tables). On the other hand, non-parametric tests (less powerful, but more appropriate for discrete values) were used for the subjective data. In all cases, the main significance level was set to $\alpha = 0.05$, but this value was adjusted with the Bonferroni factor b for pairwise comparisons (i. e., $\alpha' = \alpha/b = 0.05/10 = 0.005$). For the sake of brevity and clarity, it is avoided introducing too many statistical values in the text and the reader is encouraged to look up in the provided tables, as done in the first study.

6.3.2.1 Regular Exercises

Figure 6.8 shows exemplary plots of objective indicators related to the same subject performing D1 and D5. Additionally, Table 6.9 gathers the most relevant descriptive data of the objective dependent variables, whereas Table 6.10 summarizes the results of their statistical analysis. In Table 6.9, the task segment values of the objective variables are shown, but only the total exercise values are considered for the statistical analysis. As far as the subjective variables are concerned, Table 6.11 outlines the descriptive data and Table 6.12 their statistical analysis.

A repeated measures one-way ANOVA was performed on the **Degree of Virtualization** (D) for total exercise values of both objective and subjective dependent variables and it turned out to be significant in all cases (all $ps < 0.001$, ***). Therefore, the following subsections will deal with the post hoc pairwise comparisons between the degrees to discuss the effect of the HMD* (VF: D1-D2, D3-D4), the HUG (HD: D1-D3), the haptic rendering method (HR: D4-D5), and the whole system (VR: D1-D5).

Objective Variables: Performance (Table 6.9 & Table 6.10) \diamond After a general visual inspection, the bar diagrams (Table 6.9) show that the *Time to Complete* progressively increases with the **Degree** (increment ratios varying between $\sim 20-50\%$). In the case of the *Average Force*, the *synthetic* values (D5) are considerably higher (up to $\sim 6\times$) than the *real* ones (D1-D4). And, finally, the *Average Effort* substantially increases ($\sim 60\%$) when the haptic device comes into play (D3 onwards).

All pairwise comparisons related to the *Time* values are significant (all $ps < 1e-04$, ***). The HMD increased the means by $+37\%$ when used alone (D1-D2) and by $+21\%$ when used in combination with the HUG (D3-D4). The HUG, in addition, produced $2\times$ larger values (D1-D3), and the haptic rendering method led to $+43\%$ longer exercises. Overall, the participants needed $3.5\times$ the *Time* of the purely real **Degree** (D1) to complete the exercises in the purely virtual one (D5) (\rightarrow L2.1). Interestingly, the effects of the *synthetic* treatments do not seem to add up linearly. Additionally, the haptic feedback, and in particular, the haptic device HUG seems to be the most important bottleneck when it comes to the *Time* performance.

Post hoc pairwise comparisons of *Force* values revealed distinct difference levels between **Degrees**. The HMD did not significantly affect the *Forces* exerted by the participants when used alone (D1-D2, $p > 0.005$, ns), but it contributed to $+22\%$ larger means when used in combination with the HUG (D3-D4, $p < 0.005$, *). The HUG alone significantly contributed to $+38\%$ bigger values (D1-D3, $p < 0.001$, **). Moreover, *Synthetic*

*When the head-mounted display (HMD) is mentioned, the whole *synthetic* visual feedback is meant, including the graphical rendering.

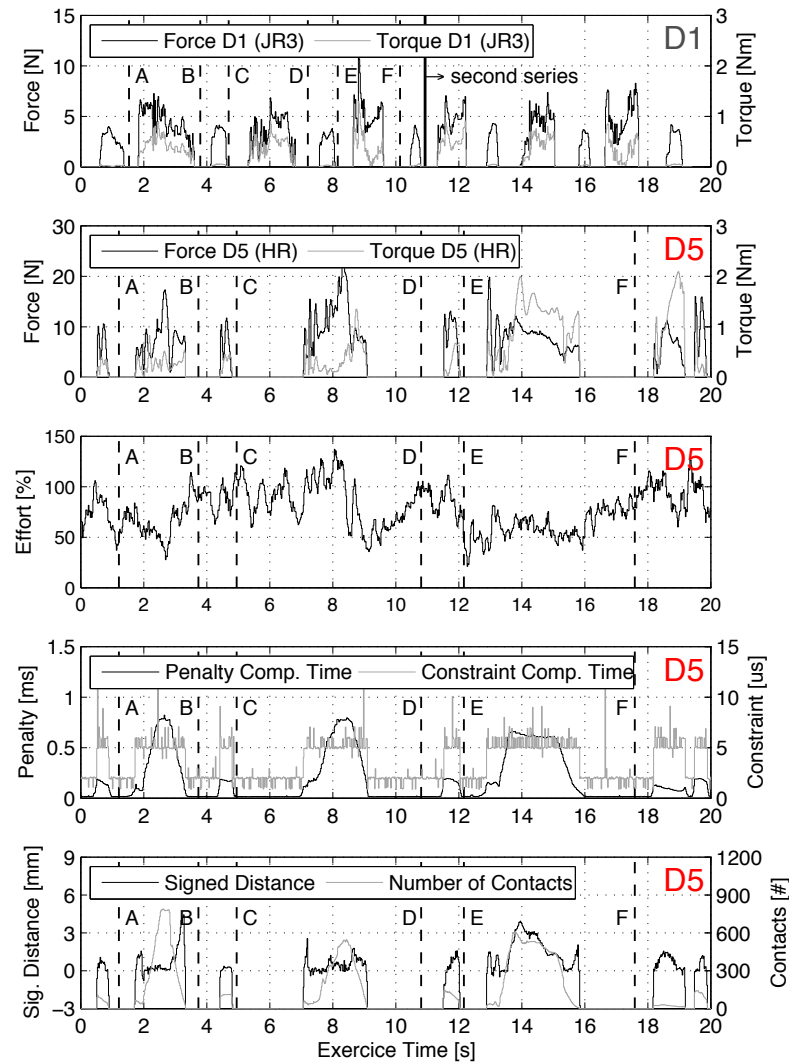

















Figure 6.8: Exemplary values recorded during regular exercises of D1 and D5 of the same subject. Only the first half of the exercise is shown in the following order: frontal insertion (A→B), side insertion (C→D), and railing (E→F). The segments corresponding to the in-between collisions with the box in the corner were ignored for the evaluation. The models from Figure 6.3 were used for D5. An effort value of 100% corresponds to the maximum steady signal recorded during the EMG calibration (see Section 6.3.1.3). The signed distance is the penetration between the objects when positive and the distance between the objects when negative (the positive part is zoomed, since it represents the inter-penetration of the two virtual objects).

Table 6.9: Descriptive data of the objective dependent variables for the whole exercise and the tasks. The five *Degrees* (D) are decomposed in the three varied factors *Visual Feedback* (VF), *Haptic Device* (HD), and *Haptic Rendering* (HR). The used *real* (R) or *synthetic* (S) treatment of the factors is also specified (see Figure 6.7). Average and standard deviation values are provided for each *Degree* (D), as well as bar diagrams of the total values. The statistical analysis can be found in Table 6.10.

D	VF	HD	HR		Total Exercise	Tasks						
						Frontal		Side		Railing		
(a) Time to Complete [s]												
D1	R	R	R		15.76 (3.50)	4.26 (1.12)	5.60 (1.30)	5.89 (1.39)				
D2	S	R	R		21.60 (4.75)	5.81 (1.56)	7.81 (1.78)	7.98 (1.67)				
D3	R	S	R		31.74 (6.43)	7.26 (1.53)	12.59 (2.53)	11.89 (2.85)				
D4	S	S	R		38.54 (8.51)	9.11 (1.84)	15.02 (3.04)	14.41 (4.34)				
D5	S	S	S		55.27 (16.02)	12.20 (4.46)	22.12 (9.69)	20.95 (8.55)				
<i>Grand M & SD</i>					32.58 (16.46)	7.73 (3.65)	12.63 (7.49)	12.23 (6.95)				
(b) Average Force [N]												
D1	R	R	R		1.06 (0.53)	0.87 (0.34)	0.91 (0.27)	1.26 (0.94)				
D2	S	R	R		1.08 (0.59)	0.95 (0.53)	0.98 (0.35)	1.25 (0.94)				
D3	R	S	R		1.45 (0.74)	1.13 (0.56)	1.18 (0.52)	1.79 (1.26)				
D4	S	S	R		1.76 (0.62)	1.72 (0.78)	1.65 (0.72)	1.83 (0.98)				
D5	S	S	S		6.70 (2.95)	6.01 (3.72)	6.87 (2.82)	6.07 (3.18)				
<i>Grand M & SD</i>					2.41 (2.59)	2.14 (2.61)	2.32 (2.65)	2.44 (2.48)				
(c) Average Effort [0: relaxation, 100: steady max. during calibration]												
D1	R	R	R		56.88 (14.13)	53.24 (14.91)	61.59 (16.26)	55.27 (14.68)				
D2	S	R	R		54.03 (16.83)	51.00 (18.32)	55.46 (16.82)	54.90 (17.93)				
D3	R	S	R		90.13 (37.23)	86.91 (40.46)	95.39 (42.03)	86.62 (31.38)				
D4	S	S	R		91.48 (47.02)	86.63 (42.89)	95.47 (49.80)	90.77 (47.87)				
D5	S	S	S		117.75 (39.63)	117.48 (41.51)	118.14 (35.48)	115.69 (47.80)				
<i>Grand M & SD</i>					82.05 (40.82)	79.05 (41.57)	85.21 (41.50)	80.65 (41.44)				

contact rendering led to $3.79\times$ higher *Force* values (D4-D5, $p < 1e-04$, ***). When it comes to the objective indicators, it is in the *Forces* where the purely virtual **Degree** (D5) most differentiates from the purely real one (D1), with $6.35\times$ bigger means for the *synthetic* condition (D1-D5, $p < 1e-04$, ***) (\rightarrow L2.2). Clearly, these virtual *Forces* are too high compared to the real ones. Decreasing the value of the stiffness could appear to be a way of solving that, however, as mentioned in Section 6.3.1.3, the used stiffness (lower than the maximum possible) was selected to produce the contacts with the highest fidelity possible. Due to the observational evidence and the insights from the previous Study 1 (Section 6.2), the reason for such high *Forces* could be that the haptic rendering algorithm causes interactions that are more jamming than real ones; as a result, the user applies higher *Forces* and requires also longer to accomplish the exercises. This point should be further investigated.

Table 6.10: Statistical analysis of the objective dependent variables (performance) to determine the effect of the *Degree* (One-Way ANOVA with Fisher tests) and the factors of *Visual Feedback* (VF), *Haptic Device* (HD), and *Haptic Rendering* (HR). Pairwise comparisons were done with T-Tests after a Bonferroni adjustment of $b = 10$. Provided values: Sphericity correction (Mauchly's W and Greenhouse-Geisser ϵ), sample size (N), degrees of freedom (df), S statistic (S is Fisher $F(df, N - 1)$ for *Degree* and T-Test $T(df)$ for pairwise comparisons), p -value, Cohen's d , and the relation (with coded effect size) between the *degrees*. The source descriptive data can be found in Table 6.9.

	W	$p(>W)$	ϵ	b	N	df	S	$p(>S)$	sig.	d	Relation
(a) Time to Complete (Total Exercise)											
Degree	4.86e-02	1.88e-10	0.41	(1)	24	4	112.57	1.40e-15	***	-	-
D1-D2 (VF)	-	-	-	10	24	23	-8.11	3.38e-08	***	1.40	D2 \gg D1
D1-D3 (HD)	-	-	-	10	24	23	-15.71	8.68e-14	***	3.08	D3 \ggg D1
D3-D4 (VF)	-	-	-	10	24	23	-5.67	8.97e-06	***	0.90	D4 \gg D3
D4-D5 (HR)	-	-	-	10	24	23	-5.94	4.74e-06	***	1.30	D5 \gg D4
D1-D5 (VR)	-	-	-	10	24	23	-12.98	4.58e-12	***	3.41	D5 \ggg D1
(b) Average Force on Contact (Total Exercise)											
Degree	8.45e-04	8.34e-28	0.28	(1)	24	4	79.03	1.32e-09	***	-	-
D1-D2 (VF)	-	-	-	10	24	23	-0.30	0.77		0.05	D2 \approx D1
D1-D3 (HD)	-	-	-	10	24	23	-4.18	3.54e-04	**	0.62	D3 $>$ D1
D3-D4 (VF)	-	-	-	10	24	23	-3.26	3.41e-03	*	0.46	D4 \gtr D3
D4-D5 (HR)	-	-	-	10	24	23	-8.43	1.72e-08	***	2.31	D5 \ggg D4
D1-D5 (VR)	-	-	-	10	24	23	-9.53	1.87e-09	***	2.66	D5 \ggg D1
(c) Average Effort (Total Exercise)											
Degree	0.145	4.52e-06	0.70	(1)	24	4	24.90	2.10e-10	***	-	-
D1-D2 (VF)	-	-	-	10	24	23	-1.33	0.20		0.18	D2 \approx D1
D1-D3 (HD)	-	-	-	10	24	23	-4.89	6.19e-05	***	1.18	D3 \gg D1
D3-D4 (VF)	-	-	-	10	24	23	-0.16	0.87		0.03	D4 \approx D3
D4-D5 (HR)	-	-	-	10	24	23	-2.71	1.25e-02	.	0.60	D5 $>$ D4
D1-D5 (VR)	-	-	-	10	24	23	-9.37	2.56e-09	***	2.05	D5 \ggg D1

Significance codes (p): 0 '***' 0.001/ b '**' 0.01/ b '*' 0.05/ b '.' 0.1/ b ',' 1
Effect size codes (d): 0 ' \approx ' 0.2 ' \gtr ' 0.5 ' $>$ ' 0.8 ' \gg ' 1.5 ' \ggg ' ∞

No significant effect of the HMD or the haptic rendering was found on the *Effort* values (all $ps > 0.005$, ns). However, the HUG led to significantly +58% bigger values (D1-D3, $p < 1e-04$, ***). In the same line, muscular *Effort* signals were on average 2× larger in the purely virtual **Degree** compared to the purely real (D1-D5, $p < 1e-04$, ***) (\rightarrow L2.3). To the best of the author's knowledge, no other comparison studies of this sort have used EMG measurements as an indicator. The found significant differences proof the validity of it.

Subjective Variables: Perception (Table 6.11 & Table 6.12) \diamond A first look at the histograms (Table 6.11) conveys the fact that the *Overall*, *Contact*, and the *Manipulation Realism* ratings progressively shift from maximum values (7) in D1 to values gravitating

Table 6.11: Descriptive data of the subjective dependent variables related to the (total) exercises. The five Degrees (D) are decomposed in the three varied factors *Visual Feedback* (VF), *Haptic Device* (HD), and *Haptic Rendering* (HR). The used *real* (R) or *synthetic* (S) treatment of the factors is also specified (see Figure 6.7). Average and standard deviation values are provided for each Degree (D), as well as a histogram. The statistical analysis can be found in Table 6.12.

D	Perception of Realism [1: very low - 4: moderate - 7: very high]							Perception of Workload [1: very low - 20: very high]																				
	Overall	1	2	3	4	5	6	7	Overall	1	2	3	4	5	6	7	Physical	Q1	Q2	Q3	Q4	Q5	Mental	Q1	Q2	Q3	Q4	Q5
D1	7 (0)								7 (0)								3.67 (2.62)	18	4	2			4.63 (3.17)	15	5	4		
D2	6 (0.66)				5	14	5	24	6.29 (0.81)								4.63 (2.76)	14	8	2			6.17 (2.90)	9	8	7		
D3	5.5 (1.02)			1	3	6	11	3	4.75 (1.36)								8.13 (4.16)	7	7	5	4	1	6.33 (3.16)	7	11	5	1	
D4	5 (1.18)	1			5	10	7	1	4.88 (1.48)	1	1	2	3	8	7	2	8.67 (4.02)	5	9	5	5		7.92 (3.97)	4	12	3	5	
D5	4.46 (0.93)	1	3	5	14	1			4.17 (1.13)	2	5	6	9	2			10.5 (3.97)	2	6	7	8	1	10.04 (3.56)	11	5	8		

around 4 – 5 in D5. In contrast, the *Physical* and *Mental Workload* seem to increase with the **Degree** from quantiles Q1–Q2 (ratings 1 – 8 of a maximum of 20) to quantiles Q2–Q3 (ratings 5 – 12).

The three ratings of *Realism Perception* decreased roughly 1 point when the HMD was used (alone) in interaction with the real models (D1-D2, all $ps < 0.005$, or smaller levels). However, no significant differences could be found due to the HMD when it was used in combination with the HUG (D3-D4, all $ps > 0.005$). The use of the HUG yielded ratings which were 1 – 2 points lower for the *Overall* and the *Manipulation Realism* (D1-D3, both $ps < 1e-04$, ***); the *Contact Realism* decreased roughly 1 point ($p < 0.001$, **). Regarding the haptic rendering method, a significant reduction of 1 point was found only in the case of the *Contact Realism* (D4-D5, $p < 0.001$, **) (\rightarrow L2.4). Comprehensibly, it seems that the HMD affects primarily the *Overall Perception of Realism*, the haptic rendering method the *Contact Realism*, and the HUG the *Manipulation Realism*; moreover, the HUG and the haptic rendering method are the *synthetic* treatments with the highest impact.

Both *Physical* and *Mental Workload* average ratings of the purely virtual condition (D5) were respectively $2\times$ ($p < 1e-04$, ***) and $3\times$ ($p < 0.001$, **) higher than the purely real ones (D1). Nevertheless, the *synthetic* treatments affected differently each dimension. In the case of the HMD, a significant increase of +1 average point could be determined for the *Physical Workload* when used alone, while the same significant increment occurred when it was used in combination with the HUG for the *Mental Workload* (both $ps < 0.005$, *). The HUG led to $2.21\times$ higher *Physical Workload* ratings ($p < 1e-04$, ***), but no significant effect could be found on the *Mental Workload*. Regarding the haptic rendering algorithm, slight significances were found for both the *Physical* ($p < 0.005$, *) and the *Mental Workload* ($p < 0.05$, .); the *synthetic* treatment resulted in ratings +2 points higher on average. Finally, and interestingly, the *Average Effort* and the *Physical Workload* had a significant and rather strong (Spearman) correlation of $\rho = 0.44$ ($p = 3.94e-07$, ***), computed across all conditions. In summary, it seems that the HMD and the haptic rendering increased the *Perception of Workload* in a similar slight but perceptible fashion, whereas the HUG had a bigger impact, especially on *Physical Workload*, which was doubled (\rightarrow L2.5).

6.3.2.2 Secondary Task and Auditory Privation

In this section, the effects of the *secondary task* and the *auditory privation* are reported, trying to bridge them with the *Workload* and the *Realism Perception*, respectively.

Table 6.12: Statistical analysis of the subjective dependent variables to determine the effect of the *Degree* (One-Way Friedman ANOVA) and the factors of *Visual Feedback* (VF), *Haptic Device* (HD), and *Haptic Rendering* (HR). Pairwise comparisons were done with T-Tests after a Bonferroni adjustment of $b = 10$. Provided values: Sample size (N), degrees of freedom (df), S statistic (S is Friedman χ^2 (df) for *Degree* and Wilcoxon V for pairwise comparisons), p -value, Cliff's δ , and the relation (with coded effect size) between the *degrees*. The source descriptive data can be found in Table 6.11.

