# Real-Time Haptic Interaction with Remote Environments using Non-contact Sensors

O. Fredrik Rydén

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2013

Reading Committee:

Howard Jay Chizeck, Chair

Blake Hannaford

Payman Arabshahi

Andrew Stewart

Program Authorized to Offer Degree:
Department of Electrical Engineering

University of Washington

**Abstract**

Real-Time Haptic Interaction with Remote Environments using Non-contact Sensors

O. Fredrik Rydén

Chair of the Supervisory Committee:
Professor Howard Jay Chizeck
Electrical Engineering

Haptic interaction with a haptic device has traditionally been limited to interaction with purely virtual objects or to static range scans of physical objects. Haptic interaction with physical objects at a distance has not been possible without the use of a remote contact manipulator.

This dissertation describes methods and algorithms for real-time haptic interaction with remote environments using non-contact sensors; remote touch. A method of rendering 3-DOF (degree-of-freedom) force feedback of streaming point cloud data (from a combined range/video sensor) is presented. This is further combined with a novel method for increasing the robustness in interaction with moving physical that is also evaluated. These methods are then extended to demonstrate that forbidden-region virtual fixtures [90] for telemanipulation can be rendered by extending the concept of remote touch to rendering of haptic virtual fixtures. Both of these methods are then extended to 6-DOF haptic interaction with force as well as torque feedback.

The presented methods of remote touch rendering and virtual fixture rendering were validated by implementation on commercially available hardware. The methods for remote touch successfully renders force feedback at $1000Hz$ based on the user's interaction with the streaming point cloud data. The methods for rendering of virtual fixtures are first demonstrated to protect an isolated beating heart using a forbidden-region virtual fixture and then to keep the user away from, as well as guide the user to, underwater valves.

## TABLE OF CONTENTS

## LIST OF FIGURES

## GLOSSARY

RANGE SENSOR: A sensor that can measure distances. Sometimes also referred to as a depth sensor.

RGB-D SENSOR: A combination of a RGB sensor and a range sensor, also referred to as a RGB-D camera.

STREAMING POINT CLOUD: A continuously refreshing point cloud derived from a range or RGB-D sensor.

HAPTIC RENDERING: Rendering of forces and torques corresponding to the user's actions in a virtual environment.

HAPTIC DEVICE CONFIGURATION: The position and rotation of the haptic device.

HIP: The haptic device position in the virtual environment.

PROXY: A virtual object that tracks the position of the HIP without violating constraints imposed by the environment. Specifically, the position of the proxy locally minimizes the distance between the proxy and the HIP.

THE VIRTUAL TOOL: The 6-DOF (degree-of-freedom) generalization of the proxy.

VIRTUAL FIXTURE: Spatial motion constraints in telemanipulation.

## ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation (NSF grant # 0930930), by the Strategic Environmental Research and Development Program (SERDP grant 13 MRSEED01-006/MR-2323) and by the Commercialization Gap Fund (University of Washington/Washington Research Foundation).

I would like to thank my advisor, Professor Howard Chizeck for his continuous support and for always challenging me. Most importantly, I would like to thank him for letting me do things my way.

I would also like to thank the members of my committee for staying patient even in times of tight deadlines and tough scheduling. Special thanks to Professor Blake Hannaford for his expertise in the fields of haptics and robotics. I would also like to thank Dr. Stewart and Dr. Arabshahi at the UW Applied Physics Lab for helping me putting my ideas to practical use.

Sina Nia Kosari deserves thanks for being an exceptional collaborator and for always asking the hard questions. Further, I would like to thank all of the members of the BioRobotics Laboratory for contributing to a great workplace.

Finally, I would like to thank my family, Mamma, Pappa, Louise och Elisabeth for always supporting me. Tack.