	N	df	b	S	$p(>S)$	sig.	δ	Relation
(a) Overall Realism (Total Exercise)								
Degree	24	4	(1)	68.58	4.53e-14	***	–	–
D1–D2 (VF)	24	23	10	190	6.33e-05	***	0.79	D1 \ggg D2
D1–D3 (HD)	24	23	10	231	4.55e-05	***	0.87	D1 \ggg D3
D3–D4 (VF)	24	23	10	98.5	0.107		0.26	D3 \gtrsim D4
D4–D5 (HR)	24	23	10	86.5	2.81e-02		0.32	D4 $>$ D5
D1–D5 (VR)	24	23	10	300	1.19e-05	***	0.97	D1 \ggg D5
(b) Contact Realism (Total Exercise)								
Degree	24	4	–	66.19	1.45e-13	***	–	–
D1–D2 (VF)	24	23	10	55	1.90e-03	*	0.42	D1 $>$ D2
D1–D3 (HD)	24	23	10	120	4.56e-04	**	0.63	D1 \gg D3
D3–D4 (VF)	24	23	10	56	0.842		0.01	D3 \approx D4
D4–D5 (HR)	24	23	10	226	1.07e-04	**	0.73	D4 \ggg D5
D1–D5 (VR)	24	23	10	300	1.57e-05	***	1	D1 \ggg D5
(c) Manipulation Realism (Total Exercise)								
Degree	24	4	(1)	66.53	1.22e-13	***	–	–
D1–D2 (VF)	24	23	10	91	9.09e-04	**	0.54	D1 \gg D2
D1–D3 (HD)	24	23	10	253	3.61e-05	***	0.92	D1 \ggg D3
D3–D4 (VF)	24	23	10	59	0.40		0.08	D3 \approx D4
D4–D5 (HR)	24	23	10	125.5	1.89e-02		0.35	D4 $>$ D5
D1–D5 (VR)	24	23	10	300	1.63e-05	***	1	D1 \ggg D5
(d) Physical Workload (Total Exercise)								
Degree	24	4	(1)	72.66	6.21e-15	***	–	–
D1–D2 (VF)	24	23	10	4.5	1.42e-03	*	0.24	D2 \gtrsim D1
D1–D3 (HD)	24	23	10	0	4.09e-05	***	0.70	D3 \ggg D1
D3–D4 (VF)	24	23	10	45	0.137		0.07	D4 \approx D3
D4–D5 (HR)	24	23	10	14	1.70e-03	*	0.26	D5 \gtrsim D4
D1–D5 (VR)	24	23	10	0	1.89e-05	***	0.85	D5 \ggg D1
(e) Mental Workload (Total Exercise)								
Degree	24	4	(1)	46.63	1.82e-09	***	–	–
D1–D2 (VF)	24	23	10	29	1.42e-02		0.32	D2 \gtrsim D1
D1–D3 (HD)	24	23	10	65	2.67e-02		0.33	D3 $>$ D1
D3–D4 (VF)	24	23	10	36	3.19e-03	*	0.24	D4 \gtrsim D3
D4–D5 (HR)	24	23	10	27	6.18e-03	.	0.32	D5 $>$ D4
D1–D5 (VR)	24	23	10	10.5	1.10e-04	**	0.75	D5 \ggg D1

Significance codes (p):

0 '***' 0.001/ b '**' 0.01/ b '*' 0.05/ b '.' 0.1/ b ' ' 1

Effect size codes (δ):

0 ' \approx ' 0.15 ' \gtrsim ' 0.3 ' $>$ ' 0.5 ' \gg ' 0.7 ' \ggg ' 1

Secondary Task \diamond For analyzing the differences in *Reaction Times* during the secondary task, a one-way ANOVA on the **Degree** was performed. Mauchly's test indicated that the assumption of sphericity had been violated ($W = \chi^2(4) = 0.198$, $p = 7.18\text{e-}05$, ***), and therefore, a Greenhouse-Geisser correction was used with $\epsilon = 0.53$. There was no significant effect of the **Degree** on the *Reaction Time*, $F(2.12, 12.19) = 1.60$, $p = 0.211$ (ns). The grand mean (and standard deviation) of the *Reaction Times* across all **Degrees** was $M = 658.86$ (185.08) ms. Similarly, no significant (Pearson) correlations were found between the *Reaction Times* and any of the subjective variables.

The number of missed horn signals was also analyzed in the set of exercises with the secondary task (i. e., how many times the pedal was not pressed after a "beep"). Although a Friedman test revealed the **Degree** non significant ($\chi^2(4) = 6.70$, $p = 0.153$, ns), two relevant subjective variables showed significant yet small Spearman correlations with the number of missed "beeps" (\rightarrow L2.6):

- *Overall Realism*, $\rho = -0.25$ ($p = 5.40\text{e-}03$, **);
- *Mental Workload*, $\rho = 0.25$ ($p = 6.57\text{e-}03$, **).

Auditory Privation \diamond A two-way ANOVA was performed on all three objective variables to analyze the effect of the factors of *Degree* (D1 & D3) and *auditory privation* (yes & no). The *Degree* turned out to be always significant (statistics are omitted since they are analogous to the ones in Table 6.10), but no effect could be found for the *auditory privation*:

- *Time*, $F(1, 23) = 0.087$, $p = 0.77$ (ns);
- *Force*, $F(1, 23) = 0.02$, $p = 0.89$ (ns);
- *Effort*, $F(1, 23) = 0.478$, $p = 0.50$ (ns).

Similarly, Friedman tests were performed with the subjective variables. Significant differences were found only for the *Perception of Realism* ratings associated with D1; however, the mean of $M = 7$ (0) points related to the regular exercises barely shifted half a point when participants wore headphones:

- *Overall*, 6.54 (0.59), $\chi^2(1) = 10$, $p = 1.57\text{e-}03$ (**);
- *Contact*, 6.71 (0.55), $\chi^2(1) = 6$, $p = 1.43\text{e-}02$ (*);
- *Manipulation*, 6.79 (0.51), $\chi^2(1) = 4$, $p = 4.55\text{e-}02$ (*).

In other words, although not hearing the contacts or the haptic device slightly decreased the *Perception of Realism*, it apparently had no effect of the performance of the users (\rightarrow L2.7).

6.3.3 Study 2: Summary of Lessons Learned and Discussion

In this section, the most significant results reported so far are highlighted and discussed. First of all, a synopsis of the analyzed data formulated as brief take-home messages is provided in the following:

- L2.1 All *synthetic* treatments increased the *Time* means compared to the *real* treatments: HMD (VF) up to +37%, HUG (HD) $2\times$, and haptic rendering (HR) +37%. Overall, the purely virtual condition required on average $3.51\times$ the *Time* of the purely real one (Section 6.3.2.1).
- L2.2 *Synthetic Force* values generated by the haptic rendering algorithm were on average $3.79\times$ bigger than the *real* ones; additionally, participants were provided in the purely virtual condition with $6.35\times$ the *Force* magnitudes of the purely real one (Section 6.3.2.1).
- L2.3 In the purely virtual condition $2\times$ larger muscular *Effort* signals than in the purely real were measured; this increment was mainly due to the HUG, which led to an increase of +58% when added to the interaction (Section 6.3.2.1).
- L2.4 Using each of the *synthetic* treatments (HMD, HUG, haptic rendering) decreased the ratings of *Overall*, *Contact*, and *Manipulation Realism* roughly 1 point in a 7-point scale from "very low" (1) to "very high" (7); these three realism dimensions were perceived as above "moderate" (4–5 points) during the purely virtual *Degree* (D5) (Section 6.3.2.1).
- L2.5 The *Perception of Physical and Mental Workload* shifted roughly from a 20% level in the purely real condition (D1) to a 50% level in the purely virtual one (D5); the HMD and the haptic rendering contributed approximately to increments of 5–10% points each, whereas the HUG added 20% points, especially in the case of the *Physical Workload* (Section 6.3.2.1).
- L2.6 During the exercises with the secondary task, no significant effect of the **Degree** on the *Reaction Times* could be found; however, the number missed horn signals significantly correlated with the *Overall Realism* ($\rho = -0.25$) and the *Mental Workload* ($\rho = 0.25$) (Section 6.3.2.2).

L2.7 No changes in the performance could be determined when contact sounds were silenced; however, almost half of the participants rated the *Overall Realism* 1 point lower (out of 7) in the purely real condition (D1) (Section 6.3.2.2).

As expected, subjects showed better performance during purely real exercises (D1) than in purely virtual ones (D5); along these lines, objective indicators show that, compared to *Degree* D1, *Degree* D5 led to $3.51\times$ larger *Time* values (L2.1), $6.35\times$ larger *Force* values (L2.2), and $2\times$ larger *Effort* values (L2.3). However, the contribution of the work lies rather on the granular quantification of disparities produced by each synthetic feedback (subsystem) on the analyzed objective and subjective indicators.

In general, haptic feedback seems to be the bottleneck. Additionally, as far as synthetic haptic feedback is concerned, two noteworthy facts that account for most of the variance between real and virtual manipulations can be observed: (i) the HUG had the highest impact on the *Time* (L2.1), *Effort* (L2.3), and *Physical Workload* (L2.4) indicators, and (ii) the *synthetic* forces were considerably higher than the *real* ones (L2.2).

It is worth to mention that, overall, *Time* ratios between real and virtual manipulations (L2.1) were close to the ones reported in some works cited in Section 6.1.1: [UNT⁺02], [LRD⁺07], and [GBMCL⁺14]. In contrast to this present study, those related works employed desktop-size haptic interfaces. Study 1 in previous section Section 6.2 concluded that around 20% shorter *Time* values could be achieved using the Sigma.7 desktop device, although a scaling of $5.5\times$ was necessary to cover the whole virtual workspace. Along these lines, smaller *Times* should be expected if smaller and lighter devices are used. Additionally, the notable increase of the *Average Muscular Effort* (L2.3) and *Physical Workload* (L2.5) due to the HUG seems to be related to that increase of *Time* values. As noted in Study 1 (Section 6.2), the HUG seems to increase contact realism perception, but in detriment of ergonomy ratings. The device is a remarkable research platform for testing large unscaled movements and a broad stiffness spectrum; nonetheless, the results seem shape as the optimum device one with a similarly large workspace, but which sacrifices stiffness capabilities for lower inertia values.

The unexpectedly large virtual forces (L2.2) reveal an improvement direction for the haptic rendering algorithm. However, it is worth to mention that the commanded virtual forces (D5) analyzed here differ from the actual exerted forces, since the influence of the robot dynamics and the hand-device coupling are not contemplated; yet, for practical reasons, it is assumed both are similar. In contrast to the results presented in the current work, Unger et al. [UNT⁺02] found no significant differences between real and virtual forces in peg-in-hole scenarios using a desktop-size magnetic levitation haptic interface and the Coriolis engine [Bar93] for force rendering. After observing recorded exercises of D5 and conducting short interviews with the participants, two major interpretations that

help explain that discrepancy between force magnitudes were gathered: (i) as opposed to other penalty-based haptic rendering algorithms, the used constraint-based approach tends to sharpen all edges, sometimes even reducing the natural slipperiness of small corner contacts, and (ii) the perceived ratio between torques and forces transmitted (i. e., not commanded) through the HUG might be smaller than in the real world. Therefore, users could have been provided with realistic but sometimes misleading collision cues, especially during insertion tasks in which accurate forces are necessary to comprehend the configuration error and perform fast corrections. Due to that resulting lower fidelity, participants might have literally *worked hard and not smart*, as opposed to how the saying goes. Study 1 from previous Section 6.2 also supports these points; nevertheless, further investigation is required to substantiate the interpretation. Along these lines, and following the debate in the VR community discussing to what extent immersion components should mimic reality [BM07], further research questions worth exploring arise: Which is the minimum fidelity necessary to obtain the maximum skill transfer? How do fidelity improvements affect performance?

Regarding the *Realism* ratings, each *synthetic* treatment seems to contribute to steady, slow, and similar decreases (L2.4). In this sense, the *Perception of Realism* is probably a more robust and less sensible variable than the objective indicators.

As far as the series of exercises beyond the regular one are concerned, more modest effects were found; yet, this reflects the effectiveness and influence of the methods and modalities tested in them, which is also a finding. The impact and explanatory power of the secondary task (L2.6) are lower than the ones suggested in the literature [Her15], [BPW14]. Thus, reactions to parallel tasks could not be as explanatory when studying assembly manipulations similar to the ones tested in this work. Similarly, although the privation of auditory cues slightly decreased the *Overall Perception Realism*, no effects were found in the performance indicators (L2.7); however, this is in line with the findings from Gupta et al. [GSW97]. Sounds related to harder materials have been shown to bias the perception of stiffness towards higher ratings [DBS97], [AC06], yet real audio cues seem not to be as dominant for the analyzed variables in the tested scenarios. Therefore, synthetic sound could probably have a secondary priority compared to haptic feedback in virtual assembly manipulations.

6.4 Summary, Conclusions, and Perspectives

This chapter presented the design, implementation, and results of two user studies which evaluate the haptic rendering methods and setups presented in previous Chapter 3, Chapter 4, and Chapter 5. The same scenario was used in both of them, consisting of well-defined and motivated peg-in-hole assemblies. Study 1 is a first step for comparing haptic

rendering algorithms and deriving the optimum configuration. On the other hand, Study 2 researches into the interplay between that optimum haptic rendering configuration and other synthetic feedback modalities, with the goal of detecting differences and bottlenecks between real and virtual manipulations. In the following, the conclusions of each study are summarized and the future work directions outlined.

Study 1: Comparison of Haptic Rendering Methods

Many haptic rendering algorithms that compute forces based on different physical principles have been presented in the last two decades. Yet, the relationship between those algorithms that take root in different fundamentals, the employed haptic interfaces, and the adjusted stiffness is not a widely researched topic. Along these lines, a within-design user study was conducted in which $N = 24$ participants performed assembly tasks with varied rendering (penalty-based vs constraint-based), device (HUG vs Sigma.7), and stiffness (low vs high) factors. To the best of the author's knowledge, this work is the first user study which compares the effects of two haptic rendering paradigms on the user performance with varied device and stiffness factors.

As summarized in Section 6.2.3, the tested constraint-based god object heuristic (Chapter 4) with a relaxed stiffness seems to be the best choice, since it decreases the object-interpenetration considerably and optimizes the perception of contact realism. When it comes to the haptic devices, the HUG led to a higher workload and a lower ergonomics, but it significantly helped to improve the realism of the contacts. Similar configurations probably lead to comparable results, as some works have suggested, but this needs further investigation before that leap can be done confidently.

Study 2: Comparison of Real and Virtual Manipulations

Although virtual manipulation with haptic feedback has been proven to be a useful tool in many fields, their fidelity is still noticeably moderate when compared to purely real interactions; this might hinder the practical use of virtual setups in training and planning applications. Literature suggests that satisfactory perceptions and performances are the result of the interplay between different feedback modalities and their constituent blocks. Along these lines, a user evaluation was performed in which $N = 24$ participants performed well-motivated assembly exercises in five different conditions, ranging gradually from purely real to purely virtual, and systematically introducing different feedback devices (i. e., haptic device or head-mounted display) or virtual feedback signals (i. e., haptic feedback). All participants tried all conditions in a different order. This study is the continuation of the previous one (Section 6.2) in which the effects of different haptic devices and haptic rendering algorithms on user performance were analyzed.

On the other hand, this current study sheds light on the understanding of the difference between the real and virtual manipulations by analyzing the effects of each system component. Moreover, altogether *visual*, *haptic*, and *audio* feedback modalities were considered, leading to a multi-modal perspective to interpret the formation of the virtual contact percept. While previous works have mainly analyzed task completion time performances, here, more than three objective (*Time*, *Force*, and *Effort*) and five subjective (related to *Realism* and *Workload*) dimensions for each tested condition are provided. To the best of the author's knowledge, this is the first multimodal and system-wise study covering so many variables. Additionally, the results are easily replicable and transferable thanks to the used scenario, consisting of a large workspace allowing unscaled movements and well-motivated but abstract exercises.

Section 6.3.3 synthesizes and discusses the most important insights, and their relationship to the literature. Independently of the systems involved, virtual manipulations inevitably lead to worse performance and subjective results compared to real ones; nonetheless quantifying them properly is an imperative step. In general, the haptic modality seems to be the bottleneck for performance: HUG, the used haptic device, affects the *Completion Time* and *Muscular Effort* indicators the most; in the same line, synthetically rendered forces differ from real ones significantly, although the effect of that needs to be studied more deeply. On the other hand, the decrease of *Realism Perception* caused by any synthetic device or rendering is similar and sound seems not to be as relevant.

The preceding Study 1 (Section 6.2) suggested that a haptic device with a large upper-body workspace but moderate stiffness capacity in favor of a lighter structure could significantly improve both performance and perception indicators. That conclusion opposes to the mainstream desktop-size systems and requires an improvement from the HUG itself. In addition to showing that the effects of the HUG on performance and subjective variables are the largest, the current study thoroughly quantifies those effects and contextualizes them in relation to other multimodal factors. That helps to gain a more holistic understanding.

Short term future work will deal with investigating and enhancing the two main bottlenecks detected in this work: (*i*) the effect of the HUG on the user and (*ii*) the discrepancy between physical and computed contact forces. Additionally, future work will try to formalize result mappings between setups of similar and different features performing further comparison studies. The ultimate goal is to study and improve the skill learning and transfer through virtual environments, focusing on practical real world scenarios. In particular, special focus is set on interactive virtual assembly [SHH⁺16] and maintenance [SHH⁺15] simulations, such as the ones presented in Chapter 5.

Chapter 7

Epilogue

This thesis analyzes interactive **virtual reality simulations with kinesthetic haptic feedback**. All aspects of the system are considered, focusing on *collision and force computation*. The two core technical contributions can be summarized as follows:

- A unified algorithm for *collision, proximity and penalty force computation* has been presented. A key factor for the algorithm are its data structures: signed distance fields and multiresolution point-sphere trees, for which fast and robust generation approaches are provided. At runtime, arbitrarily complex rigid objects can be handled at an update rate faster than 1 kHz, which makes the algorithm suitable for haptic and robotic applications.
- A *constraint-based force rendering method* which works on top of penalty-based algorithms is introduced. It is designed as a heuristic god object approach and tackles the major issues characteristic for penalty-based algorithms (usually computationally expensive); these successfully solved issues are: the tunneling-effect and object overlaps, which are handled with the approximation of the constrained motion, and soft contacts. Additionally, static, kinetic, and dynamic friction are modeled. In contrast to similar approaches, the algorithm is easy to implement and performs faster than 20 kHz.

Furthermore, these basic methods have been employed in the context of two further major contributions:

- They have been successfully *integrated into several applications*: (i) a complete virtual car assembly platform in which multibody simulations are handled, (ii) the

widely known physics engine Bullet [Cou03], which otherwise could not handle complex rigid geometries at haptic rates, and (iii) simulation and robotic applications that require realtime collision and force computation for complex rigid bodies.

- They have been *evaluated in two user studies* that shed light on (i) how haptic rendering methods should be optimally parametrized and (ii) which are the major factors that lead to the current performance and perception challenges in virtual manipulations, taking as reference the real ones.

Altogether, the work covers the presentation of new haptic rendering methods, their application to realistic virtual and robotic simulations, and their evaluation with user studies from which basic and practical insights are distilled. In the following, each of the contributions is discussed, pointing out to future directions.

Collision and Proximity Computation (Chapter 3)

The presented solution for distance and potential or penalty force computation works with complex rigid bodies in haptic/robotic realtime even with high model resolutions; this brings the contribution to a small selected group of state-of-the-art techniques, which are, in general, not available. The method is based on the principles of the Voxelmap-Pointshell (VPS) algorithm [MPT99], but it re-defines its data structures and their runtime handling. In contrast to previous approaches, signed distance fields with three levels of accuracy can be used, smoothing signal values, and point-sphere trees are employed for faster collision region detection; these hierarchical representations are built bottom-up to exploit local information and maintain sampling uniformity, and minimally enclosing spheres are computed to increase packing efficiency. The used breadth-first traverse improves the distance or force output progressively, making possible the delivery of approximative computations when high loads or low geometrical variation in contact regions is expected.

The data structures fulfill a key and significant requirement: they encode pre-computations that are quickly accessible at runtime. Future work will mainly deal with extending their capabilities, such as by storing of additional geometrical or physical information, and enabling their actualization at runtime, e.g., with streamed point clouds or deformations. The automatized and fast GPGPU generation of the data structures would also improve the workflow for the user considerably.

Constrained Motion and Force Rendering (Chapter 4)

The presented constrained motion computation algorithm and the force rendering model derived from it provide a robust heuristic that solves object-overlap and stiffness limita-

tions that appear in haptic simulations. Furthermore, the method is unique due to its computational speed and error-bounded output, and it can be applied on top of probably any penalty-based algorithm. Thanks to these practical features, the approach is readily able to improve many existing haptic interactions.

The formal or theoretical analysis provided in the thesis is the basis for a comparison framework that benchmarks the algorithm with other traditional but computationally more expensive methods. Additionally, as far as virtual interactions are considered, the extension to multibody constrained motion computation is the next logical step.

Virtual Reality and Robotics Applications (Chapter 5)

The properties of the presented data structures and algorithms make them suitable not only for haptics, but also for robotic task perception, collision awareness, or realtime grasp planing, to mention some of the use cases featured in this work. In particular, the introduced virtual car assembly platform is a good example of how the methods fit in a complex system in which all components must run synchronized and calibrated. In addition to the system integration work, topics related to virtual manipulations have been researched for the framework, such as multibody scenarios, bi-manual interaction, elbow interaction, navigation, or user collaboration. Such a platform can be used to train and verify assemblies minimizing eventually the need of physical mockups, which dramatically decreases the duration and cost associated to the design process.

The maturity and versatility of the collision and force computation methods motivates putting available the developed engine and related engineering works. The integration to the physics engine Bullet is a first step in that direction; a first and important step, since, as shown in the experiments, the presented methods outperform state-of-the-art approaches with up to $130\times$ speedups when using complex models. Thus, haptic simulations become available for developers through the widespread interfaces of Bullet.

Insights from User Studies (Chapter 6)

The presented two user studies (with $N = 24$ participants each) evaluate the contributed methods and provide valuable insights; these are expected to be generalizable, upon the replication of similar results. Both studies used well-motivated exercises and conditions, which were systematically randomized in a within-design. The thorough statistical analysis of the large number of variables is unprecedented for haptic simulations.

Altogether, the data seems to favor the constraint-based haptic rendering approach which simulates the proxy object on the surface, minimizing visual overlap; additionally, reducing the applied stiffness does not result in a decreased sensation of contact realism, but it improves user performance. The effects of all multimodal devices and synthetic

rendering methods on user performance and perception have been tabulated. The haptic feedback system seems to be the largest factor for the gap between real and virtual manipulations; in particular, haptic devices with small mass and bigger workspace in sacrifice of stiffness would be lead to better interactions.

Along these lines, the next logical steps consist in translating these insights into technological implementations and, ultimately, evaluating the skill transfer in more concrete and realistic scenarios.

Appendix A

Used Haptic Devices

Although the contributions of this thesis consist in research topics very different than the design of haptic devices, understanding how this key piece of hardware is conceived and controlled is essential. It is basically through these interfaces that the human operator is able to haptically comprehend virtually rendered or remote environments; thus, any artifacts due to inadequate design, use, or control parametrization might be perceived as incorrect renderings, masking some of the contributions reported in this work, and in detriment of fidelity.

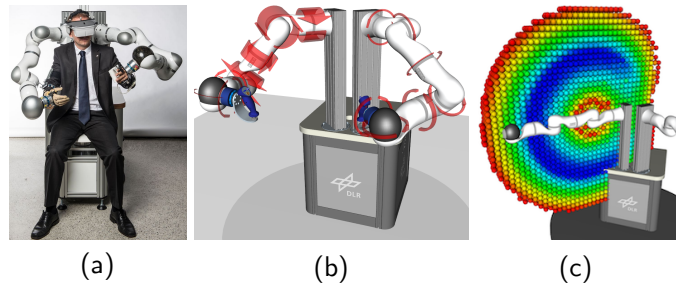
Along these lines, Section 2.2.5 already introduces the general properties and requirements of haptic devices, and notes on control issues and approaches. Additionally, Figure 2.2 from that section illustrates and classifies the five haptic devices successfully integrated into the systems presented in the current work: Falcon (Novint), Sigma.7 (Force Dimension), Virtuose6D (Haption), the Wearable Haptic Interface WHI (TU Berlin and DLR), and the HUG (DLR). A performance comparison study of these five interfaces using the same virtual scenario as in Chapter 6 was carried out and reported in [Sch15].

This appendix describes in detail the HUG and the Sigma.7 devices, since these are the main interfaces integrated in the virtual assembly platform from Section 5.2, and above all, they were tested in the user studies from Chapter 6.

A.1 The HUG

The main haptic device used in the work is the HUG [HHK⁺11], shown in Figure A.1. It consists of two torque-controlled DLR / KUKA Light-Weight Robot (LWR) arms

Figure A.1: (a) The HUG bimanual haptic device, composed of two DLR/KUKA Light-Weight Robots (LWR) [HHK⁺11], reaching $2 \times$ seven active DoF. (b) Recorded torque values in joints displayed as red rotating arrows. (c) Reachability map of the LWR; its overlap with the human reachability map was optimized for obtaining the configuration of the HUG.



[ASHO⁺07]* able of providing with six-DoF force feedback. Electronics and redundant sensors integrated in the seven revolute joints of each LWR are controlled at different frequencies, being 1 kHz the minimum update rate guaranteed. On the other hand, an additional force-torque sensor at the end-effector enables a feed-forward compensation that reduces up to a 33% the inertia introduced by the the dynamic mass of each robot, which is $m_{LWR} = 14$ kg.

This bimanual device is characterized by its large workspace that covers the whole upper body (maximum arm span of 0.94 m) and the high forces that it can display (peak values of 150 N). In particular, the device configuration was optimized by analyzing reachability maps [ZHHH10] so that its workspace maximally covers the human upper-body workspace. In the same line, thanks to the the redundant kinematic of seven-DoF, the robot elbows can be optimally configured to react compliantly to external forces and optimize their position with respect to the position of the user's elbows, avoiding singularities. All these properties make of the HUG a haptic interface suited for a great variety of common and unscaled hand manipulations.

In addition to the interaction with virtual environments, the device has often been used also in teleoperation applications with humanoids [KWA⁺09]. Despite of the successful applications carried out with it, the author is not aware of any user evaluations in which the impact of the HUG or the LWR as an input device on the user performance is analyzed. Hence, the user studies reported in Chapter 6 contribute with some new insights in this regard.

It is worth mentioning that only one arm was used during those user studies, activated by the user with a foot-pedal. Furthermore, although several interfaces (such as, data gloves, joysticks, or grippers) can be magnetically coupled to the device (as explained in Section 5.2), during the user studies, participants interacted with a simple handle in order to reduce additional influences and simplify the analysis.

*<http://www.kuka-lbr-iiwa.com>

Figure A.2: The Sigma.7 haptic device, joint development of Force Dimension and the DLR (German Aerospace Center) [THH⁺11]. The desktop-sized interface consists of parallel mechanism and a grasping unit attached to a wrist, reaching seven active DoF. This highly precise device was conceived for telerobotic applications. The picture is courtesy of Force Dimension, Switzerland.



A.2 The Sigma.7

The Sigma.7 [THH⁺11]* is a joint development of Force Dimension and the DLR (German Aerospace Center), shown in Figure A.2. The desktop-sized haptic interface consists of three main components: (i) a parallel mechanism providing three DoF translations and forces (up to 20 N), (ii) a wrist attached to the translational base for other three DoF rotations and torques (up to 0.4 Nm), and (iii) a one DoF grasping unit (with maximum pinch forces of 8 N). All three components are inherently decoupled; that results, for instance, in full rotational dexterity possible in every translational pose.

The interface is able to operate at a refresh rate of 8 kHz (a frequency of 4 kHz was used in this work) and its workspace is contained in a sphere of 0.12 m diameter. Its dedicated motors were optimized with respect to friction and torque smoothness at low rotational speeds. Additionally, although it has a rather small dynamic mass of $m_{\text{Sigma}} \simeq 1.55$ kg (measured in experiments), its force controller further targets decreasing inertia and friction effects.

All these features make of the Sigma.7 a transparent and precise desktop interface which utterly matches the requirements of the main application for which it was designed: minimally invasive surgery through telepresence, specifically with the robotic surgery slave system MiroSurge [HNJ⁺08]. Indeed, in that context of medical robotics, restricted vision, motion, and haptic perception are inevitable constraints that nonetheless can be alleviated with interfaces as the Sigma.7. Thanks to the device, suturing and palpation are possible, as well as other common fatigue-proof operations. User studies have been performed to test the device and related methods in the context of telesurgery [WHTL13].

The device was designed left/right-hand specific for a bi-manual console in which the layout and additional ergonomic parameters are carefully set. For this for thesis, and in particular during the user study from Section 6.2, only a right-handed Sigma.7 was used, together with an adjustable armrest to ease the characteristic wrist rotation movements required by the device.

*<http://www.forcedimension.com/products/sigma-7/overview>

Appendix B

Implementation and Performance Issues of the Primitive Data Structures

This appendix briefly reports relevant implementation and performance issues related to the primitive data structures introduced in Section 3.2.1: the voxelmap and the pointshell.

Since the collision and proximity queries must be carried out in realtime (every 1 ms), the implementation of the data structures and methods handled online must be as efficient as possible. All the structures and algorithms explained in this work are programmed in C++, having in general the C++ Standard Library [Ric14] as the only dependency, except in cases in which the use of third party libraries is explicitly specified.

The description of the whole framework design can be simplified to state that the basic primitive structures explained in Section 3.2.1 are implemented in the two main classes `Voxelmap` and `Pointshell`, analogous, respectively, to the structures **V** and **P** presented in that section. Each of them contains essentially a `std::vector` array of `Voxel` or `Point` class instances, respectively. Table B.1 describes those classes and summarizes the most important realtime methods associated to them with which the voxel value of a pointshell point can be obtained.

It is worth to highlight that, although the voxelmap is a regular 3D grid, its voxels are stored in an unidimensional array (see Table B.1(e) and (f)). This allows for keeping all values next to each other in memory. Accessing a random voxel value of a point has a $\mathcal{O}(1)$ complexity, independently of the resolution used for the voxelmap, if the

whole array is assumed to be allocated in a unique memory chunk [Ric14]. Therefore, a pointshell (collision) query has $\mathcal{O}(n)$ complexity, being n the number of points checked for collision.

As for the enhanced data structures and algorithms explained in and , the introduced `Voxelmap`, `Pointshell`, `Voxel`, and `Point` classes or types are inherited and extended to contain additional information. In the case of the voxelmaps, this information consists in improved distance-to-surface data, whereas topological neighborhood information is integrated for the pointshells.

Table B.1: Main realtime methods in the classes **Voxelmap (V)** and **Pointshell (P)** that lead to a voxel value v associated to a pointshell point P . The methods are listed in order of call ((a) – (f)).

Value	Call and Brief Implementation Description
(a) $P_i \leftarrow$	<code>Pointshell::getPoint(i)</code> Point coordinates are accessed in <code>Pointshell::points[i]</code> ; <code>Pointshell::points</code> represents \mathcal{P} and it is implemented as <code>std::vector<Point></code> , being <code>Point</code> a <code>class</code> which contains the pair $(P, \mathbf{n})_i$.
(b) $\mathbf{n}_i \leftarrow$	<code>Pointshell::getNormal(i)</code> Point normal vector is accessed in <code>Pointshell::points[i]</code> .
(c) $(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) \leftarrow$	<code>Voxelmap::getDiscreteCoordinates(P_i \leftarrow \mathbf{V}\mathbf{H}_\mathbf{P}P_i)</code> $\mathbf{X} = \lfloor \frac{P_{i,x}}{s} \rfloor$, $\mathbf{Y} = \lfloor \frac{P_{i,y}}{s} \rfloor$, $\mathbf{Z} = \lfloor \frac{P_{i,z}}{s} \rfloor$, with $P_i = (P_{i,x}, P_{i,y}, P_{i,z})^\top$. Assuming P_i is in pointshell coordinates, it must be transformed into voxelmap coordinates with $\mathbf{V}\mathbf{H}_\mathbf{P}$. If any of the discrete coordinates is outside of the boundaries, the closest boundary voxel is selected constraining each coordinate. For instance, if $\mathbf{X} < 0 \rightarrow \mathbf{X} = 0$, or if $\mathbf{X} > N_x - 1 \rightarrow \mathbf{X} = N_x - 1$.
(d) $C_i \leftarrow$	<code>Voxelmap::getVoxelCenter(X, Y, Z)</code> $C_i = s(\mathbf{X}, \mathbf{Y}, \mathbf{Z})^\top$. Assuming the origin of the voxelmap is set to the minimum point in the bounding box of the voxelmap (see Figure 3.2(a)).
(e) $j \leftarrow$	<code>Voxelmap::getVoxelIndex(X, Y, Z)</code> $j = \mathbf{Z} + \mathbf{Y}N_z + \mathbf{X}N_yN_z$. N_x, N_y, N_z are the total number of voxels in each axis and their product yields the total number of voxels $N_{\mathbf{V}} = N_xN_yN_z$.
(f) $v(P_i) \leftarrow$	<code>Voxelmap::getVoxelValue(j \xrightarrow{(a),(c),(e)} P_i)</code> Voxel layer value is accessed in <code>Voxelmap::voxels[j]</code> . <code>Voxelmap::voxels</code> represents \mathcal{V} and it is implemented as <code>std::vector<Voxel></code> , being <code>Voxel</code> a <code>typedef</code> of a type <code>int</code> . Note that the voxel value is constant for any point inside the voxel, hence, $v(P_i) = v(C_i)$.

Appendix C

Virtual Reality and Haptics: Evolution of Relevance

This appendix presents the evolution of the relevance related to the fields of *virtual reality*, *collision detection*, and *haptics* in the academia during the past years. The data was taken from <https://scholar.google.de> and is tabulated and illustrated in Table C.1 and Figure C.1.

In general, *haptic rendering* became an active field of research in the second half of the 90's, probably after the Phantom [MS+94] was released, one of the first commercial haptic devices. However, *collision detection* has been researched more extensively beforehand. Additionally, *haptics* seems to be increasingly more significant in the *virtual reality* community: if in the 90's the term appeared in $\sim 4\%$ of the publications containing also the term virtual reality, nowadays the percentage increased to $\sim 15\%$.

The number of found items seems to have declined in the first half of the 2010's; analyzing the reasons for that phenomenon is not in the scope of this thesis. The reader might consult the Gartner* technology hype and maturity cycles for a broader perspective.

*<http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp>

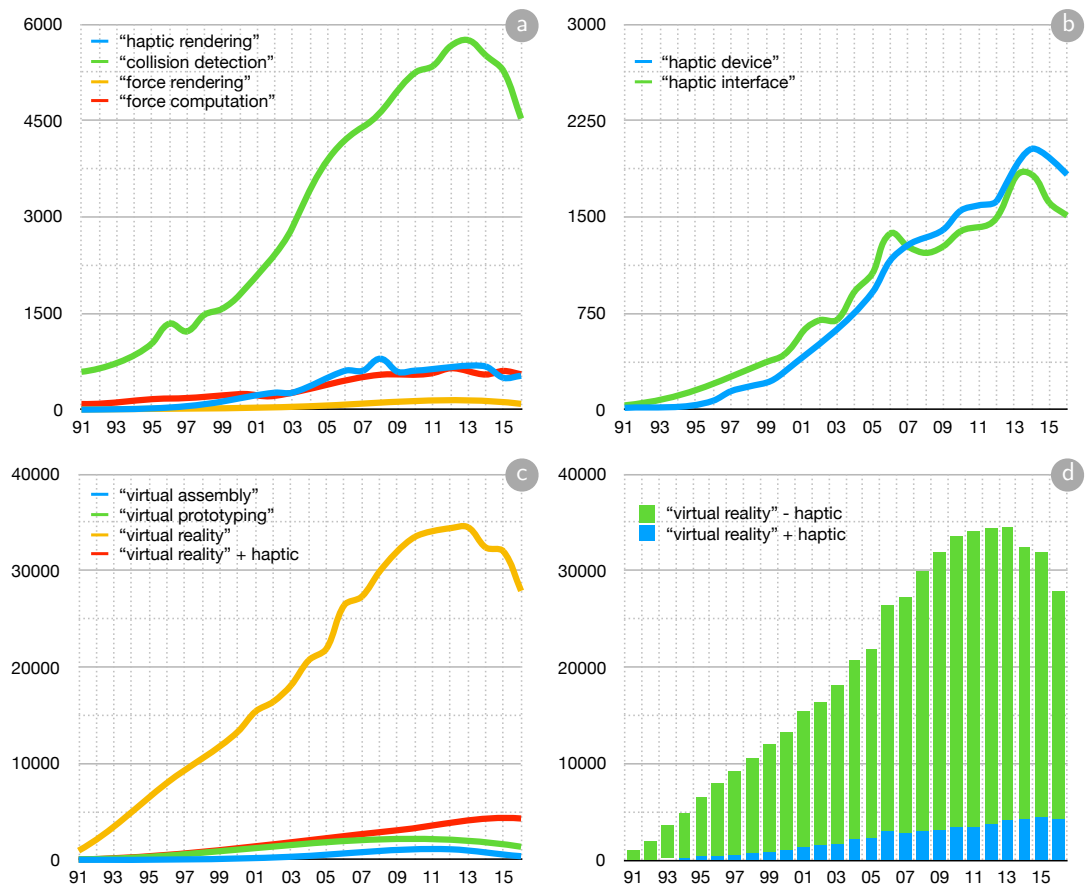


Figure C.1: Results from Google Scholar related to relevant Virtual Reality (VR) terms. The values were collected on February 7, 2017, under <https://scholar.google.de>, and are tabulated in Table C.1. Number of results from 1991 – 2016 are yearly plotted, displaying: (a) the development of terms related to *haptic rendering*, (b) *haptic interfaces*, (c) *virtual reality* and applications, and (d) the weight of *haptics* within the domain defined by the term *virtual reality*.

Table C.1: Results from Google Scholar related to relevant Virtual Reality (VR) terms; number of found items (#) and percentages (%) of nested terms (↔) are shown for each year or period of years. The values were collected on February 7, 2017, under <https://scholar.google.de>, and are plotted in Figure C.1.

Term	≤ 90	91	92	93	94	95	96	97	98	99	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	≤ 16	
"haptic rendering"	#	12	2	5	6	7	25	44	83	75	128	171	210	266	266	419	437	608	607	794	589	610	602	621	665	672	497	530	9140
"collision detection"	#	4095	591	621	764	860	1030	1340	1220	1480	1570	1790	2190	2400	2840	3560	3780	4400	4590	4940	5250	5350	5660	5750	5510	5270	4530	84800	
"force rendering"	#	117	5	8	6	5	10	16	15	19	12	25	21	34	53	55	73	93	101	117	112	129	120	134	121	129	125	92	1780
"force computation"	#	641	90	83	114	117	168	175	169	171	187	244	224	217	302	333	358	456	482	545	532	535	572	646	600	548	604	552	9810
"virtual assembly"	#	53	12	7	9	30	51	62	80	112	173	171	229	293	321	450	506	671	688	758	872	898	997	1100	871	660	509	381	11200
"virtual prototyping"	#	283	65	70	151	236	395	442	589	719	786	812	1030	1110	1330	1610	1790	1940	2400	2110	2400	2200	2280	2390	2260	1920	1650	1360	37000
"virtual reality"	#	3750	958	1980	3490	4910	6430	8000	9270	10600	12000	13300	16400	18100	20700	21900	26400	27300	29800	32000	33500	34100	34400	34500	32400	32000	27900	981000	
↔ + "haptic"	#	166	38	76	131	171	284	338	555	681	821	1030	1340	1540	1630	2080	2220	2840	2810	2970	3070	3350	3410	3720	4000	4270	4440	4590	59200
	%	4.4	4.0	3.8	3.8	3.5	4.4	4.2	6.0	6.4	6.9	7.7	8.7	9.4	10.1	10.1	10.8	10.3	9.9	9.6	10.0	10.0	10.8	11.6	13.2	13.9	15.4	6.0	
"haptic device"	#	127	15	22	33	18	51	70	144	180	210	292	436	509	585	801	913	1160	1280	1350	1400	1550	1620	1870	2030	1960	1830	23000	
"haptic interface"	#	275	34	49	78	79	132	182	234	315	371	427	596	696	699	921	1060	1370	1270	1220	1270	1390	1420	1490	1790	1830	1610	1510	23500

Appendix D

Publications by the Author

Table D.1: List of publications by the author, ordered chrono-alphabetically. The weight on the contents presented thesis is labeled from *none* () to *small* (○), *medium* (◐), and *strong* (●); additionally, related chapter(s) are indicated.

Reference	Publication Details	Weight	Chapter
[SHPH08]	Mikel Sagardia, Thomas Hulin, Carsten Preusche, and Gerd Hirzinger. Improvements of the voxmap-pointshell algorithm – fast generation of haptic data-structures. In <i>Internationales Wissenschaftliches Kolloquium (IWK, TU Ilmenau)</i> , 2008.	○	Chapter 3
[HSA ⁺ 08]	Thomas Hulin, Mikel Sagardia, Jordi Artigas, Simon Schaetzle, Philipp Kremer, and Carsten Preusche. Human-scale bimanual haptic interface. In <i>Proc. of the Int. Conf. on Enactive Interface</i> , 2008.	○	Chapter 6, Chapter A
[SHPH09]	Mikel Sagardia, Thomas Hulin, Carsten Preusche, and Gerd Hirzinger. A benchmark of force quality in haptic rendering. In <i>Proc. of the Int. Conf. on Human-Computer Interaction (HCI)</i> , 2009.		
[WSM ⁺ 10]	Rene Weller, Mikel Sagardia, David Mainzer, Thomas Hulin, Gabriel Zachmann, and Carsten Preusche. A benchmarking suite for 6-dof real time collision response algorithms. In <i>Proc. ACM Symp. on Virtual Reality and Software Technology (VRST)</i> , pages 63–70. ACM, 2010.		

- [[HHK⁺11](#)] Thoma Hulin, Katharina Hertkorn, Philipp Kremer, Simon Schätzle, Jordi Artigas, Mikel Sagardia, Franziska Zacharias, and Carsten Preusche. The DLR bimanual haptic device with optimized workspace (video). In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3441–3442. IEEE, 2011. ○ Chapter 6, Chapter A
- [[SWH⁺12](#)] Mikel Sagardia, Bernhard Weber, Thomas Hulin, Carsten Preusche, and Gerd Hirzinger. Evaluation of visual and force feedback in virtual assembly verifications. In *Proc. IEEE Virtual Reality (VR)*, pages 23–26. IEEE, 2012. **Best Short Paper Nomination Award.**
- [[SH13](#)] Mikel Sagardia and Thomas Hulin. Fast and accurate distance, penetration, and collision queries using point-sphere trees and distance fields. In *ACM SIGGRAPH Posters*, page 83. ACM, 2013. ● Chapter 3
- [[SHH⁺13](#)] Mikel Sagardia, Katharina Hertkorn, Thomas Hulin, Robin Wolff, Johannes Hummel, Janki Dodiya, and Andreas Gerndt. An interactive virtual reality system for on-orbit servicing. In *Proc. IEEE Virtual Reality (VR)*, 2013. (Video). ○ Chapter 5
- [[WSHP13](#)] Bernhard Weber, Mikel Sagardia, Thomas Hulin, and Carsten Preusche. Visual, vibrotactile, and force feedback of collisions in virtual environments: effects on performance, mental workload and spatial orientation. In *Virtual Augmented and Mixed Reality. Designing and Developing Augmented and Virtual Environments*, volume 8021 of *Lecture Notes in Computer Science*, pages 241–250. Springer Berlin Heidelberg, 2013. **Best Paper Award.**
- [[DMB⁺13](#)] Ralf Dörner, Geert Matthys, Manfred Bogen, Stefan Rilling, Andreas Gerndt, Janki Dodiya, Katharina Hertkorn, Thomas Hulin, Johannes Hummel, Mikel Sagardia, Robin Wolff, Tom Kühnert, Guido Brunnett, Hagen Buchholz, Lisa Blum, Christoffer Menk, Christian Bade, Werner Schreiber, Matthias Greiner, Thomas Alexander, Michael Kleiber, Gerd Bruder, and Frank Steinicke. *Virtual und Augmented Reality (VR / AR)*, chapter Fallbeispiele für VR/AR, pages 295–326. Springer Berlin Heidelberg, 2013.