## DEDICATION

Till min morfar

# Chapter 1
# INTRODUCTION

In this dissertation, we present novel methods and algorithms for haptic interaction with a remote physical environment as a way to enhance the sense of remote presence (that is, telepresence). This approach allows a remote user to touch objects in the field of a range sensor. Such methods have become highly relevant due to recent advancements in the development of affordable range sensors (e.g., Microsoft Kinect, Primesense Carmine, Asus Xtion) with high accuracy and a small form factor. With range sensors becoming readily available in many homes today, one can imagine a video conferencing application with haptic capabilities such that a grandparent, potentially living far away, would be able to remotely feel the smooth skin of a newly born grandchild. In this work, we focus on challenges and solutions for making this vision a reality.

There are additional applications where it is advantageous to receive haptic feedback from a remote environment without actually touching the remote object. For this purpose, we propose a modified haptic rendering approach to aid in robotic telemanipulation. Specifically, we address the task of selectively keeping the remote manipulator out of forbidden zones in task space using haptic feedback. This method does not simply work as a collision avoidance algorithm, but rather in an anticipatory fashion that maintains accurate distances between the manipulator and the remote environment. This allows for more accurate manipulation even in time-varying environments with unknown dynamics.

## *1.1 Contributions*

In this dissertation, we present several contributions to haptic rendering and the broader field of telemanipulation. A background and explanation of these ideas appears in Chapter 3. The technical contributions can be summarized as:

- A method for haptic interaction with point clouds derived from a range sensor with-

out pre-processing or conversion of the point cloud. This is done by importing data from a range sensor to a virtual environment and developing an algorithm for haptic interaction with the range sensor data (the depth image).

- A method for 3-DOF (degree-of-freedom) haptic rendering of streaming point cloud data from a range sensor. This is done by placing a RGB-D (video + depth) sensor in a remote location and then streaming the sensor to a virtual environment in a computer. The user can then, using a haptic device, touch virtual representations of objects at the remote location as they appear in real-time.

- A method for increasing robustness in 3-DOF haptic interaction by locally estimating and correcting for the velocity of the point cloud. This solved a common problem where the user erroneously could push through a representation of an impenetrable surface in interaction with a moving object. When this occured, the haptic proxy fell through the point cloud representation and got stuck inside.

- A method for 3-DOF haptic rendering of forbidden-region virtual fixtures implicitly defined by streaming point cloud data. This is an important step towards anticipatory force feedback in teleoperation such as the imposition of protective force fields or forbidden regions around objects of interest.

- A method for 6-DOF full rigid-body haptic rendering of streaming point cloud data. This is important as most human touch interaction can be approximated as a rigid body interaction. This allows for rotations as well as translations wich means that the user can interact with a point cloud representation of the remote objects using for instance a hand model rather than just a point-like interaction.

- A method for defining and rendering 6-DOF virtual fixtures implicitly from the streaming point cloud data using color segmentation. With this method, the whole end-effector is considered in the rendering rather than just a spherical approximation. This results in less conservative rendering and gives the operator a large task space to operate in without violating the constraints imposed by the forbidden regions.

In addition to these theoretical contributions, the following experimental applications are developed in this dissertation:

- Remote touch with a moving object in front of a range sensor, for instance we demonstrated a *remote haptic handshake*.

- A demonstration of a virtual fixture defined implicitly by the range sensor data and rendered to protect a beating heart in real-time.

- An implementation of the aforementioned virtual fixture rendering method on a cable driven surgical robot.

- A demonstration of 6-DOF haptic rendering of forbidden-region virtual fixtures implicitly defined from RGB-D data captured in an underwater tank.

## *1.2 Publications Arising from Dissertation*

### *1.2.1 Journal Articles*

- **F. Rydén**, and H.J. Chizeck. "A Proxy Method for Real-Time 3-DOF Haptic Rendering of Streaming Point Cloud Data." IEEE Transactions on Haptics (2013). ©2013 IEEE, reprinted with permission in Chapter 4.

- **F. Rydén**. "Tech to the Future: Making a." Potentials, IEEE 31, no. 3 (2012): 34-36.

### *1.2.2 Articles in Conference Proceedings*

- **F. Rydén**, Andrew Stewart and H.J. Chizeck. "Advanced Telerobotic Underwater Manipulation Using Virtual Fixtures and Haptic Rendering." In OCEANS 2013. IEEE, 2013. ©2013 IEEE, partly reprinted with permission in Chapter 7.