- [SSeS14a] Mikel Sagardia, Theodoros Stouraitis, and João Lopes e Silva. A New Fast and Robust Collision Detection and Force Computation Algorithm Applied to the Physics Engine Bullet: Method, Integration, and Evaluation. In *Prof. of the Conf. and Exhibition of the European Association of Virtual and Augmented Reality (EuroVR)*, pages 65–76. Eurographics Association, 2014. ● Chapter 3, Chapter 5
- [SSeS14b] Mikel Sagardia, Theodoros Stouraitis, and João Lopes e Silva. Poster: Integration of a haptic rendering algorithm based on voxelized and point-sampled structures into the physics engine bullet. In *Proc. IEEE Symposium on 3D User Interfaces (3DUI)*, pages 133–134. IEEE, 2014. ● Chapter 3, Chapter 5
- [CHS⁺14] Claudio Castellini, Katharina Hertkorn, Mikel Sagardia, David Sierra González, and Markus Nowak. A virtual piano-playing environment for rehabilitation based upon ultrasound imaging. In *Proc. IEEE RAS & EMBS Int. Conf. on Biomedical Robotics and Biomechatronics*, pages 548–554. IEEE, 2014. ○ Chapter 5
- [SHGC14] Mikel Sagardia, Katharina Hertkorn, David Sierra González, and Claudio Castellini. Ultrapiano: A novel human-machine interface applied to virtual reality (video). In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2089–2089. IEEE, 2014. **Best Video Nomination.** ○ Chapter 5
- [SHH⁺15] Mikel Sagardia, Katharina Hertkorn, Thomas Hulin, Simon Schätzle, Robin Wolff, Johannes Hummel, Janki Dodiya, and Andreas Gerndt. VR-OOS: The DLR’s virtual reality simulator for telerobotic on-orbit servicing with haptic feedback. In *Proc. IEEE Aerospace Conf.*, pages 1–17, 2015. ○ Chapter 5
- [JLP⁺15] Steffen Jaekel, Roberto Lampariello, Giogio Panin, Mikel Sagardia, Bernhard Brunner, Oliver Porges, Erich Kraemer, Matthias Wieser, Richard Haarmann, Markus Pietras, and Robin Biesbrock. Robotic capture and de-orbiting of a heavy, uncooperative and tumbling target satellite in low earth orbit. In *Proc. Symp. on Advanced Space Technologies in Robotics and Automation (ASTRA)*, 2015. Chapter 5

- | | | | |
|-----------------------|--|---|-----------|
| [NSSB16] | Korbinian Nottensteiner, Mikel Sagardia, Andreas Stemmer, and Christoph Borst. Narrow passage sampling in the observation of robotic assembly tasks. In <i>Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)</i> , pages 130–137. IEEE, 2016. | ○ | Chapter 5 |
| [SH16] | Mikel Sagardia and Thomas Hulin. A fast and robust six-dof god object heuristic for haptic rendering of complex models with friction. In <i>Proc. ACM Symp. on Virtual Reality and Software Technology (VRST)</i> , pages 163–172. ACM, 2016. | ● | Chapter 4 |
| [SHH ⁺ 16] | Mikel Sagardia, Thomas Hulin, Katharina Hertkorn, Philipp Kremer, and Simon Schätzle. A platform for bimanual virtual assembly training with haptic feedback in large multi-object environments. In <i>Proc. ACM Symp. on Virtual Reality and Software Technology (VRST)</i> , pages 153–162. ACM, 2016. | ● | Chapter 5 |
| [SH17a] | Mikel Sagardia and Thomas Hulin. Evaluation of a penalty and a constraint-based haptic rendering algorithm with different haptic interfaces and stiffness values. In <i>Proc. IEEE Virtual Reality (VR)</i> , pages 64–73. IEEE, 2017. | ● | Chapter 6 |
| [SH17b] | Mikel Sagardia and Thomas Hulin. Multimodal evaluation of the differences between real and virtual assemblies. <i>IEEE Trans. on Haptics</i> , 11:107–118, 2017. | ● | Chapter 6 |
| [STH18] | Mikel Sagardia, Alexander Martín Turrillas, and Thomas Hulin. Realtime collision avoidance for mechanisms with complex geometries. In <i>Proc. IEEE Virtual Reality (VR)</i> , 2018. (Video). | ○ | Chapter 5 |
-

Bibliography

- [AAAAD16] Abdulrahman M Al-Ahmari, Mustufa H Abidi, Ali Ahmad, and Saber Darmoul. Development of a virtual manufacturing assembly simulation system. *Advances in Mechanical Engineering*, 8(3), 2016. [cited on page(s) 145]
- [AC06] Federico Avanzini and Paolo Crosato. Haptic-auditory rendering and perception of contact stiffness. In *Int. Workshop on Haptic and Audio Interaction Design*, pages 24–35. Springer, 2006. [cited on page(s) 12, 206, 219]
- [AGG⁺03] Marco Agus, Andrea Giachetti, Enrico Gobbetti, Gianluigi Zanetti, and Antonio Zorcolo. Real-time haptic and visual simulation of bone dissection. *Presence: Teleoperators and Virtual Environments*, 12(1):110–122, 2003. [cited on page(s) 62]
- [AGHWK16] Christoph Anthes, Rubén Jesús García-Hernández, Markus Wiedemann, and Dieter Kranzlmüller. State of the art of virtual reality technology. In *Proc. IEEE Aerospace Conf.*, pages 1–19, 2016. [cited on page(s) 16]
- [AH99] Richard J Adams and Blake Hannaford. Stable haptic interaction with virtual environments. *IEEE Trans. Robotics and Automation*, 15(3):465–474, 1999. [cited on page(s) 23]
- [AKO95] Yoshitaka Adachi, Takahiro Kumano, and Kouichi Ogino. Intermediate representation for stiff virtual objects. In *Proc. IEEE Virtual Reality (VR)*, pages 203–210. IEEE, 1995. [cited on page(s) 36]
- [AM01] Thomas Akenine-Möller. Fast 3d triangle-box overlap testing. Technical report, Department of Computer Engineering, Chalmers University of Technology, 2001. [cited on page(s) 38, 70]
- [AMF16] A Aguilera, FJ Melero, and FR Feito. Out-of-core real-time haptic interaction on very large models. *Computer-Aided Design*, 77:98–106, 2016. [cited on page(s) 34]

- [ASB07] Sören Andersson, Anders Söderberg, and Stefan Björklund. Friction models for sliding dry, boundary and mixed lubricated contacts. *Tribology international*, 40(4):580–587, 2007. [cited on page(s) 58, 117]
- [ASHO⁺07] Alin Albu-Schäffer, Sami Haddadin, Christian Ott, Andreas Stemmer, Thomas Wimböck, and Gerd Hirzinger. The dlr lightweight robot: Design and control concepts for robots in human environments. *Industrial Robot: An Int. Journal*, 34(5):376—385, 2007. [cited on page(s) 171, 227]
- [Bar89] David Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Proc. ACM SIGGRAPH*, 23(3):223–232, 1989. [cited on page(s) 51, 54, 57]
- [Bar92] David Baraff. *Dynamic Simulation of Non-Penetrating Rigid Bodies*. PhD thesis, Cornell University, 1992. [cited on page(s) 146]
- [Bar93] David Baraff. Issues in computing contact forces for non-penetrating rigid bodies. *Algorithmica*, 10(2-4):292–352, 1993. [cited on page(s) 218]
- [Bar94] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proc. ACM SIGGRAPH*, pages 23–34. ACM, 1994. [cited on page(s) 54]
- [Bar97a] David Baraff. An introduction to physically based modeling: Rigid body simulation i - unconstrained rigid body dynamics. Technical report, Carnegie Mellon University, 1997. SIGGRAPH Course Notes. [cited on page(s) 16]
- [Bar97b] David Baraff. An introduction to physically based modeling: Rigid body simulation ii - nonpenetration constraints. Technical report, Carnegie Mellon University, 1997. SIGGRAPH Course Notes. [cited on page(s) 16]
- [Bar07] Jernej Barbič. *Real-time Reduced Large-Deformation Models and Distributed Contact for Computer Graphics and Haptics*. PhD thesis, Carnegie Mellon University, 2007. [cited on page(s) 77]
- [Bar10] Woodrow Barfield. The use of haptic display technology in education. *Themes in science and technology education*, 2(1-2):11–30, 2010. [cited on page(s) 26]
- [BB07] Adrian Boeing and Thomas Bräunl. Evaluation of real-time physics simulation systems. In *Proc. ACM SIGGRAPH*, pages 281–288. ACM, 2007. [cited on page(s) 15, 146, 177]
- [BBT04] Abdouslam M Bashir, Robert Bicker, and Paul M Taylor. An investigation into different visual/tactual feedback modes for a virtual object manipulation task. In *Proc. ACM SIGGRAPH Int. Conf. on Virtual Reality Continuum and its Applications in Industry*, pages 359–362. ACM, 2004. [cited on page(s) 178]
- [BDH96] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996. [cited on page(s) 33]

- [BDRyB14] Stuart A Bowyer, Brian L Davies, and Ferdinando Rodriguez y Baena. Active constraints/virtual fixtures: A survey. *IEEE Trans. on Robotics*, 30(1):138–157, 2014. [cited on page(s) 24]
- [BDSR06] Jurgen Broeren, Mark Dixon, Katharina Stibrant Sunnerhagen, and Martin Rydmark. Rehabilitation after stroke using virtual reality, haptics (force feedback) and telemedicine. *Studies in health technology and informatics*, 124:51, 2006. [cited on page(s) 26]
- [BDW⁺03] Aaron Bloomfield, Yu Deng, Jeff Wampler, Pascale Rondot, Dina Harth, Mary McManus, and Norman Badler. A taxonomy and comparison of haptic actions for disassembly tasks. In *Proc. IEEE Virtual Reality (VR)*, pages 225–231. IEEE, 2003. [cited on page(s) 183]
- [BETC14] Jan Bender, Kenny Erleben, Jeff Trinkle, and Erwin Coumans. Interactive simulation of rigid body dynamics in computer graphics. *Computer Graphics Forum*, 33(1):246–270, 2014. [cited on page(s) 15, 146]
- [BH99] Doug A Bowman and Larry F Hodges. Formalizing the design, evaluation, and application of interaction techniques for immersive virtual environments. *J. of Visual Languages & Computing*, 10(1):37–53, 1999. [cited on page(s) 178]
- [BJ08] Jernej Barbič and Doug L. James. Six-dof haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Trans. on Haptics*, 1(1):39–52, 2008. [cited on page(s) 42, 47, 60, 67, 68, 101, 151]
- [BKLJP01] Doug A Bowman, Ernst Kruijff, Joseph J LaViola Jr, and Ivan Poupyrev. An introduction to 3-d user interface design. *Presence: Teleoperators and virtual environments*, 10(1):96–108, 2001. [cited on page(s) 16, 17]
- [BLBH12] Manuel Brucker, Simon Léonard, Tim Bodenmüller, and Gregory D Hager. Sequential scene parsing using range and intensity information. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 5417–5424. IEEE, 2012. [cited on page(s) 172]
- [BM07] Doug A Bowman and Ryan P McMahan. Virtual reality: how much immersion is enough? *Computer*, 40(7):36–43, 2007. [cited on page(s) 14, 181, 219]
- [BPT15] Yuval S Boger, Ryan A Pavlik, and Russell M Taylor. Osvr: An open-source virtual reality platform for both industry and academia. In *Proc. IEEE Virtual Reality (VR)*, pages 383–384. IEEE, 2015. [cited on page(s) 17]
- [BPW14] Cheryl Campanella Bracken, Gary Pettey, and Mu Wu. Revisiting the use of secondary task reaction time measures in telepresence research: exploring the role of immersion and attention. *AI & society*, 29(4):533–538, 2014. [cited on page(s) 206, 219]

- [BS02] Cagatay Basdogan and Ayam A. Srinivasan. Haptic rendering in virtual environments. *Stanney, K. (Ed.), Handbook of Virtual Environments*, pages 117–134, 2002. [cited on page(s) 11, 15, 19, 21, 116]
- [Cam97] Stephen Cameron. Enhancing gjk: Computing minimum and penetration distances between convex polyhedra. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, volume 97, pages 20–25, 1997. [cited on page(s) 40]
- [CB94] J. Edward Colgate and J. Michael Brown. Factors affecting the z-width of a haptic display. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3205–3210. IEEE, 1994. [cited on page(s) 21]
- [CB17] Erwin Coumans and Yunfei Bai. pybullet, a python module for physics simulation in robotics, games and machine learning. Technical report, Google Brain, 2017. [cited on page(s) 26]
- [CBS13] Shing-Chow Chan, Nikolas H Blevins, and Kenneth Salisbury. Deformable haptic rendering for volumetric medical image data. In *Proc. IEEE World Haptics Conference*, pages 73–78. IEEE, 2013. [cited on page(s) 55, 61, 117]
- [CCBS11] Sonny Chan, François Conti, Nikolas H. Blevins, and Kenneth Salisbury. Constraint-based six degree-of-freedom haptic rendering of volume-embedded isosurfaces. In *Proc. IEEE World Haptics Conference*, pages 89–94. IEEE, 2011. [cited on page(s) 34, 43, 67, 117]
- [CD68] Richard W Cottle and George B Dantzig. Complementary pivot theory of mathematical programming. *Linear algebra and its applications*, 1(1):103–125, 1968. [cited on page(s) 54]
- [Cha84] Bernard Chazelle. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM Journal on Computing*, 13(3):488–507, 1984. [cited on page(s) 37]
- [CHS⁺14] Claudio Castellini, Katharina Hertkorn, Mikel Sagardia, David Sierra González, and Markus Nowak. A virtual piano-playing environment for rehabilitation based upon ultrasound imaging. In *Proc. IEEE RAS & EMBS Int. Conf. on Biomedical Robotics and Biomechatronics*, pages 548–554. IEEE, 2014. [cited on page(s) 170, 241]
- [CJA⁺10] Hadrien Courtecuisse, Hoeryong Jung, Jérémie Allard, Christian Duriez, Doo Yong Lee, and Stéphane Cotin. Gpu-based real-time soft tissue deformation with cutting and haptic feedback. *Progress in biophysics and molecular biology*, 103(2-3):159–168, 2010. [cited on page(s) 61]
- [CK05] Francois Conti and Oussama Khatib. Spanning large workspaces using small haptic devices. In *Proc. IEEE World Haptics Conference*, pages 183–188. IEEE, 2005. [cited on page(s) 20, 156]

- [CKB03] Francois Conti, Oussama Khatib, and Charles Baur. Interactive rendering of deformable objects based on a filling sphere modelling approach. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 3716–3721. IEEE, 2003. [cited on page(s) 36, 60]
- [CLMP95] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, and Madhav Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proc. of ACM Interactive 3D Graphics Conference*, pages 189–ff. ACM, 1995. [cited on page(s) 32, 44, 46, 151]
- [CMJ11] T.R. Coles, D. Meglan, and N. John. The role of haptics in medical training simulators: A survey of the state of the art. *IEEE Trans. on Haptics*, 4(1):51–66, 2011. [cited on page(s) 26, 176]
- [CNSD93] Carolina Cruz-Neira, Daniel J Sandin, and Thomas A DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *Proc. ACM SIGGRAPH*, pages 135–142. ACM, 1993. [cited on page(s) 16]
- [COPG15] Loïc Corenthy, Miguel A Otaduy, Luis Pastor, and Marcos García. Volume haptics with topology-consistent isosurfaces. *IEEE Trans. on Haptics*, 8(4):480–491, 2015. [cited on page(s) 26]
- [Cou03] Erwin Coumans. Bullet physics library 2.82, 2003. Web page accessed on June 30, 2016. [cited on page(s) 5, 15, 100, 105, 113, 143, 146, 147, 161, 162, 173, 174, 224]
- [CS97] J Edward Colgate and Gerd G Schenkel. Passivity of a class of sampled-data systems: Application to haptic interfaces. *J. of Robotic Systems*, 14(1):37–47, 1997. [cited on page(s) 22]
- [CSB95] J. E. Colgate, M. C. Stanley, and J. M. Brown. Issues in the haptic display of tool use. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 3, pages 140–145, 1995. [cited on page(s) 22, 54, 55, 56, 57, 66, 116]
- [CSC05] Daniela Constantinescu, Septimiu E. Salcudean, and Elizabeth A. Croft. Haptic rendering of rigid contacts using impulsive and penalty forces. *IEEE Trans. on Robotics*, 21(3):309–323, 2005. [cited on page(s) 51, 52]
- [DABS06] Lionel Dominjon, Lécuyer Anatole, Jean-Marie Burkhardt, and Richir Simon. A Comparison of Three Techniques to Interact in Large Virtual Environments Using Haptic Devices with Limited Workspace. *J. of Material Forming*, 4035:288–299, 2006. [cited on page(s) 156]
- [Dan63] George Dantzig. *Linear programming and extensions*. Princeton university press, 1963. [cited on page(s) 40]

- [DBS97] David E DiFranco, G Lee Beauregard, and Mandavam A Srinivasan. The effect of auditory cues on the haptic perception of stiffness in virtual environments. In *Proc. of the ASME Dynamic Systems and Control Division*, volume 61, pages 17–22. American Society of Mechanical Engineers (ASME), 1997. [cited on page(s) 12, 206, 219]
- [DDKA06] Christian Duriez, Frédéric Dubois, Abderrahmane Kheddar, and Claude Andriot. Realistic haptic rendering of interacting deformable objects in virtual environments. *IEEE Trans. on Visualization and Computer Graphics*, 12(1):36–47, 2006. [cited on page(s) 60]
- [Dem07] Barbara Deml. Human factors issues on the design of telepresence systems. *Presence: Teleoperators and Virtual Environments*, 16(5):471–487, 2007. [cited on page(s) 183]
- [DMB⁺13] Ralf Dörner, Geert Matthys, Manfred Bogen, Stefan Rilling, Andreas Gerndt, Janki Dodiya, Katharina Hertkorn, Thomas Hulin, Johannes Hummel, Mikel Sagardia, Robin Wolff, Tom Kühnert, Guido Brunnett, Hagen Buchholz, Lisa Blum, Christoffer Menk, Christian Bade, Werner Schreiber, Matthias Greiner, Thomas Alexander, Michael Kleiber, Gerd Bruder, and Frank Steinicke. *Virtual und Augmented Reality (VR / AR)*, chapter Fallbeispiele für VR/AR, pages 295–326. Springer Berlin Heidelberg, 2013. [cited on page(s) 240]
- [DZ93] Paul Dworkin and David Zeltzer. A new model for efficient dynamic simulation. In *Proc. of the Eurographics Workshop on Animation and Simulation*, 1993. [cited on page(s) 43, 45, 54, 116]
- [EB02] Marc O Ernst and Martin S Banks. Humans integrate visual and haptic information in a statistically optimal fashion. *Nature*, 415(6870):429–433, 2002. [cited on page(s) 12, 176]
- [EBB00] Marc O Ernst, Martin S Banks, and Heinrich H Bühlhoff. Touch can change visual slant perception. *Nature neuroscience*, 3(1):69–73, 2000. [cited on page(s) 12]
- [Ebe99] David Eberly. Distance between point and triangle in 3d. In *Magic Software*, 1999. [cited on page(s) 38, 72]
- [EL00] Stephen A. Ehmann and Ming C. Lin. Accelerated proximity queries between convex polyhedra by multi-level voronoi marching. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 3, pages 2101–2106. IEEE, 2000. [cited on page(s) 32, 41]
- [EL01] Stephen A. Ehmann and Ming C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum*, 20(3):500–511, 2001. [cited on page(s) 48]

- [EMK07] Adrien Escande, Sylvain Miossec, and Abderrahmane Kheddar. Continuous gradient proximity distance for humanoids free-collision optimized-postures. In *Proc. IEEE-RAS Int. Conf. on Humanoid Robots*, pages 188–195. IEEE, 2007. [cited on page(s) 33, 37]
- [Erl05] Kenny Erleben. *Stable, Robust, and Versatile Multibody Dynamics Animation*. PhD thesis, University of Copenhagen, 2005. [cited on page(s) 146]
- [FBAF08] François Faure, Sébastien Barbier, Jérémie Allard, and Florent Falipou. Image-based collision detection and response between arbitrary volume objects. In *Proc. of the Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, pages 155–162. Eurographics Association, 2008. [cited on page(s) 38]
- [FDS90] Patrick Fischer, Ron Daniel, and KV Siva. Specification and design of input devices for teleoperation. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 540–545. IEEE, 1990. [cited on page(s) 19]
- [Fec60] G Th Fechner. *Elemente der Psychophysik*. Breitkopf & Härtel, 1860. [cited on page(s) 10]
- [FG04] Kaspar Fischer and Bernd Gärtner. The smallest enclosing balls of balls: Combinatorial structure and algorithms. *International Journal of Computational Geometry & Applications*, 14:341–378, 2004. [cited on page(s) 68, 92]
- [FGH⁺16] Kaspar Fischer, Bernd Gärtner, Thomas Herrmann, Michael Hoffmann, and Sven Schönherr. Bounding volumes. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.9 edition, 2016. [cited on page(s) 92]
- [FSG03] Arnulph Fuhrmann, Gerrit Sobotka, and Clemens Groß. Distance fields for rapid collision detection in physically based modeling. In *Proc. of GraphiCon*, pages 58–65, 2003. [cited on page(s) 34]
- [FUF06] Christoph Fünfzig, Torsten Ullrich, and Dieter W Fellner. Hierarchical spherical distance fields for collision detection. *IEEE Computer Graphics and Applications*, 26(1):64–74, 2006. [cited on page(s) 33]
- [Gau29] Carl Friedrich Gauss. Über ein neues allgemeines grundgesetz der mechanik. *Reine angewandte Mathematik*, 4:232–235, 1829. [cited on page(s) 54, 116, 134]
- [GBMCL⁺14] Germanico Gonzalez-Badillo, Hugo Medellin-Castillo, Theodore Lim, James Ritchie, and Samir Garbaya. The development of a physics and constraint-based haptic virtual assembly system. *Assembly Automation*, 34(1):41–55, 2014. [cited on page(s) 145, 178, 218]
- [GFL04] Philippe Garrec, Jean-Pierre Friconneau, and François Louveaux. Virtuose 6d: a new force-control master arm using innovative ball-screw actuators. In *Int. Symp. on Robotics (ISR)*, 2004. Web page accessed on February 26, 2018. [cited on page(s) 18, 19]

- [GJK88] Elmer G. Gilbert, Daniel W. Johnson, and S. Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, 1988. [cited on page(s) 32, 36, 37, 40, 100, 146]
- [GLGT05] Arthur Gregory, Ming C. Lin, Stefan Gottschalk, and Russell Taylor. A framework for fast and accurate collision detection for haptic interaction. In *ACM SIGGRAPH Courses*, page 34, 2005. [cited on page(s) 34, 44]
- [GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *Proc. ACM SIGGRAPH*, pages 171–180. ACM, 1996. [cited on page(s) 32, 33, 39, 46, 47]
- [GME⁺00] Arthur Gregory, Ajith Mascarenhas, Stephen Ehmann, Ming Lin, and Dinesh Manocha. Six degree-of-freedom haptic display of polygonal models. In *Proc. IEEE Visualization*, pages 139–146. IEEE Computer Society Press, 2000. [cited on page(s) 32, 41, 46]
- [GO09] Carlos Garre and Miguel A. Otaduy. Haptic rendering of complex deformations through handle-space force linearization. In *Proc. Eurohaptics Conf.*, pages 422–427. IEEE, 2009. [cited on page(s) 36, 60]
- [GRV⁺10] T Gutiérrez, J Rodríguez, Yaiza Velaz, Sara Casado, A Suescun, and Emilio J Sánchez. Ima-vr: a multimodal virtual training system for skills transfer in industrial maintenance and assembly tasks. In *Proc. IEEE Int. Symp. on Robots and Human Interactive Communications (ROMAN)*, pages 428–433. IEEE, 2010. [cited on page(s) 145, 157]
- [GSW97] Rakesh Gupta, Thomas Sheridan, and Daniel Whitney. Experiments using multimodal virtual environments in design for assembly analysis. *Presence: Teleoperators and Virtual Environments*, 6(3):318–338, 1997. [cited on page(s) 176, 177, 183, 219]
- [GW08] Antony W. Goodwin and Heather E. Wheat. *Human Haptic Perception*, chapter Physiological Mechanism of the Receptor System, pages 93–102. Birkhaeuser Verlag, 2008. [cited on page(s) 9, 10]
- [GZC07] Samir Garbaya and U. Zaldivar-Colado. The affect of contact force sensations on user performance in virtual assembly tasks. *Virtual Reality*, 11(4):287–299, 2007. [cited on page(s) 176, 178]
- [HA00] Vincent Hayward and Brian Armstrong. A new computational model of friction applied to haptic rendering. In *Experimental Robotics VI*, pages 403–412. Springer, 2000. [cited on page(s) 58, 117]

- [Hay08] Vincent Hayward. *Human Haptic Perception*, chapter Haptic Shape Cues, Invariants, Priors and Interface Design, pages 381–392. Birkhaeuser Verlag, 2008. [cited on page(s) 12]
- [HB08] Zdenek Halata and Klaus I. Baumann. *Human Haptic Perception*, chapter Anatomy of Receptors, pages 85–92. Birkhaeuser Verlag, 2008. [cited on page(s) 8, 9]
- [HB12] Sandra Hirche and Martin Buss. Human-oriented control for haptic teleoperation. *Proceedings of the IEEE*, 100(3):623–647, 2012. [cited on page(s) 24]
- [HBA⁺06] Matthias Harders, Alexander Barlit, Katsuhito Akahane, Makoto Sato, and Gabor Szekely. Comparing 6dof haptic interfaces for application in 3d assembly tasks. In *Proc. Eurohaptics Conf.*, volume 6, pages 523–526, 2006. [cited on page(s) 178, 200]
- [HBDH93] Gerd Hirzinger, Bernhard Brunner, Johannes Dietrich, and Johann Heindl. Sensor-based space robotics-rotex and its telerobotic features. *IEEE Transactions on Robotics and Automation*, 9(5):649–663, 1993. [cited on page(s) 24]
- [HBS99] Chih-Hao Ho, Cagatay Basdogan, and Mandayam A Srinivasan. Efficient point-based rendering techniques for haptic display of virtual objects. *Presence*, 8(5):477–491, 1999. [cited on page(s) 44]
- [HDB⁺14] Andreas Hermann, Florian Drews, Joerg Bauer, Sebastian Klemm, Arne Roennau, and Ruediger Dillmann. Unified gpu voxel collision detection for mobile manipulation planning. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 4154–4160. IEEE, 2014. [cited on page(s) 34, 43, 45, 67]
- [HDE⁺16] Johannes Hummel, Janki Dodiya, Laura Eckardt, Robin Wolff, Andreas Gerndt, and Torsten W. Kuhlen. A lightweight electrotactile feedback device for grasp improvement in immersive virtual environments. In *Proc. IEEE Virtual Reality (VR)*, pages 39 – 48. IEEE, 2016. [cited on page(s) 168]
- [Her96] Heinrich Hertz. Über die berührung fester elastischer körper – on the contact of elastic solids. *Reine und angewandte Mathematik*, 1896. [cited on page(s) 62]
- [Her15] Katharina Hertkorn. *Shared Grasping: A Combination of Telepresence and Grasp Planning*. PhD thesis, Karlsruher Institut für Technologie (KIT) and German Aerospace Center (DLR), 2015. [cited on page(s) 24, 67, 100, 172, 206, 219]
- [HHC⁺08] Peter Hinterseer, Sandra Hirche, Subhasis Chaudhuri, Eckehard Steinbach, and Martin Buss. Perception-based data reduction and transmission of haptic data in telepresence and teleaction systems. *IEEE Transactions on Signal Processing*, 56(2):588–597, 2008. [cited on page(s) 11]

- [HHK⁺11] Thoma Hulin, Katharina Hertkorn, Philipp Kremer, Simon Schätzle, Jordi Artigas, Mikel Sagardia, Franziska Zacharias, and Carsten Preusche. The DLR bimanual haptic device with optimized workspace (video). In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3441–3442. IEEE, 2011. [cited on page(s) 6, 18, 19, 150, 154, 179, 184, 201, 204, 227, 228, 240]
- [HLC⁺97] Thomas C. Hudson, Ming C. Lin, Jonathan Cohen, Stefan Gottschalk, and Dinesh Manocha. V-collide: Accelerated collision detection for vrml. In *Proceedings VRML*, pages 117–ff. ACM, 1997. [cited on page(s) 44]
- [HM02] William S. Harwin and Nicholas Melder. Improved haptic rendering for multi-finger manipulation using friction cone based god-objects. In *Proc. Eurohaptics Conf.*, pages 82–85, 2002. [cited on page(s) 58, 117, 133]
- [HNJ⁺08] U Hagn, M Nickl, S Jörg, G Passig, T Bahls, A Nothhelfer, F Hacker, L Le-Tien, A Albu-Schäffer, R Konietschke, M Grebenstein, R Warpup, R Hasslinger, M Frommberger, and G Hirzinger. The dlr miro: a versatile lightweight robot for surgical applications. *Industrial Robot: An International Journal*, 35(4):324–336, 2008. [cited on page(s) 229]
- [HO08] Blake Hannaford and Allison M Okamura. *Springer Handbook of Robotics*, chapter Haptics, pages 1063–1084. Springer, 2008. [cited on page(s) 19, 21, 26]
- [HR02] Blake Hannaford and Jee-Hwan Ryu. Time-domain passivity control of haptic interfaces. *IEEE Transactions on Robotics and Automation*, 18(1):1–10, 2002. [cited on page(s) 22]
- [HR09] David Hecht and Miriam Reiner. Sensory dominance in combinations of audio, visual and haptic stimuli. *Experimental brain research*, 193(2):307–314, 2009. [cited on page(s) 12, 176, 206]
- [HS04] Shoichi Hasegawa and Makoto Sato. Real-time rigid body simulation for haptic interactions based on contact volume of polygonal objects. *Computer Graphics Forum*, 23(3):529–538, 2004. [cited on page(s) 38]
- [HS16] Xiyuan Hou and Olga Sourina. Real-time adaptive prediction method for smooth haptic rendering. *arXiv preprint arXiv:1603.06674*, 2016. [cited on page(s) 25, 49]
- [HSA⁺08] Thomas Hulin, Mikel Sagardia, Jordi Artigas, Simon Schaetzle, Philipp Kremer, and Carsten Preusche. Human-scale bimanual haptic interface. In *Proc. of the Int. Conf. on Enactive Interface*, 2008. [cited on page(s) 132, 239]
- [Hub96] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. on Graphics*, 15(3):179–210, 1996. [cited on page(s) 32, 46]

- [Hul17] Thomas Hulin. *Control of Hybrid Systems Affected by Time Delay with Application in Haptic Rendering*. PhD thesis, Leibniz Universitaät Hannover, 2017. [cited on page(s) 22, 23]
- [HV07] Brad M Howard and Judy M Vance. Desktop haptic virtual assembly using physically based modelling. *Virtual Reality*, 11(4):207–215, 2007. [cited on page(s) 145]
- [HWS⁺12] Johannes Hummel, Robin Wolff, Tobias Stein, Andreas Gerndt, and Torsten Kuhlen. An evaluation of open source physics engines for use in virtual reality assembly simulations. In *Advances in Visual Computing*, pages 346–357. Springer, 2012. [cited on page(s) 15, 146]
- [HY08] Steven Hsiao and Jeffrey Yau. *Human Haptic Perception*, chapter Neural Basis of Haptic Perception, pages 103–112. Birkhaeuser Verlag, 2008. [cited on page(s) 10]
- [ICG⁺06] Rosa Iglesias, Sara Casado, Teresa Gutierrez, Alejandro García-Alonso, Kian Meng Yap, Wai Yu, and Alan Marshall. A peer-to-peer architecture for collaborative haptic assembly. In *Proc. IEEE Int. Symp. on Distributed Simulation and Real-Time Applications*, pages 25–34. IEEE, 2006. [cited on page(s) 157]
- [JBS06] Mark W Jones, J Andreas Baerentzen, and Milos Sramek. 3d distance fields: A survey of techniques and applications. *IEEE Transactions on visualization and Computer Graphics*, 12(4):581–599, 2006. [cited on page(s) 34]
- [JC98] David E Johnson and Elaine Cohen. A framework for efficient minimum distance computations. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, volume 4, pages 3678–3684. IEEE, 1998. [cited on page(s) 42]
- [JCRR09] Li Jiang, Mark R Cutkosky, Juhani Ruutiainen, and Roope Raisamo. Using haptic feedback to improve grasp force control in multiple sclerosis patients. *IEEE Trans. on Robotics*, 25(3):593–601, 2009. [cited on page(s) 26]
- [JLP⁺15] Steffen Jaekel, Roberto Lampariello, Giogio Panin, Mikel Sagardia, Bernhard Brunner, Oliver Porges, Erich Kraemer, Matthias Wieser, Richard Haarmann, Markus Pietras, and Robin Biesbrock. Robotic capture and de-orbiting of a heavy, uncooperative and tumbling target satellite in low earth orbit. In *Proc. Symp. on Advanced Space Technologies in Robotics and Automation (ASTRA)*, 2015. [cited on page(s) 241]
- [Jon00] Lynette A Jones. Kinesthetic sensing. In *Human and Machine Haptics*. MIT Press, 2000. [cited on page(s) 11]
- [JP01] Doug L. James and Dinesh K. Pai. A unified treatment of elastostatic contact simulation for real time haptics. *Haptics-e: The Electronic Journal of Haptics Research*, 2(1):1–13, 2001. [cited on page(s) 60]

- [JWC05] David E. Johnson, Peter Willemsen, and Elaine Cohen. Six degree-of-freedom haptic rendering using spatialized normal cone search. *IEEE Trans. on Visualization and Computer Graphics*, 11(6):661–670, 2005. [cited on page(s) 33]
- [KDK10] Ingo Kossyk, Jonas Dörr, and Konstantin Kondak. Design and evaluation of a wearable haptic interface for large workspaces. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 4674–4679. IEEE, 2010. [cited on page(s) 18, 19]
- [KEP05] Danny M. Kaufman, Timothy Edmunds, and Dinesh K. Pai. Fast frictional dynamics for rigid bodies. *ACM Trans. on Graphics*, 24(3):946–956, 2005. [cited on page(s) 117]
- [KFN06] Katherine J. Kuchenbecker, Jonathan Fiene, and Günter Niemeyer. Improving contact realism through event-based haptic feedback. *IEEE Trans. on Visualization and Computer Graphics*, 12(2):219–230, 2006. [cited on page(s) 20, 59, 179, 181]
- [KHM⁺98] James T Klosowski, Martin Held, Joseph SB Mitchell, Henry Sowizral, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Trans. on Visualization and Computer Graphics*, 4(1):21–36, 1998. [cited on page(s) 32, 46, 47]
- [KKT⁺04] Jung Kim, Hyun Kim, Boon K Tay, Manivannan Muniyandi, Mandayam A Srinivasan, Joel Jordan, Jesper Mortensen, Manuel Oliveira, and Mel Slater. Transatlantic touch: A study of haptic collaboration over long distance. *Presence: Teleoperators & Virtual Environments*, 13(3):328–337, 2004. [cited on page(s) 26]
- [KLM02] Young J. Kim, Ming C. Lin, and Dinesh Manocha. Deep: Dual-space expansion for estimating penetration depth between convex polytopes. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 921–926. IEEE, 2002. [cited on page(s) 37]
- [KLM04] Young J. Kim, Ming C. Lin, and Dinesh Manocha. Incremental penetration depth estimation between convex polytopes using dual-space expansion. *IEEE Trans. on Visualization and Computer Graphics*, 10(2):152–163, 2004. [cited on page(s) 37]
- [KLR15] Jaeha Kim, Chang-Gyu Lee, and Jeha Ryu. Depth cube-based six degree-of-freedom haptic rendering for rigid bodies. *IEEE Trans. on Haptics*, 8(4):345–355, 2015. [cited on page(s) 33, 34, 42]
- [KOKM11] H. Kawasaki, Y. Ohtuka, S. Koide, and T. Mouri. Perception and haptic rendering of friction moments. *IEEE Trans. on Haptics*, 4(1):28–38, 2011. [cited on page(s) 118, 133]

- [KP09] Kimin Kim and Jinah Park. Virtual bone drilling for dental implant surgery training. In *Proc. ACM Symp. on Virtual Reality and Software Technology (VRST)*, pages 91–94. ACM, 2009. [cited on page(s) 62]
- [KS87] Ariel Kaufman and Eyal Shimony. 3d scan-conversion algorithms for voxel-based graphics. In *Proc. of the Workshop on Interactive 3D Graphics*, pages 45–75. ACM, 1987. [cited on page(s) 71]
- [KWA⁺09] P. Kremer, T. Wimböck, J. Artigas, S. Schätzle, K. Jöhl, F. Schmidt, C. Preusche, and G. Hirzinger. Multimodal telepresent control of DLR’s Rollin’ JUSTIN (video). In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1601–1602. IEEE, 2009. [cited on page(s) 168, 172, 228]
- [KWZ17] Maximilian Kaluschke, Rene Weller, and Gabriel Zachmann. A volumetric penetration measure for 6-dof haptic rendering of streaming point clouds. In *Proc. IEEE World Haptics Conference*, pages 511–516. IEEE, 2017. [cited on page(s) 33, 48]
- [KZ04] Jan Klein and Gabriel Zachmann. Point cloud collision detection. *Computer Graphics Forum*, 23(3):567–576, 2004. [cited on page(s) 33, 42]
- [Law93] Dale A Lawrence. Stability and transparency in bilateral teleoperation. *IEEE Trans. Robotics and Automation*, 9(5):624–637, 1993. [cited on page(s) 23]
- [Law11] Michael A Lawrence. *ez: Easy analysis and visualization of factorial experiments*, 2011. [cited on page(s) 208]
- [LBF⁺02] Ming C Lin, William Baxter, Mark Foskey, Miguel A Otaduy, and Vincent Scheib. Haptic interaction for creative processes with simulated media. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, volume 1, pages 598–604. IEEE, 2002. [cited on page(s) 26]
- [LC87] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proc. ACM SIGGRAPH*, volume 21(4), pages 163–169. ACM, 1987. [cited on page(s) 62]
- [LC91] Ming C. Lin and John F. Canny. A fast algorithm for incremental distance calculation. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1008–1014. IEEE, 1991. [cited on page(s) 32, 37, 41, 48, 52]
- [LCS12] Adam Leeper, Sonny Chan, and Kenneth Salisbury. Point clouds can be represented as implicit surfaces for constraint-based haptic rendering. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 5000–5005. IEEE, 2012. [cited on page(s) 33, 34, 42, 117]
- [Leo07] Francisco Leon. Gimpact - geometric tools for vr, 2007. Web page accessed on August 29th, 2014. [cited on page(s) 146]

- [LGLM99] Eric Larsen, Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. Fast proximity queries with swept sphere volumes. Technical report, University of North Carolina, Chapel Hill, 1999. [cited on page(s) 32, 46, 48]
- [LJ00] Joseph J LaViola Jr. A discussion of cybersickness in virtual environments. *ACM SIGCHI Bulletin*, 32(1):47–56, 2000. [cited on page(s) 7, 13]
- [LK87] Susan J Lederman and Roberta L Klatzky. Hand movements: A window into haptic object recognition. *Cognitive psychology*, 19(3):342–368, 1987. [cited on page(s) 11]
- [LK09] S.J. Lederman and R.L. Klatzky. Haptic perception: A tutorial. *Attention, Perception, & Psychophysics*, 71(7):1439–1459, 2009. [cited on page(s) 7, 8, 9, 10, 11, 176]
- [LKG⁺94] Donald R Lampton, Bruce W Knerr, Stephen L Goldberg, James P Bliss, J Michael Moshell, and Brian S Blau. The virtual environment performance assessment battery (vepab): Development and evaluation. *Presence: Teleoperators and Virtual Environments*, 3(2):145–157, 1994. [cited on page(s) 183]
- [LM04] Ming C. Lin and Dinesh Manocha. *Handbook of discrete and computational geometry*, chapter Collision and Proximity Queries (Book chapter). CRC press, 2004. [cited on page(s) 29, 39]
- [LMM10] Christian Lauterbach, Qi Mo, and Dinesh Manocha. gproximity: hierarchical gpu-based operations for collision and distance queries. *Computer Graphics Forum*, 29(2):419–428, 2010. [cited on page(s) 49]
- [LO08] Min C. Lin and Miguel A. Otaduy, editors. *Haptic Rendering: Foundations, Algorithms and Applications*. A K Peters, Ltd., 2008. [cited on page(s) 26, 29]
- [LRD⁺07] Theodore Lim, James M Ritchie, Richard G Dewar, Jonathan R Corney, P Wilkinson, Mustafa Calis, M Desmulliez, and J-J Fang. Factors affecting user performance in haptic assembly. *Virtual Reality*, 11(4):241–252, 2007. [cited on page(s) 178, 218]
- [LS08] Simon Lacey and K. Sathian. *Human Haptic Perception*, chapter Haptically Evoked Activation of Visual Cortex, pages 251–257. Birkhaeuser Verlag, 2008. [cited on page(s) 12]
- [LYFH15] Keyan Liu, Xuyue Yin, Xiumin Fan, and Qichang He. Virtual assembly with physical information: a review. *Assembly Automation*, 35(3):206–220, 2015. [cited on page(s) 144]
- [LYG02] Karljohan Lundin, Anders Ynnerman, and Björn Gudmundsson. Proxy-based haptic feedback from volumetric density data. In *Proc. Eurohaptics Conf.*, pages 104–109, 2002. [cited on page(s) 55]