- **F. Rydén**, and H.J. Chizeck. "A Method for Constraint-Based Six Degree-of-Freedom Haptic Interaction with Streaming Point Clouds" In Robotics and Automation (ICRA),

2013 IEEE International Conference on. IEEE, 2013. ©2013 IEEE, reprinted with permission in Chapter 6.

- **F. Rydén**, and H.J. Chizeck. "Forbidden-region virtual fixtures from streaming point clouds: Remotely touching and protecting a beating heart." In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pp. 3308-3313. IEEE, 2012. ©2012 IEEE, reprinted with permission in Chapter 5.

- **F. Rydén**, S.N. Kosari and H.J. Chizeck. "Proxy method for fast haptic rendering from time varying point clouds." In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, pp. 2614-2619. IEEE, 2011.

### *1.2.3 Workshop Papers and Abstracts*

- **F. Rydén**, S.N. Kosari and H.J. Chizeck. "A Computer Vision Approach to Virtual Fixtures in Surgical Robotics." In Proceedings of the Workshop on Algorithmic Frontiers in Medical Robotics: Manipulation in Uncertain, Deformable, Heterogenous Environments (in conjunction with RSS 2012). 2012.

- **F. Rydén**, H.J. Chizeck, S.N. Kosari, H. King and B. Hannaford. "Using kinect and a haptic interface for implementation of real-time virtual fixtures." In Proceedings of the 2nd Workshop on RGB-D: Advanced Reasoning with Depth Cameras (in conjunction with RSS 2011). 2011.

## *1.3 Dissertation organization*

The organization of this dissertation is as follows. In Chapter 2 we review important literature on 3-DOF and 6-DOF haptic rendering of virtual environments as well as the most relevant efforts in the virtual fixture. In Chapter 3 we present the main idea behind the proposed methods and how it relates to prior art. In Chapter 4 we present our method for 3-DOF haptic interaction with streaming point clouds derived from an RGB-D sensor as initially presented in [3]. In this chapter we also obtain a frame rate dependent bound on the robustness of the haptic interaction. In Chapter 5 we present a rendering method

for implicitly defining and, in real-time, updating forbidden-region virtual fixtures using streaming point cloud data, initially presented in [2]. In Chapter 6 we present a method of extending the 3-DOF rendering algorithm to 6-DOF haptic interaction with streaming point clouds using quasi-static simulation, initially presented in [4]. In Chapter 7 we present a method for 6-DOF rendering of forbidden-region- and guidance virtual fixtures implicitly defined by streaming point cloud data. Future work and research directions are discussed in Chapter 8.

# Chapter 2

# LITERATURE REVIEW

In this chapter we review work related to our proposed methods and algorithms. First we start by focusing on methods for rendering force feedback in a virtual environment. Then we review methods for doing the same in combination with rendering of torques. We continue by reviewing methods for rendering of haptic feedback in environments consisting of point cloud data. We review methods for implementing assistive force feedback in teleoperation—virtual fixtures. Finally, we review relevant literature for assisted teleoperation for underwater manipulators.

## *2.1 Haptic Interaction*

Haptic interaction with virtual objects in a computer, so called computer haptics, is a well investigated field. Some of the important works from the 1990s include [5], [6], [7], [8] although the idea had been around since the 1960s [9]. It has been shown that haptic feedback enhances the sense of presence in a virtual environment [10] and also the sense of being together in a multi-user shared virtual environment [11], [12]. Successful applications for haptics include medical simulators [13], [14], [15], [16], [17], for scientific visualizations [18], [19], and for kinesthetic- [20] and tactile [21] feedback in teleoperation. Haptic feedback is typically subdivided into tactile and kinesthetic feedback; the latter is often also called force feedback and will be the main focus in this dissertation.

Rather than focusing on specific applications or theory related to stability [22], [23], [25], [26], [27], [28], this dissertation addresses the general problem of rendering forces and torques from virtual environments consisting of point clouds derived from range sensors. In this section we present the most important works addressing the rendering of haptic feedback. In addition, we would like to refer the interested reader to the broader reviews [31], [32] and [33].