- [Mas03] Takeshi Masuda. Surface curvature estimation from the signed distance field. In *Proc. Int. Conf. on 3-D Digital Imaging and Modeling (3DIM)*, pages 361–368. IEEE, 2003. [cited on page(s) 88]
- [MG09] Khaled Mamou and Faouzi Ghorbel. A simple and efficient approach for 3d mesh approximate convex decomposition. In *IEEE Int. Conf. on Image Processing (ICIP)*, pages 3501–3504. IEEE, 2009. [cited on page(s) 33, 100, 146, 167]
- [Mir96] Brian Vincent Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California at Berkeley, 1996. [cited on page(s) 50, 52, 57]
- [Mir98] Brian Mirtich. V-clip: Fast and robust polyhedral collision detection. *ACM Trans. on Graphics*, 17(3):177–208, 1998. [cited on page(s) 41]
- [Mir00] Brian Mirtich. Timewarp rigid body simulation. In *Proc. ACM SIGGRAPH*, pages 193–200. ACM, 2000. [cited on page(s) 43, 44, 45, 48]
- [MN08] Probal Mitra and Günter Niemeyer. Model-mediated telemanipulation. *Int. J. Robotics Research*, 27(2):253–262, 2008. [cited on page(s) 24]
- [Moe97] Tomas Moeller. A fast triangle-triangle intersection test. *Journal of Graphics Tools*, 2(2):25–30, 1997. [cited on page(s) 38]
- [Mou15] Konstantinos Moustakas. 6dof haptic rendering using distance maps over implicit representations. *Multimedia Tools and Applications*, pages 1–15, 2015. [cited on page(s) 42, 59]
- [MPT99] William A. McNeely, Kevin D. Puterbaugh, and James J. Troy. Six degree-of-freedom haptic rendering using voxel sampling. In *Proc. ACM SIGGRAPH*, pages 401–408. ACM, 1999. [cited on page(s) 4, 5, 33, 34, 42, 50, 53, 57, 60, 65, 66, 75, 94, 106, 112, 122, 224]
- [MPT06] William A. McNeely, Kevin D. Puterbaugh, and James J. Troy. Voxel-based 6-dof haptic rendering improvements. *Haptics-e: The Electronic Journal of Haptics Research*, 3(7):1–12, 2006. [cited on page(s) 42, 48, 66, 103, 151]
- [MS⁺94] Thomas H Massie, J Kenneth Salisbury, et al. The phantom haptic interface: A device for probing virtual objects. In *Proc. of the ASME Symp. on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, volume 55:1, pages 295–300. ASME, 1994. [cited on page(s) 18, 20, 235]
- [MS96] Hugh B. Morgenbesser and Mandayam A. Srinivasan. Force shading for haptic shape perception. In *Proc. of the ASME Dynamics Systems and Control Division*, volume 58, pages 407–412, 1996. [cited on page(s) 58]
- [MW88] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. *Computer Graphics*, 22(4):289–298, 1988. [cited on page(s) 44, 53]

- [NB04] Wolfgang Neuwirth and Michael Benesch. Motorische leistungsserie. Technical report, Dr. G. Schuhfried GmbH, 2004. [cited on page(s) 183]
- [NJC99] Donald D. Nelson, David E. Johnson, and Elaine Cohen. *Haptic Rendering Of Surface-To-Surface Sculpted Model Interaction*, volume 67, pages 101–108. ASME, 1999. [cited on page(s) 43]
- [NMK⁺06] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. *Computer graphics forum*, 25(4):809–836, 2006. [cited on page(s) 59]
- [NPH08] Guenter Niemeyer, Carsten Preusche, and Gerd Hirzinger. *Springer Handbook on Robotics*, chapter Telerobotics, pages 741–757. Springer, 2008. [cited on page(s) 23, 25]
- [NSSB16] Korbinian Nottensteiner, Mikel Sagardia, Andreas Stemmer, and Christoph Borst. Narrow passage sampling in the observation of robotic assembly tasks. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 130–137. IEEE, 2016. [cited on page(s) 67, 171, 242]
- [Nvi01] Nvidia. Physx library, 2001. Web page accessed on February 1, 2018. [cited on page(s) 15]
- [OG02] Marcia O’Malley and Michael Goldfarb. The effect of force saturation on the haptic perception of detail. *IEEE/ASME Transactions on Mechatronics*, 7(3):280–288, 2002. [cited on page(s) 11]
- [OG04] Marcia K O’Malley and Michael Goldfarb. The effect of virtual surface stiffness on the haptic perception of detail. *IEEE/ASME Transactions on Mechatronics*, 9(2):448–454, 2004. [cited on page(s) 11, 179, 200]
- [OGL13] Miguel A Otaduy, Carlos Garre, and Ming C Lin. Representations and algorithms for force-feedback display. *Proc. of the IEEE*, 101(9):2068–2080, 2013. [cited on page(s) 29, 51]
- [OL05] Miguel A. Otaduy and Ming C. Lin. Sensation preserving simplification for haptic rendering. In *ACM SIGGRAPH 2005 Courses*. ACM, 2005. [cited on page(s) 47]
- [OL06] Miguel A. Otaduy and Ming C. Lin. A modular haptic rendering algorithm for stable and transparent 6-dof manipulation. *IEEE Trans. on Robotics*, 22(4):751–762, 2006. [cited on page(s) 15, 50, 56]
- [ORC07] Michael Ortega, Stephane Redon, and Sabine Coquillart. A six degree-of-freedom god-object method for haptic display of rigid bodies with surface properties. *IEEE Trans. on Visualization and Computer Graphics*, 13(3):458–469, 2007. [cited on page(s) 44, 50, 54, 57, 59, 67, 116, 117, 134, 142]

- [Ott05] Martin Otterbach. Software-werkzeug zur diskretisierung virtueller 3d-objekte. Master's thesis, German Aerospace Center (DLR) - Fachhochschule Esslingen, 2005. [cited on page(s) 71]
- [PG12] Uwe Proske and Simon C Gandevia. The proprioceptive senses: their roles in signaling body shape, body position and movement, and muscle force. *Physiological Reviews*, 92(4):1651–1697, 2012. [cited on page(s) 8]
- [Pho75] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975. [cited on page(s) 58]
- [PM11] Jia Pan and Dinesh Manocha. Gpu-based parallel collision detection for fast motion planning. *The International Journal of Robotics Research*, page 0278364911429335, 2011. [cited on page(s) 25, 44, 48]
- [PND⁺11] I. Peterlík, M. Nouicer, C. Duriez, S. Cotin, and A. Kheddar. Constraint-based haptic rendering of multirate compliant mechanisms. *IEEE Trans. on Haptics*, 4(3):175–187, 2011. [cited on page(s) 36, 60]
- [PPM17] Jae Sung Park, Chonhyon Park, and Dinesh Manocha. Efficient probabilistic collision detection for non-convex shapes. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1944–1951. IEEE, 2017. [cited on page(s) 49]
- [PSCM13] Jia Pan, Ioan A Sutan, Subhashini Chitta, and Dinesh Manocha. Real-time collision detection and distance computation on point cloud sensor data. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3593–3599. IEEE, 2013. [cited on page(s) 34]
- [PV11] Ryan A Pavlik and Judy M Vance. Expanding haptic workspace for coupled-object manipulation. In *Proc. World Conf. on Innovative Virtual Reality (ASME)*, pages 293–299. American Society of Mechanical Engineers (ASME), 2011. [cited on page(s) 156]
- [PV14] Jérôme Perret and Pierre Vercruyse. Advantages of mechanical backdrivability for medical applications of force control. In *Workshop on Computer/Robot Assisted Surgery (CRAS)*, 2014. [cited on page(s) 19]
- [PWBI97] Ivan Poupyrev, Suzanne Weghorst, Mark Billingham, and Tadao Ichikawa. A framework and testbed for studying manipulation techniques for immersive vr. In *Proc. ACM Symp. on Virtual Reality and Software Technology (VRST)*, pages 21–28. ACM, 1997. [cited on page(s) 183]
- [PZFea04] B. Petzold, M. F. Zaeh, B. Faerber, and B. Deml et al. A study on visual, auditory and haptic feedback for assembly tasks. *Presence: Teleoperators and Virtual Environments*, 13(1):16–21, 2004. [cited on page(s) 14]

- [PZM13] Jia Pan, Xinyu Zhang, and Dinesh Manocha. Efficient penetration depth approximation using active learning. *ACM Trans. on Graphics*, 32(6), 2013. [cited on page(s) 36, 49]
- [Qui94] Sean Quinlan. Efficient distance computation between non-convex objects. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3324–3329. IEEE, 1994. [cited on page(s) 32, 33, 46]
- [RA93] Louis B Rosenberg and Bernard D Adelstein. Perceptual decomposition of virtual haptic surfaces. In *Proc. IEEE Virtual Reality (VR)*, pages 46–53. IEEE, 1993. [cited on page(s) 10, 176, 178, 181]
- [RC11a] Hugh Reynolds and Steven Collins. Havok physics engine, 2011. Web page accessed on February 1, 2018. [cited on page(s) 15]
- [RC11b] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011. [cited on page(s) 33]
- [RC13a] Fredrik Ryden and Howard J Chizeck. A proxy method for real-time 3-dof haptic rendering of streaming point cloud data. *IEEE Trans. on Haptics*, 6(3):257–267, 2013. [cited on page(s) 25, 33]
- [RC13b] Fredrik Rydén and Howard Jay Chizeck. A method for constraint-based six degree-of-freedom haptic interaction with streaming point clouds. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2353–2359. IEEE, 2013. [cited on page(s) 33, 42, 67, 117]
- [RdlTH01] Gabriel Robles-de-la Torre and Vincent Hayward. Force can overcome object geometry in the perception of shape through active touch. *Nature*, 412(6845):445–448, 2001. [cited on page(s) 12]
- [RHBH11] Maximo A. Roa, Katharina Hertkorn, Christoph Borst, and Gerd Hirzinger. Reachable independent contact regions for precision grasps. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 5337–5343. IEEE, 2011. [cited on page(s) 172]
- [Ric14] Richard Smith and C++ ISO Board. Standard for programming language c++ [working draft], 2014. [cited on page(s) 111, 231, 232]
- [RKC02a] Stéphane Redon, Abderrahmane Kheddar, and Sabine Coquillart. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum*, 21(3):279–287, 2002. [cited on page(s) 43, 55, 116]
- [RKC02b] Stéphane Redon, Abderrahmane Kheddar, and Sabine Coquillart. Gauss’ least constraints principle and rigid body simulations. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 517–522, 2002. [cited on page(s) 55, 116]

- [RKK97] Diego C. Ruspini, Krasimir Kolarov, and Oussama Khatib. The haptic display of complex graphical environments. In *Proc. ACM SIGGRAPH*, pages 345–352. ACM, 1997. [cited on page(s) 54, 59, 116]
- [RLB10] Silvio H Rizzi, Cristian J Luciano, and P Pat Banerjee. Haptic interaction with volumetric datasets using surface-based haptic libraries. In *Proc. IEEE Haptics Symposium (HAPTICS)*, pages 243–250. IEEE, 2010. [cited on page(s) 44]
- [RLB12] Silvio H. Rizzi, Cristian J. Luciano, and P. Pat Banerjee. Comparison of algorithms for haptic interaction with isosurfaces extracted from volumetric datasets. *Journal of computing and information science in engineering*, 12(2):021004 1–10, 2012. [cited on page(s) 177]
- [RMB⁺08] Emanuele Ruffaldi, Dan Morris, Federico Barbagli, Ken Salisbury, and Massimo Bergamasco. Voxel-based haptic rendering using implicit sphere trees. In *Proc. IEEE Haptics Symposium (HAPTICS)*, pages 319–325. IEEE, 2008. [cited on page(s) 34, 52]
- [Ros93] Louis B. Rosenberg. Virtual fixtures: Perceptual tools for telerobotic manipulation. In *Proc. IEEE Virtual Reality (VR)*, pages 76–82. IEEE, 1993. [cited on page(s) 24]
- [RPP⁺01] Matthias Renz, Carsten Preusche, Marco Pötke, Hans-Peter Kriegel, and Gerd Hirzinger. Stable haptic interaction with virtual environments using an adapted voxmap-pointshell algorithm. In *Proc. Eurohaptics Conf.*, 2001. [cited on page(s) 42, 56, 66]
- [S⁺03] James A Sethian et al. Level set methods and fast marching methods. *Journal of Computing and Information Technology*, 11(1):1–2, 2003. [cited on page(s) 34]
- [Sag08] Mikel Sagardia. Enhancements of the voxmap-pointshell algorithm. Master’s thesis, Tecnum – University de Navarra and German Aerospace Center (DLR), 2008. [cited on page(s) 4, 70, 72]
- [SBB96] Mandayam A Srinivasan, Gerald Lee Beauregard, and David L Brock. The impact of visual information on the haptic perception of stiffness in virtual environments. In *Proc. ASME Dynamics Systems and Control Division*, volume 58, pages 555–559, 1996. [cited on page(s) 12]
- [SBCC17] Florian Schmidt, Robert Burger, Jan Cremer, and Maxime Chalon. Links and nodes, 2017. Find correct link. [cited on page(s) 154]
- [SBG⁺07] Jean Sreng, Florian Bergez, Jérémie Le Garrec, Anatole Lécuyer, and Claude Andriot. Using an event-based approach to improve the multimodal rendering of 6dof virtual contact. In *Proc. ACM Symp. on Virtual Reality and Software Technology (VRST)*, pages 165–173. ACM, 2007. [cited on page(s) 14, 15]

- [SC12] KG Sreeni and Subhasis Chaudhuri. Haptic rendering of dense 3d point cloud data. In *Proc. IEEE Haptics Symposium (HAPTICS)*, pages 333–339. IEEE, 2012. [cited on page(s) 34]
- [SCB04] Keneth Salisbury, Francois Conti, and Federico Barbagli. Haptic rendering: Introductory concepts. *IEEE Computer Graphics and Applications*, 24(2):24–32, 2004. [cited on page(s) 156]
- [Sch15] Anja Katharina Schneider. Evaluation of haptic human-machine interfaces for virtual reality applications. Master’s thesis, Technische Universität München and German Aerospace Center (DLR), 2015. Supervised by Bernhard Weber and Mikel Sagardia. [cited on page(s) 185, 227]
- [SEWP10] S. Schätzle, T. Ende, T. Wuesthoff, and C. Preusche. VibroTac: an ergonomic and versatile usable vibrotactile feedback device. In *Proc. IEEE Int. Symp. on Robots and Human Interactive Communications (ROMAN)*, pages 705–710, 2010. [cited on page(s) 16, 155]
- [SH13] Mikel Sagardia and Thomas Hulin. Fast and accurate distance, penetration, and collision queries using point-sphere trees and distance fields. In *ACM SIGGRAPH Posters*, page 83. ACM, 2013. [cited on page(s) 65, 240]
- [SH16] Mikel Sagardia and Thomas Hulin. A fast and robust six-dof god object heuristic for haptic rendering of complex models with friction. In *Proc. ACM Symp. on Virtual Reality and Software Technology (VRST)*, pages 163–172. ACM, 2016. [cited on page(s) 115, 151, 242]
- [SH17a] Mikel Sagardia and Thomas Hulin. Evaluation of a penalty and a constraint-based haptic rendering algorithm with different haptic interfaces and stiffness values. In *Proc. IEEE Virtual Reality (VR)*, pages 64–73. IEEE, 2017. [cited on page(s) 176, 242]
- [SH17b] Mikel Sagardia and Thomas Hulin. Multimodal evaluation of the differences between real and virtual assemblies. *IEEE Trans. on Haptics*, 11:107–118, 2017. [cited on page(s) 176, 242]
- [SHE⁺12] Eckehard Steinbach, Sandra Hirche, Marc Ernst, Fernanda Brandi, Rahul Chaudhari, Julius Kammerl, and Iason Vittorias. Haptic communications. *Proceedings of the IEEE*, 100(4):937–956, 2012. [cited on page(s) 25]
- [SHGC14] Mikel Sagardia, Katharina Hertkorn, David Sierra González, and Claudio Castellini. Ultrapiano: A novel human-machine interface applied to virtual reality (video). In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2089–2089. IEEE, 2014. [cited on page(s) 170, 241]

- [SHH⁺13] Mikel Sagardia, Katharina Hertkorn, Thomas Hulin, Robin Wolff, Johannes Hummel, Janki Dodiya, and Andreas Gerndt. An interactive virtual reality system for on-orbit servicing. In *Proc. IEEE Virtual Reality (VR)*, 2013. (Video). [cited on page(s) 168, 240]
- [SHH⁺15] Mikel Sagardia, Katharina Hertkorn, Thomas Hulin, Simon Schätzle, Robin Wolff, Johannes Hummel, Janki Dodiya, and Andreas Gerndt. VR-OOS: The DLR's virtual reality simulator for telerobotic on-orbit servicing with haptic feedback. In *Proc. IEEE Aerospace Conf.*, pages 1–17, 2015. [cited on page(s) 65, 145, 159, 170, 221, 241]
- [SHH⁺16] Mikel Sagardia, Thomas Hulin, Katharina Hertkorn, Philipp Kremer, and Simon Schätzle. A platform for bimanual virtual assembly training with haptic feedback in large multi-object environments. In *Proc. ACM Symp. on Virtual Reality and Software Technology (VRST)*, pages 153–162. ACM, 2016. [cited on page(s) 143, 221, 242]
- [Shi93] Karun B Shimoga. A survey of perceptual feedback issues in dexterous telemanipulation: Part i. finger force feedback. In *Virtual Reality Annual International Symposium (VRAIS)*, pages 263–270. IEEE, 1993. [cited on page(s) 11]
- [SHPH08] Mikel Sagardia, Thomas Hulin, Carsten Preusche, and Gerd Hirzinger. Improvements of the voxmap-pointshell algorithm – fast generation of haptic data-structures. In *Internationales Wissenschaftliches Kolloquium (IWK, TU Ilmenau)*, 2008. [cited on page(s) 65, 72, 75, 81, 93, 239]
- [SHPH09] Mikel Sagardia, Thomas Hulin, Carsten Preusche, and Gerd Hirzinger. A benchmark of force quality in haptic rendering. In *Proc. of the Int. Conf. on Human-Computer Interaction (HCI)*, 2009. [cited on page(s) 4, 239]
- [SIT89] PH Sutter, JC Iatridis, and NV Thakor. Response to reflected-force feedback to fingers in teleoperations. In *Proc. of the NASA Conf. on Space Telerobotics*, 1989. [cited on page(s) 20]
- [Sla03] Mel Slater. A note on presence terminology. *Presence connect*, 3(3):1–5, 2003. [cited on page(s) 14]
- [SLY99] Chuan-Jun Su, Fuhua Lin, and Lan Ye. A new collision detection method for csg-represented objects in virtual manufacturing. *Computers in industry*, 40(1):1–13, 1999. [cited on page(s) 35]
- [Smi01] Russell Smith. Open dynamics engine library 0.13, 2001. Web page accessed on February 1, 2018. [cited on page(s) 15]
- [SMK98] Kay M Stanney, Ronald R Mourant, and Robert S Kennedy. Human factors issues in virtual environments: A review of the literature. *Presence: Teleoperators and Virtual Environments*, 7(4):327–351, 1998. [cited on page(s) 13, 15]

- [SNF14] Tanner Schmidt, Richard A Newcombe, and Dieter Fox. Dart: Dense articulated real-time tracking. In *Robotics: Science and Systems*, 2014. [cited on page(s) 67]
- [SR06] Axel Seugling and Martin Rölin. Evaluation of physics engines and implementation of a physics module in a 3d-authoring tool. Master’s thesis, Umea University, 2006. [cited on page(s) 146]
- [SSB13] Fun Shing Sin, Daniel Schroeder, and J Barbič. Vega: Non-linear fem deformable object simulator. *Computer Graphics Forum*, 32(1):36–48, 2013. [cited on page(s) 36, 60, 61]
- [SSeS14a] Mikel Sagardia, Theodoros Stouraitis, and João Lopes e Silva. A New Fast and Robust Collision Detection and Force Computation Algorithm Applied to the Physics Engine Bullet: Method, Integration, and Evaluation. In *Prof. of the Conf. and Exhibition of the European Association of Virtual and Augmented Reality (EuroVR)*, pages 65–76. Eurographics Association, 2014. [cited on page(s) 42, 65, 143, 241]
- [SSeS14b] Mikel Sagardia, Theodoros Stouraitis, and João Lopes e Silva. Poster: Integration of a haptic rendering algorithm based on voxelized and point-sampled structures into the physics engine bullet. In *Proc. IEEE Symposium on 3D User Interfaces (3DUI)*, pages 133–134. IEEE, 2014. [cited on page(s) 143, 241]
- [SSV06] Abhishek Seth, Hai-Jun Su, and Judy M Vance. Sharp: a system for haptic assembly and realistic prototyping. In *Proc. ASME Int. Design Engineering Technical Conf. and Computers and Information in Engineering Conf.*, pages 905–912. American Society of Mechanical Engineers (ASME), 2006. [cited on page(s) 145]
- [ST97] Kenneth Salisbury and Christopher Tarr. Haptic rendering of surfaces defined by implicit functions. In *Proc. Annual ASME Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, volume 61, pages 61–67, 1997. [cited on page(s) 42, 55, 58, 117, 118, 133]
- [Sta97] Richard Eugene Stamper. *A Three Degree of Freedom Parallel Manipulator with Only Translational Degrees of Freedom*. PhD thesis, University of Maryland, 1997. [cited on page(s) 18, 19, 26]
- [STH18] Mikel Sagardia, Alexander Martín Turrillas, and Thomas Hulin. Realtime collision avoidance for mechanisms with complex geometries. In *Proc. IEEE Virtual Reality (VR)*, 2018. (Video). [cited on page(s) 171, 242]
- [SVO11] Abhishek Seth, Judy M Vance, and James H Oliver. Virtual reality for assembly methods prototyping: a review. *Virtual Reality*, 15(1):5–20, 2011. [cited on page(s) 26, 144]

- [SWH⁺12] Mikel Sagardia, Bernhard Weber, Thomas Hulin, Carsten Preusche, and Gerd Hirzinger. Evaluation of visual and force feedback in virtual assembly verifications. In *Proc. IEEE Virtual Reality (VR)*, pages 23–26. IEEE, 2012. [cited on page(s) 4, 14, 175, 240]
- [SWSB07] Evren Samur, Fei Wang, Ulrich Spaelter, and Hannes Bleuler. Generic and systematic evaluation of haptic interfaces based on testbeds. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 2113–2119. IEEE, 2007. [cited on page(s) 178, 183]
- [TDP10] Loïc Tching, Georges Dumont, and Jérôme Perret. Interactive simulation of cad models assemblies using virtual constraint guidance. *Int. J. on Interactive Design and Manufacturing (IJIDeM)*, 4(2):95–102, 2010. [cited on page(s) 24]
- [THH⁺11] Andreas Tobergte, Patrick Helmer, Ulrich Hagn, Patrice Rouiller, Sophie Thielmann, Sébastien Grange, Alin Albu-Schäffer, François Conti, and Gerd Hirzinger. The sigma. 7 haptic interface for mirosurge: A new bi-manual surgical console. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 3023–3030. IEEE, 2011. [cited on page(s) 6, 18, 19, 179, 184, 229]
- [TIHS⁺01] Russell M Taylor II, Thomas C Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, and Aron T Helser. Vrpn: a device-independent, network-transparent vr peripheral system. In *Proc. ACM Symp. on Virtual Reality and Software Technology (VRST)*, pages 55–61. ACM, 2001. [cited on page(s) 16]
- [TJC97] Thomas Thompson, David E Johnson, and Elaine Cohen. Direct haptic rendering of sculptured models. In *Proc. of the Symposium on Interactive 3D Graphics*, pages 167–176. ACM, 1997. [cited on page(s) 35, 42, 43]
- [TKM09] Min Tang, Young J. Kim, and Dinesh Manocha. C2a: Controlled conservative advancement for continuous collision detection of polygonal models. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 849–854. IEEE, 2009. [cited on page(s) 43]
- [TKZ⁺05] M. Teschner, S. Kimmerle, G. Zachmann, B. Heidelberger, Laks Raghupathi, A. Fuhrmann, Marie-Paule Cani, François Faure, N. Magnetat-Thalmann, and W. Strasser. Collision detection for deformable objects. *Computer Graphics Forum*, 24(1):61–81, 2005. [cited on page(s) 59]
- [TML14] Anthony Talvas, Maud Marchal, and Anatole Lecuyer. A survey on bimanual haptic interaction. *IEEE Trans. on Haptics*, 7(3):285–300, 2014. [cited on page(s) 19]
- [TMOT12] Min Tang, Dinesh Manocha, Miguel A Otaduy, and Ruofeng Tong. Continuous penalty forces. *ACM Trans. on Graphics*, 31(4):107–1, 2012. [cited on page(s) 38, 43, 53]

- [TRC⁺93] Russell M Taylor, Warren Robinett, Vernon L Chi, Frederick P Brooks Jr, William V Wright, R Stanley Williams, and Erik J Snyder. The nanomanipulator: a virtual-reality interface for a scanning tunneling microscope. In *Proc. ACM SIGGRAPH*, pages 127–134. ACM, 1993. [cited on page(s) 26]
- [TSEC94] Hong Z Tan, Mandayam A Srinivasan, Brian Eberman, and Belinda Cheng. Human factors for the design of force-reflecting haptic interfaces. *Dynamic Systems and Control*, 55(1):353–359, 1994. [cited on page(s) 10, 11, 20, 185]
- [UNT⁺02] B. J. Unger, A. Nicolaidis, A. Thompson, R. L. Klatzky, R. L. Hollis, P. J. Berkelman, and S. Lederman. Virtual peg-in-hole performance using a 6-dof magnetic levitation haptic device: Comparison with real forces and with visual guidance alone. In *Proc. IEEE Haptics Symposium (HAPTICS)*, pages 263–270. IEEE, 2002. [cited on page(s) 178, 218]
- [vdB97] Gino van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools*, 2(4):1–13, 1997. [cited on page(s) 32, 39, 46, 49]
- [vdB99] Gino van den Bergen. A fast and robust gjk implementation for collision detection of convex objects. *Journal of Graphics Tools*, 4(2):7–25, 1999. [cited on page(s) 35, 40]
- [vdB01] Gino van den Bergen. Proximity queries and penetration depth computation on 3d game objects. In *Game Developers Conference*, volume 170, 2001. [cited on page(s) 40, 146]
- [VO09] Lawton N Verner and Allison M Okamura. Force & torque feedback vs force only feedback. In *Proc. IEEE World Haptics Conference*, pages 406–410. IEEE, 2009. [cited on page(s) 57]
- [WAH⁺13] J.G.W. Wildenbeest, D.A. Abbink, C.J.M. Heemskerk, F.C.T. van der Helm, and H. Boessenkool. The impact of haptic feedback quality on the performance of teleoperated assembly tasks. *IEEE Trans. on Haptics*, 6(2):242–252, 2013. [cited on page(s) 183]
- [WCW⁺12] Qiong Wang, Hui Chen, Wen Wu, Jing Qin, and Pheng Ann Heng. Impulse-based rendering methods for haptic simulation of bone-burring. *IEEE Trans. on Haptics*, 5(4):344–355, 2012. [cited on page(s) 52, 59, 61]
- [WGP⁺04] Huagen Wan, Shuming Gao, Qunsheng Peng, Guozhong Dai, and Fengjun Zhang. Mivas: a multi-modal immersive virtual assembly system. In *Proc. Int. Design Engineering Technical Conf. (ASME)*, pages 113–122. American Society of Mechanical Engineers (ASME), 2004. [cited on page(s) 144]

- [WHTL13] Bernhard Weber, Anja Hellings, Andreas Tobergte, and Martin Lohmann. Human performance and workload evaluation of input modalities for telesurgery. In *Proceedings of the German Society of Ergonomics (GfA) Spring Congress*, 2013. [cited on page(s) 229]
- [Wil76] R. Wilhelmsen. A nearest point algorithm for convex polyhedral cones and applications to positive linear approximation. *Mathematics of computation*, 30(133):48–57, 1976. [cited on page(s) 55, 135]
- [WKH11] Bing Wu, R.L. Klatzky, and R.L. Hollis. Force, torque, and stiffness: Interactions in perceptual discrimination. *IEEE Trans. on Haptics*, 4(3):221 – 228, 2011. [cited on page(s) 57]
- [WM03] Ming Wan and William A McNeely. Quasi-static approximation for 6 degrees-of-freedom haptic rendering. In *Proc. IEEE Visualization (VIS)*, pages 257 – 262. IEEE, 2003. [cited on page(s) 56, 66]
- [WS98] Bob G. Witmer and Michael J. Singer. Measuring presence in virtual environments: A presence questionnaire. *Presence: Teleoperators and Virtual Environments*, 7(3):225–240, 1998. [cited on page(s) 14]
- [WSHP13] Bernhard Weber, Mikel Sagardia, Thomas Hulin, and Carsten Preusche. Visual, vibrotactile, and force feedback of collisions in virtual environments: effects on performance, mental workload and spatial orientation. In *Virtual Augmented and Mixed Reality. Designing and Developing Augmented and Virtual Environments*, volume 8021 of *Lecture Notes in Computer Science*, pages 241–250. Springer Berlin Heidelberg, 2013. [cited on page(s) 4, 14, 175, 240]
- [WSM⁺10] Rene Weller, Mikel Sagardia, David Mainzer, Thomas Hulin, Gabriel Zachmann, and Carsten Preusche. A benchmarking suite for 6-dof real time collision response algorithms. In *Proc. ACM Symp. on Virtual Reality and Software Technology (VRST)*, pages 63–70. ACM, 2010. [cited on page(s) 4, 78, 177, 239]
- [WYW⁺09] Jun Wu, Ge Yu, Dangxiao Wang, Yuru Zhang, and Charlie CL Wang. Voxel-based interactive haptic simulation of dental drilling. In *Int. Design Engineering Technical Conf. and Computers and Information in Engineering Conf. (ASME)*, pages 39–48. American Society of Mechanical Engineers (ASME), 2009. [cited on page(s) 62]
- [WZ09a] Rene Weller and Gabriel Zachmann. Inner sphere trees for proximity and penetration queries. In *Proc. Robotics Science and Systems (RSS)*, volume 2, 2009. [cited on page(s) 33, 38, 53]
- [WZ09b] Rene Weller and Gabriel Zachmann. A unified approach for physically-based simulations and haptic rendering. In *Proc. ACM SIGGRAPH Symp. on Video Games*, pages 151–159. ACM, 2009. [cited on page(s) 46, 92]

- [WZ12] Rene Weller and Gabriel Zachmann. User performance in complex bi-manual haptic manipulation with 3 dofs vs. 6 dofs. In *Proc. IEEE Haptics Symposium (HAPTICS)*, pages 315–322. IEEE, 2012. [cited on page(s) 57]
- [WZZX11] Dangxiao Wang, Xin Zhang, Yuru Zhang, and Jing Xiao. Configuration-based optimization for six degree-of-freedom haptic rendering using sphere-trees. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 906–912. IEEE, 2011. [cited on page(s) 51]
- [WZZX13] Dangxiao Wang, Xin Zhang, Yuru Zhang, and Jing Xiao. Configuration-based optimization for six degree-of-freedom haptic rendering for fine manipulation. *IEEE Trans. on Haptics*, 6(2):167–180, 2013. [cited on page(s) 51, 117]
- [XB14] Hongyi Xu and Jernej Barbič. Signed distance fields for polygon soup meshes. In *Proc. of the Graphics Interface Conf.*, pages 35–41. Canadian Information Processing Society, 2014. [cited on page(s) 34, 67]
- [XB16] Hongyi Xu and Jernej Barbic. Adaptive 6-dof haptic contact stiffness using the gauss map. *IEEE Trans. on Haptics*, 9(3):323–332, 2016. [cited on page(s) 54, 99]
- [XB17] Hongyi Xu and Jernej Barbic. 6-dof haptic rendering using continuous collision detection between points and signed distance fields. *IEEE Trans. on Haptics*, 10(2):151–161, 2017. [cited on page(s) 44]
- [XCANS14] Xiao Xu, Burak Cizmeci, Anas Al-Nuaimi, and Eckehard Steinbach. Point cloud-based model-mediated teleoperation with dynamic and perception-based model updating. *IEEE Transactions on Instrumentation and Measurement*, 63(11):2558–2569, 2014. [cited on page(s) 24, 34]
- [Xia16] P. Xia. Haptics for product design and manufacturing simulation. *IEEE Trans. on Haptics*, 9(3):358–375, 2016. [cited on page(s) 26, 176]
- [XLR11] Pinjun Xia, António Lopes, and Maria Restivo. Design and implementation of a haptic-based virtual assembly system. *Assembly Automation*, 31(4):369–384, 2011. [cited on page(s) 145]
- [YKY03] Tsuneo Yoshikawa, Masayuki Kawai, and Kouki Yoshimoto. Toward observation of human assembly skill using virtual task space. In *Experimental Robotics VIII*, pages 540–549. Springer, 2003. [cited on page(s) 178]
- [YWZX15] Ge Yu, Dangxiao Wang, Yuru Zhang, and Jing Xiao. Simulating sharp geometric features in six degrees-of-freedom haptic rendering. *IEEE Trans. on Haptics*, 8(1):67–78, 2015. [cited on page(s) 51]
- [Zac98] Gabriel Zachmann. Rapid collision detection by dynamically aligned dop-trees. In *Proc. IEEE Virtual Reality Annual Int. Symp.*, pages 90–97. IEEE, 1998. [cited on page(s) 32, 46]

- [Zac00] Gabriel Zachmann. *Virtual Reality in Assembly Simulation – Collision Detection, Simulation Algorithms and Interaction Techniques*. PhD thesis, TU Darmstadt, 2000. [cited on page(s) 39]
- [Zac01] Gabriel Zachmann. Optimizing the collision detection pipeline. In *Proc. of the Int. Game Technology Conf. (GTEC)*, 2001. [cited on page(s) 30]
- [Zei93] M Zeiller. Collision detection for objects modelled by csg. *WIT Trans. on Information and Communication Technologies*, 5, 1993. [cited on page(s) 35]
- [ZHHH10] F. Zacharias, I. S. Howard, T. Hulin, and G. Hirzinger. Workspace comparisons of setup configurations for human-robot interaction. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 3117–3122. IEEE, 2010. [cited on page(s) 228]
- [ZK12] Xinyu Zhang and Young J Kim. k-ios: Intersection of spheres for efficient proximity query. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 354–359. IEEE, 2012. [cited on page(s) 33, 37]
- [ZKM07] Liangjun Zhang, Young J Kim, and Dinesh Manocha. A fast and practical algorithm for generalized penetration depth computation. In *Proc. Robotics Science and Systems (RSS)*, 2007. [cited on page(s) 35, 38]
- [ZKM14] Xinyu Zhang, Young J Kim, and Dinesh Manocha. Continuous penetration depth. *Computer-Aided Design*, 46:3–13, 2014. [cited on page(s) 36, 37]
- [ZLK06] Xinyu Zhang, Minkyung Lee, and Young J. Kim. Interactive continuous collision detection for non-convex polyhedra. *The Visual Computer: International Journal of Computer Graphics*, 22(9-11):749–760, 2006. [cited on page(s) 43]
- [ZLSWF13] Fei Zheng, Wen Feng Lu, Yoke San Wong, and Kelvin Weng Chiong Foong. Graphic processing units (gpu)-based haptic simulator for dental implant surgery. *Journal of Computing and Information Science in Engineering*, 13(4):041005, 2013. [cited on page(s) 48, 61]
- [ZR01] Gabriel Zachmann and Alexander Rettig. Natural and robust interaction in virtual assembly simulation. In *Proc. Int. Conf. on Concurrent Engineering: Research and Applications (ISPE)*, volume 1, pages 425–434, 2001. [cited on page(s) 17]
- [ZS95] C. B. Zilles and J. K. Salisbury. A constraint-based god-object method for haptic display. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 146–151, 1995. [cited on page(s) 30, 54, 58, 78, 116, 118]
- [ZW06] Gabriel Zachmann and Rene Weller. Kinetic bounding volume hierarchies for deformable objects. In *Proc. of the ACM Int. Conf. on Virtual Reality Continuum and its Applications*, pages 189–196. ACM, 2006. [cited on page(s) 36]

Notation and List of Symbols

This chapter describes the notation followed in the thesis, classified in the following tables:

- Table 1 collects a list of all **abbreviations**.
- Table 2 gathers **conventions for mathematical symbols**.
- Table 3 contains all symbols used for **mathematical operators and constants**.
- Table 4 presents the list of all **specific symbols**, with their description and unit.

Table 1: List of abbreviations. The abbreviations are valid through the whole text, unless exceptions are clearly stated. The factors and treatments from the user studies (Section 6.2 and Section 6.3) have dedicated sections at the end of the listing.

Abbreviation	Description	Definition
2D	Two Dimensions (planar Space)	page 35
3D	Three Dimensions (regular Euclidean space)	page 35
6D	Six Dimensions (translation and rotation or point and normal)	page 16
7D	Seven Dimensions (used for points in the enhanced pointshell)	page 86
AABB	Axis Aligned Bounding Boxe	page 32
ANOVA	Analysis of Variance	page 189
BV	Bounding Volume	page 31
BVH	Bounding Volume Hierarchy	page 45
BVTT	Bounding Volume Test Tree	page 46
CAD	Computer Aided Design	page 35
CCD	Continuous Collision Detection	page 38
CH	Convex Hulls	page 32
CoM	Center of Mass	page 91
CPU	Central Processing Unit	page 48

CSG	Constructive Solid Geometry	page 35
CSO	Configuration Space Obstacle	page 40
CT	Computer Tomography	page 61
DoF	Degree-of-Freedom (context of kinematics)	page 82
df	Degree-of-Freedom (context of statistics)	page 197
EMG	Electro-Myography	page 175
FA I	Fast Adapting, Small Field Mechanoreceptor (Meissner disk)	page 9
FA II	Fast Adapting, Large Field Mechanoreceptor (Pacini corpuscle)	page 9
FEM	Finite Element Method	page 60
FIFO	First In First Out	page 87
k -DOP	(k) Discrete Orientation Polytope	page 32
k -IOS	(k) Intersection of Spheres	page 33
GJK	Gilbert-Johnson-Keerthi (Algorithm)	page 146
GPUGPU	General Purpose GPU	page 48
GPU	Graphics Processing Unit	page 48
HACD	Hierarchical Approximate Convex Decomposition	page 146
HIP	Haptic Interface Point	page 54
I/O	Input and Output	page 149
IST	Inner Sphere Tree	page 33
KKT	Karush-Kuhn-Tucker conditions for quadratic programming	page 73
LC	Lin-Canny (Algorithm)	page 41
LIFO	Last In First Out	page 71
LWR	Light Weight Robot	page 171
LoD	Level of Detail	page 67
MRI	Magnetic Resonance Imaging	page 61
NURBS	Non-Uniform Rotational B-Splines	page 35
OBB	Object Oriented Box	page 32
ODE	Open Dynamics Engine	page 15
OSVR	Open Source Virtual Reality (Framework)	page 17
PCA	Principal Component Analysis	page 46
QoE	Quality of Experience	page 25
SA I	Slow Adapting, Small Field Mechanoreceptor (Merkel disk)	page 9
SA II	Slow Adapting, Large Field Mechanoreceptor (Ruffini corpuscle)	page 9
SAT	Separating Axis Theorem	page 39
SMC	Sequential Monte Carlo	page 171
SVM	Support Vector Machine	page 49
UDP	User Datagram Protocol	page 25

VA	Virtual Assembly	page 144
VE	Virtual Environment	page 13
VPS	Voxmap/Voxelmap Pointshell (Algorithm)	page 224
VR	Virtual Reality	page 3
VRPN	Virtual Reality Peripheral Network (Library)	page 16
WIMP	Window, Icon, Menu, Pointer	page 17

Section 6.2: Factor and treatment abbreviations in User Study 1

C	Constraint-based treatment for rendering factor	page 185
D	Haptic device factor	page 185
K	Force stiffness factor	page 186
l	Low treatment for stiffness factor	page 186
h	High treatment for stiffness factor	page 186
H	HUG treatment for device factor	page 185
P	Penalty-based treatment for rendering factor	page 185
R	Haptic rendering factor	page 185
S	Sigma.7 treatment for device factor	page 185

Section 6.3: Factor and treatment abbreviations in User Study 2

D1 ... D5	Virtualization degree factor	page 205
HD	Haptic device factor	page 204
HR	Haptic rendering factor	page 205
R	Real treatment	page 204
S	Synthetic treatment	page 204
VF	Visual feedback factor	page 204

Table 2: Conventions for mathematical symbols. If not specified otherwise, these listed conventions have been followed; most of them are related to parameter dimensionality. Table 4 gathers *specific symbols for specific magnitudes*, not covered in this table. Additionally, Table 3 shows the conventions used for *mathematical operators*.

Symbol	Description
a	Scalar value or parameter
\mathbf{a}	Vector
A	Coordinate frame in space
A	3D point or line (specified in text)
\mathbf{A}	Matrix – Exception: \mathbf{V} is a voxelmap and \mathbf{P} is a pointshell (see Table 4)
\mathcal{A}	Set – Exception: $\mathcal{O}(\cdot)$, Big O notation
a_D	Parameter associated with the <i>haptic device</i> frame
a_E	Parameter associated with the <i>eigen</i> frame

a_i	A concrete i instance of a
a_S	Parameter associated with the <i>god object</i> frame constrained to the <i>surface</i>
a_W	Parameter associated with the <i>world</i> frame
\mathbf{H}	Homogeneous transformation
${}^B\mathbf{H}_A$	Homogeneous 3×4 transformation matrix from frame A to B
\mathbf{u}	Unitary vector

Table 3: List of mathematical operators and constants. Mathematical dimensionality or related conventions are listed in Table 2.