### *2.1.1 3-DOF Rendering of Polygonal and Implicit Surfaces*

In haptic interaction, an operator uses a physical *haptic device* to interact with objects in a virtual environment. This virtual environment can consist of representations of physical objects, as captured by sensor data. Or it could be purely virtual—produced by computer simulation. The *Haptic Interface Point* (or *HIP*) can be thought of as the position of an end effector in virtual space. It matches the position of the haptic device. When the user moves the haptic device, the HIP moves accordingly in the virtual environment. In this work we focus on haptic devices that sense position and exert force although other designs are possible as well. Fig. 2.1 shows a typical device for displaying translational forces (left) and a slightly more complex device for displaying both forces and torques (right). Forces

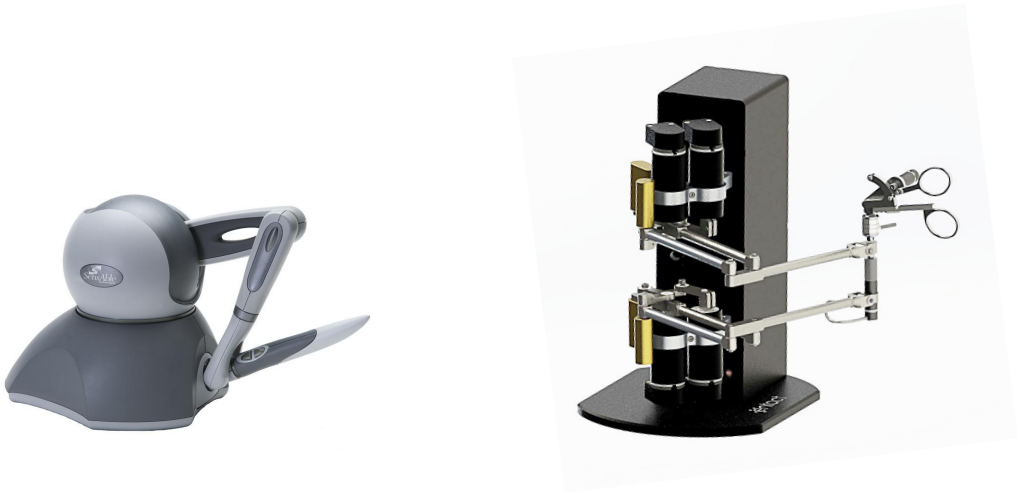

Figure 2.1: A 3-DOF (degree-of-freedom) Touch (Geomagic), formerly known as Phantom Omni, haptic device that senses position as well as orientation and exerts translational forces (Left). A 7-DOF (6-DOF plus grasping) W7D (Entact Robotics) haptic device that senses position and orientation but and exerts forces as well as torques (Right). In addition, this device also has the ability to sense the configuration on the grasper and can exert forces on the same.

can be exerted by the virtual environment to simulate touch sensations with objects. The simplest method for haptic rendering is the *direct method* (also known as a *penalty-based* method in 3-DOF rendering literature) where the force on the haptic device is based on the distance between the HIP and the surface. In some cases this works well, but upon

interaction with thin shapes these methods fail and the HIP *pops through* the surface. That is, the HIP moves through a surface when it actually should be constrained by the surface.

In the mid 1990s, several researchers developed alternatives to direct methods for haptic rendering. Most of these are classified as *virtual coupling* methods [34]. The god-object method [35] and the proxy method [36], [37], are two widely used virtual coupling methods for 3-DOF haptic rendering of polygons. By using these methods, the HIP did not pop through thin shapes. The main idea was to construct a virtual object (sphere or point) that could not penetrate shapes. This object was variously denoted as a *proxy*, a *god-object* or an *ideal HIP* (IHIP). The virtual object was then tied to the HIP by a virtual spring (this is the virtual coupling), thus specifying the force exerted on the user by the haptic device. The movement of the object relative to the HIP was then determined by iteratively solving a set of linear equations. Similar ideas were used to render haptic feedback based on implicit surfaces [38], [39], [40]. A step towards adding torque feedback in the haptic simulation was taken in [41] where the virtual object was modeled as a line segment.