Operator	Description
(\cdot, \cdot)	Structure or vector
$\{\cdot, \cdot\}$	Set
\cdot	Scalar product; often omitted
\times	Vectorial product
∇	Gradient operator
Δ	Delta or difference operator
\top	Transpose operator
$\lceil \cdot \rceil$	Ceil rounding operator
$\lfloor \cdot \rfloor$	Floor rounding operator
\oplus	Sum operation for all elements of a set one-by-one
\cap	Intersection operation
$\ \cdot\ _2$	Euclidean norm-2
\emptyset	Empty set
$\cos(\cdot)$	Cosine function
$\sin(\cdot)$	Sine function
$\tan(\cdot)$	Tangent function
$\max(\cdot)$	Maximization operation
$\min(\cdot)$	Minimization operation
$\mathcal{O}(\cdot)$	Big O
$\&a$	Reference operator: address in memory for a
\dot{a}	Time derivative of a
\tilde{a}	Approximation of a
\hat{a}	Unfiltered a
δa	Infinitesimal element of a
$\partial a / \partial b$	Partial derivative of a with respect to b
$a _{b=c}$	Selection operator $\cdot _{\cdot}$: elements of a such that $b = c$, being b a variable property of a and c a concrete value for property b

$a(b = c)$	Function operator (\cdot): value of a , which depends on variable b , for the case $b = c$
∂A	Boundary of object A
e	Euler's irrational number, approximated with 2.71828 – Exception: Restitution coefficient in Chapter 2
π	Pi irrational number, approximated with 3.14159

Table 4: List of symbols for specific magnitudes used in the thesis. Symbol units are also given in the last column, where dashes (–) stand for dimensionless parameters and stars (*) are related to case-specific units. Chapter or section-specific symbols have been classified into corresponding sections; otherwise, the symbol meaning holds for the whole text. General (magnitude-agnostic) notation conventions are described in Table 2 and mathematical constants in Table 3.

Symbol	Description	Unit
b, B	Virtual damping	N·s/m
b_D	Virtual damping related to the haptic device	N·s/m
b_{VC}	Virtual damping related to virtual coupling	N·s/m
\mathbf{f}	Force vector	N
\mathbf{f}_C	Force vector for constraint-based approaches	N
\mathbf{f}_D	Force vector displayed to the haptic device	N
\mathbf{f}_P	Force vector for penalty-based approaches	N
\mathbf{f}_{VC}	Force vector for virtual coupling	N
k, K	Virtual stiffness	N/m
k_C	Virtual stiffness related to constraint-based approaches	N/m
k_D	Virtual stiffness related to the haptic device	N/m
k_P	Virtual stiffness related to penalty-based approaches	N/m
k_{VC}	Virtual stiffness related to virtual coupling	N/m
μ	Coulomb friction coefficient	–
\mathbf{n}	Unitary normal vector	–
\mathbb{N}	Set of natural numbers (integer numbers)	–
\mathbb{R}	Set of real numbers (floating point numbers)	–
$\mathbb{R}^{n \times m}$	Set of real numbers (floating point numbers) in a space of dimension $n \times m$	–
$\mathbb{R}^3 \times \text{SO}(3)$	Set of configurations of an object in 3D Euclidean space: translation and rotation	–
$\text{SO}(3)$	3D rotation group which contains all rotations around the origin of 3D Euclidean space \mathbb{R}^3	–
$t, \Delta t$	Time, time step	s

t_d	Time delay	s
\mathbf{t}	Torque vector	N·m
T	Sampling time	s
$\mathbf{x}, \Delta\mathbf{x}$	Translation vector	m

Chapter 2: Background and Related Works

C_v, C_p, C_u	In a BVTT, time cost of BV checks (v), primitive checks (p), and node updated (u)	s
$d(\cdot, \cdot)$	Distance (function)	m
e	Restitution coefficient	–
f	Scalar force (on virtual spring)	N
$f(\mathbf{x}) = 0$	Scalar function or mapping on a vectorial variable \mathbf{x}	m*
f_A	Feature of object A , e.g., a vertex, an edge, or a face	–
λ_i	Lagrange multiplier	–
$\boldsymbol{\lambda}$	Vector of Lagrange multipliers	–
m	Mass of the haptic display and human arm	kg
N_v, N_p, N_u	In a BVTT, number of BV checks (v), primitive checks (p), and node updated (u)	–
$p(\cdot, \cdot)$	Penetration (function)	m
T_{BVTT}	Time cost of a BVH traverse or a BVTT	s
$V(f)$	Exterior Voronoi region of feature f	–
$\text{Vol}(\cdot)$	Volume (function)	m ³
$x, \Delta x$	Scalar translation or displacement (of virtual spring)	m
$\mathbf{x}_u, \Delta\mathbf{x}_u$	Unconstrained movement of an object	m
$\mathbf{x}_c, \Delta\mathbf{x}_c$	Constrained movement of an object	m
\mathbf{x}_{VC}	Virtual coupling or difference between the unconstrained and constrained motion	m
Z	Impedance exerted by a haptic device	kg/s, N · m · s/rad

Chapter 3: Collision Computation

A	Area	m ²
$\mathbf{b}_{\mathbf{P}}$	Bounding box vector the pointshell data structure	m
$\mathbf{b}_{\mathcal{T}}$	Bounding box vector the triangle	m
$\mathbf{b}_{\mathbf{V}}$	Bounding box vector the voxelmap data structure	m
c	Cluster of an enhanced pointshell	–
c^c	Children cluster associated to a cluster c	–
c^p	Parent cluster associated to a cluster c	–
C	Voxel center	m
\mathbf{C}	Matrix with ΔC_{ij} values	m
\mathcal{C}	List or set of clusters in an enhanced pointshell	–
ΔC_{ij}	Vector from voxel center i to j : $C_j - C_i$	m

d	Safety margin	m
\mathbf{d}	Vector from point P to its voxel center C : \overrightarrow{PC}	m
η	Computational load	–
η_c	Critical load	–
\mathcal{I}	Set of solid voxels: surface ($v = 0$) and inner ($v > 0$) voxels from \mathcal{V}	–
K	Number of cluster points in a cluster c	–
κ_1, κ_2	Principal curvatures	m^{-1}
L	Level in the pointshell hierarchy	–
L_c	Critical level in the pointshell hierarchy	–
\mathcal{M}	Contact manifold	–
$\Delta \mathbf{n}_{ij}$	Normal vector variation from voxel i to j : $\mathbf{n}_j - \mathbf{n}_i$	–
N_C	Number of clusters in the pointshell data structure	–
N_C^v	Number of spheres (or clusters) visited or checked for collision	–
$N_{\mathcal{P}}^c$	Number of colliding points	–
$N_{\mathcal{I}}$	Number of (solid) voxels in \mathcal{I}	–
N_K	Branching factor in the point-sphere tree	–
N_L	Number of levels in the pointshell hierarchy	–
$N_{\mathbf{P}}$	Number of points in the pointshell data structure	–
$N_{\mathcal{P}}$	Number of points in the pointshell data structure	–
$N_{\mathcal{P},L}$	Number of points in a level L	–
$N_{\mathcal{P},L_c}$	Number of points in the critical level L_c	–
$N_{\mathcal{P}}^v$	Number of points visited or checked for collision	–
$N_{\mathcal{Q}}$	Number of clusters in the query FIFO queue	–
$N_{\mathcal{S}}$	Number of surface voxel elements	–
$N_{\mathcal{T}}$	Number of triangles in the triangle mesh	–
$N_{\mathbf{V}}$	Number of voxels in the voxelmap data structure	–
$N_{\mathcal{W}}$	Number of voxels in the voxelmap with floating point distance values (w)	–
N_x, N_y, N_z	Number of voxels in each of the x, y, z axes	–
\mathbf{N}	Matrix with $\Delta \mathbf{n}_{ij}$ values	–
$\mathcal{N}_{\mathbf{V},n}(P)$	Voxel neighborhood of width $2n + 1$ around the voxel where point P is	–
ξ_{∇}	Step length form factor in gradient descent	–
ξ_V	Voxel form factor in layered signed distance computation (V_L)	–
p	Signed distance (or penetration) value between a voxelmap and a pointshell	m
p_c	Critical distance computation threshold	m
p_d	Signed distance (or penetration) value between a voxelmap and a pointshell dilated with the safety margin	m
P	A point in space, often a pointshell point, which is the projection of a C	m

\mathbf{P}	Pointshell data structure	–
\mathcal{P}	List of points in an enhanced pointshell	–
q	Quality or approximated curvature of a point or a cluster	–
q_c	Critical relative quality value of a cluster	–
q_{\max}	Maximum quality value of all points that are recursively branched from the cluster until the leaf level	–
Q	Pointshell point with the largest absolute V value in a query (i. e., deepest or closest point)	m
\mathbf{Q}	Shape matrix which relates matrices \mathbf{C} and \mathbf{N}	–
\mathcal{Q}	FIFO queue of clusters to check created in realtime	–
(R, X)	Radius R and the center X of the minimally bounding sphere that contains all children points recursively branched until the leaf level	m
s	Voxel size or voxel edge length	m
$\mathbf{s}_x, \mathbf{s}_y, \mathbf{s}_z$	Three unitary axes of a voxel or the voxelmap	–
$S(P)$	Closest point S constrained to the surface for a point P in space	m
\mathcal{S}	Surface voxel structure containing projected points and triangle vertices	–
T_1, T_2, T_3	Three vertices of a triangle	m
\mathcal{T}	Triangle mesh	–
v	Voxel layer value	–
$\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$	Three edge vectors of a triangle	m
V_L, V_S, V_I	Signed distance functions of a point P related to the layer (L), surface point (S), or interpolation (I) computation	m
V	Volume	m ³
\mathcal{V}	Voxelmap data structure; the primitive structure contains voxels with v values stored in \mathcal{V} , and the enhanced, additionally the structures \mathcal{S} and \mathcal{W}	–
\mathcal{V}	Primitive voxelmap or list of voxels containing v voxel layer values	–
w	Floating point distance value in the voxelmap	m
\mathcal{W}	Floating point distance voxel structure containing in each voxel w values and indices of the closest surface voxels in \mathcal{S}	–
ω_C	Weighting factor for the load part related to colliding spheres or clusters	–
ω_P	Weighting factor for the load part related to colliding points	–
X, Y, Z	Discrete voxelmap coordinates	–

Chapter 4: Force Rendering

$\mathbf{a} = (\ddot{\mathbf{x}}, \ddot{\mathbf{r}})$	Generalized acceleration	m/s ² , rad/s ²
A	Overlap region ($p_d > 0$)	–
α, β, γ	Angles	rad
B	Object body frame	–

B	Contact safety region ($0 < p_d \leq 0$)	–
$\mathbf{c} = (\mathbf{f}, \mathbf{t})$	Generalized contact wrench	N, N·m
C	No-overlap region ($p_d < 0$)	–
D	Device or tool frame	–
δ	Force-torque lever	m
e_c	Kinetic energy	J
$E(\mathbf{J}_B)$	Eigen-ellipsoid obtained after the PCA of the inertia tensor \mathbf{J}_B	m
E	Eigen frame of a body	–
η	Correction gain	–
θ	Correction rotation magnitude	rad
f_c	Cut-off frequency for god object pose filtering	Hz
\mathbf{F}	Generalized contact matrix	N·kg ² , N·kg ² ·m ⁴
${}^W\mathbf{H}_D$	Device frame	m, rad
${}^W\mathbf{H}_S$	God object frame	m, rad
${}^D\mathbf{H}_S$	Virtual coupling frame from device to god object proxy	m, rad
$\Delta\mathbf{H}^{\text{pen}}$	God object correction homogeneous transformation	m, rad
$\Delta\mathbf{H}^{\text{surf}}$	Constrained god object correction movement from previous to current cycle	m, rad
\mathbf{J}	Normalized inertia tensor	–
\mathbf{J}_B	Inertia tensor	kg·m ²
$L(\mathbf{f}_P, \mathbf{t}_P)$	Force application line	–
λ	Correction translation-rotation distribution factor	–
m	Mass scalar	kg
\mathbf{M}	Generalized mass matrix	kg, kg·m ²
\mathbf{M}_B	Mass matrix	kg
μ	Unknown parameter in the force application line $L(\mathbf{f}_P, \mathbf{t}_P)$	–
$\mu_{x,s}, \mu_{x,k}, \mu_{x,v}$	Static (s), kinetic (k), and viscous (v) friction coefficients for translation (x) and rotation (r)	–
$\mu_{r,s}, \mu_{r,k}, \mu_{r,v}$		
Q_L	Approximation of Q , or the equivalent deepest colliding point on force application line L	m
Q'_L	$\frac{Q_L}{GQ_L}$ after its associated penetration p has been solved rotating θ around the CoM G	m
ξ_p	God object correction transformation form factor or gain	–
r	Rotation radius	m
$\Delta\mathbf{R}^{\text{pen}}$	God object correction rotation matrix	rad
ρ	Correction factor	–
S	God object proxy or surface frame	–
S'	Corrected god object proxy or surface frame	–
σ	Inertia scalar	kg·m ²

τ	Torque-rotation projection factor	–
$\mathbf{u}_f, \mathbf{u}_t$	Normalized force and torque correction directions	–
$\mathbf{u}_x, \mathbf{u}_r$	Normalized translation and rotation correction directions	–
W	World frame	–
$\mathbf{x}_c, \mathbf{r}_c$	Constrained translation and rotation vectors	m, rad
$\mathbf{x}_{c,\parallel}, \mathbf{r}_{c,\parallel}$	Parallel component of the constrained translation and rotation vectors with respect to the penalty force and torque values	m, rad
$\mathbf{x}_{c,\perp}, \mathbf{r}_{c,\perp}$	Perpendicular component of the constrained translation and rotation vectors with respect to the penalty force and torque values	m, rad
\mathbf{x}_{fr}	Friction restriction movement	m
$\Delta\mathbf{x}_p, \Delta\mathbf{r}_p$	Translation and rotation correction vectors	m, rad
$\mathbf{x}_u, \mathbf{r}_u$	Unconstrained translation and rotation vectors	m, rad
$\mathbf{x}_{u,\parallel}, \mathbf{r}_{u,\parallel}$	Parallel component of the unconstrained translation and rotation vectors with respect to the penalty force and torque values	m, rad
$\mathbf{x}_{u,\perp}, \mathbf{r}_{u,\perp}$	Perpendicular component of the unconstrained translation and rotation vectors with respect to the penalty force and torque values	m, rad
$\mathbf{x}_{VC}, \mathbf{r}_{VC}$	Virtual coupling vectors from the device or tool to the god object proxy	m, rad
ω	Angular velocity	rad/s

Chapter 5: Applications

C_h	Grasping point of the object being manipulated	m
C_w	Workspace center	m
λ_w	Workspace movement rate	s ⁻¹
r_h	Distance between grasping point C_h and workspace center C_w	m
r_w	Workspace radius	m

Chapter 6: Evaluation of Methods

α	Level of significance	–
b	Bonferroni factor	–
d	Cohen's effect size statistic	–
δ	Cliff's effect size statistic	–
ϵ	Greenhouser-Geisser or Huynh-Feldt correction factor for due to sphericity violation	–
M	Average	–
Md	Median	–
N	Sample size (number of subjects) in user studies	–
F	Fisher statistic	–
p	Probability value	–
Q_i	Quantile i	–
r	Pearson correlation statistic	–
ρ	Spearman correlation statistic	–

S	General statistic, specified in text to be F or T	–
SD	Standard deviation	–
T	Student test statistic	–
V	Wilcoxon statistics	–
W	Mauchly's statistic	–
χ^2	Chi square distribution	–

List of Figures

1.1	Virtual simulations with haptic feedback	3
2.1	Human haptic sensory system	8
2.2	Integrated haptic interfaces	19
2.3	Energy gain in discrete simulations	21
2.4	Passivity, stability, and optimal control	22
2.5	Object representations	31
2.6	Common methods in collision and distance queries	39
2.7	Bounding Volume Hierarchies	45
2.8	Schematics of <i>penalty-</i> and <i>constraint-based</i> haptic rendering approaches	53
2.9	Virtual Coupling	56
3.1	Overview of the data structures and the algorithm	67
3.2	Generation of the basic primitives	69
3.3	Volumetric Forces	77
3.4	Pitfalls of computing collisions with voxelmaps and point clouds	79
3.5	Gradient or steepest descent in the voxelmap	80
3.6	The Signed Distance Voxelmap Function	82
3.7	Signed distance field sections of the Utah Teapot	85
3.8	Point quality values	88
3.9	Point clustering	90
3.10	Pointshell hierarchy of the Stanford Bunny	90
3.11	Diagram of the unified proximity and collision computation algorithm	94
3.12	Distance computation	99
3.13	Time critical queries with varied load and quality thresholds	102
3.14	Sphere and cube benchmarking scenario	106
3.15	Cube and sphere benchmarking results	107
3.16	Bunny and Teapot benchmarking scenario	109
3.17	Bunny and Teapot benchmarking results	110

4.1	Overview of the <i>god object</i> simulation in two consecutive time steps	119
4.2	Workflow of the <i>god object</i> simulation and force rendering method	121
4.3	Computation of the correction rotation θ	124
4.4	Computation of the <i>constrained</i> movement out of the <i>unconstrained</i> movement . . .	128
4.5	Friction model	133
4.6	God object algorithm benchmarking scenarios	139
4.7	God object algorithm benchmarking results: Bunny and Teapot scenario	140
4.8	God object algorithm benchmarking results: Bunny and thin plane scenario	141
5.1	Point sampled and voxelized representations of a virtual electronic box and a screw driver	148
5.2	Overview of the multi-body simulation framework focusing on collision computation	149
5.3	Multibody library architecture	150
5.4	Game control workflow	152
5.5	Integration of the vibrotactile armband VibroTac for cutaneous collision feedback .	155
5.6	Workspace navigation in large scenarios	156
5.7	Car assembly scenario description	158
5.8	Snapshots of the car assembly sequence	160
5.9	Performance results of the car assembly	161
5.10	Integration of new collision shapes to Bullet	163
5.11	Results of the bouncing ball experiment	164
5.12	Snapshots of the Stanford Bunny dropped onto a horizontal plane	166
5.13	Performance results of the Stanford Bunny dropped onto a horizontal plane	166
5.14	Segmented data structures of the Stanford Bunny	167
5.15	Overview of other applications	169
6.1	Model for virtual contact perception	177
6.2	Study 1: General diagram of the setup and the varied factors	182
6.3	Data structures of the virtual models used in the user studies	182
6.4	Models and tasks in the scenario of the first user study	183
6.5	Study 2: General diagram of the setup in the user evaluation	202
6.6	Real and virtual models and tasks in the scenario of the second user study	203
6.7	Degrees of virtualization, from purely real to purely virtual	205
6.8	Diagrams of values recorded during regular exercises of the same subject	209
A.1	The HUG bi-manual haptic device	228
A.2	The Sigma.7 haptic device	229
C.1	Results from Google Scholar related to relevant virtual reality terms	236

List of Tables

2.1	Works on collision detection and force rendering 1/2	27
2.2	Works on collision detection and force rendering 2/2	28
3.1	Solution of the triangle-point projection problem	74
3.2	Summary of structures, values, and queries that are defined in primitive and enhanced voxelmaps (V)	86
3.3	Summary of structures, values, and queries that are defined in primitive and enhanced pointshells (P)	93
3.4	Descriptive values during time critical queries	103
5.1	Results of the bouncing ball experiment	165
5.2	Results of the Stanford Bunny dropped onto a horizontal plane using different levels in the segmented collision detection	167
6.1	Descriptive data of the subjective dependent variables for each of the devices	188
6.2	Statistical analysis of the effect of the device on the subjective dependent variables	190
6.3	Correlations between subjective variables related to the haptic devices	191
6.4	Descriptive data of the objective dependent variables	192
6.5	Statistical analysis of the effect of all factors on the objective dependent variables	193
6.6	Descriptive data of the subjective dependent variables related to the exercises	196
6.7	Statistical analysis of the effect of all varied factors on the subjective dependent variables	197
6.8	Correlations between the independent variables related to the exercises	198
6.9	Descriptive data of the objective dependent variables for the whole exercise and the tasks	211
6.10	Statistical analysis of the objective dependent variables to determine the effect of the <i>Degree</i>	212
6.11	Descriptive data of the subjective dependent variables related to the exercises	213
6.12	Statistical analysis of the subjective dependent variables to determine the effect of the <i>Degree</i>	215

B.1	Main realtime methods in the classes <code>Voxelmap</code> (V) and <code>Pointshell</code> (P)	233
C.1	Results from Google Scholar related to relevant virtual reality terms	237
D.1	List of Publications by the Author	239
1	Abbreviations	271
2	Conventions for mathematical symbols	273
3	Mathematical operators and constants	274
4	Symbols	275

List of Algorithms

3.1 Proximity and collision computation algorithm: <code>collisionQuery()</code>	95
4.1 Friction computation algorithm: <code>computeFriction()</code>	134