### *2.1.2 6-DOF Haptic Rendering*

There exists a rich body of literature on six degree-of-freedom haptic rendering in virtual environments consisting of polygons and/or voxels (volume pixels). These efforts are typically divided into *direct rendering*- and *virtual coupling* methods where the latter can further be subdivided into *penalty*-, *impulse*- and *constraint*-based methods.

#### *Direct Methods*

The simplest method of 6-DOF haptic rendering techniques is the direct method [42], [43], [44] and [45]. In this approach, the virtual tool perfectly matches the configuration of the haptic device. The force sent to the user is directly based on the amount of penetration in the virtual environment. Unfortunately this method suffers from problems with pop-through, which is a rendering artifact that arises when the rendering algorithm erroneously penetrates a thin surface.

*Virtual Coupling Methods*

A virtual coupling between the haptic device and the virtual tool was suggested in [34] and successfully implemented for 3-DOF polygon haptic rendering in [35]. In this method, the force on the haptic device is simply calculated as a spring between the virtual tool, referred to also as 'god-object', 'proxy' or 'IHIP', and the configuration of the haptic device.

6-DOF rendering methods using this technique rely heavily on the work addressing rigid body simulation [46], [47], [37], [48], [49] since the virtual tool has to be simulated as a 6-DOF object, as compared to 3-DOF rendering where the rotational component can be ignored.

In penalty-based methods [50], [51], [52], [53], [54], [55], [56], [57], [58], [59], [60], [61] and [62], the configuration (position and rotation) of the virtual tool is calculated using penalty forces based on the tool's penetration depth into objects (similar to how penalty-costs are used in traditional optimization). These penalty forces are then integrated to produce the motion of the virtual tool. This method results in a virtual tool that actually penetrates objects in the environment. Fortunately this penetration is typically very small.

Impulse-based dynamics was first presented for rigid body simulation in [63]. In that work, a virtual object is moved by a series of impulses upon contact/collision (rather than forces based on penetration depth). Impulse-based haptic rendering [64], [65], [66] and [67], provides a computationally efficient approach to 6-DOF haptic rendering. This method is also much easier to implement compared to constraint-based haptic rendering since each point of contact can be dealt with individually [64].

In constraint-based methods [68], [69], [70], [71], [72], [73] and [74], the virtual tool moves into contact with the environment but (ideally) never violates the constraints imposed by the environment. The main approach in these papers is to simulate the virtual tool as a 'quasi-static' object (at rest) at every time step. This allows for simplified calculations. It is also desirable for the virtual tool to move into contact but not bounce off the surface at the contact point.

### *2.1.3 Point Cloud Haptic Rendering*

For the convenience of the reader, our work is referred to in the following discussion so that the context of this work and the literature is more readily described.

A point cloud can be thought of as an unstructured set of points representing the boundary of physical objects. It can be obtained using stereo cameras, laser scanners or depth cameras [75] in which case the data will be sorted in the image plane. RGB-D data is an extension of the point cloud structure by augmenting the data with color information.

Haptic rendering from point clouds has been investigated by several groups. In [50], haptic rendering was done on a point cloud was constructed by sampling a highly detailed mesh. A direct rendering method for static point clouds was presented in [76]. In this work, the surfaces were constructed a priori using a moving least square method [77]. Another approach can be found in [78], where haptic rendering of a static point cloud was done by placing a small bounding box around each point. They then implemented a tree-structure for rapid collision detection. Recently, an approach for directly interacting with a static point cloud has been presented in [79], [80] and [81].

Haptic rendering of recorded of point cloud data was first done in [82] and [83]. In those works, RGB-D data was recorded, after which each frame was tessellated. A god-object haptic rendering method was then used.

We presented the first haptic rendering method for streaming point cloud data, thus allowing for near real-time interaction (so called remote touch), in [1] where a constraint-based method for interaction with streaming RGB-D data was presented. We then extended this method to rendering of forbidden-region haptic virtual fixtures [2]. Another method for 3-DOF haptic rendering of streaming point cloud data was presented in [84] where a point cloud was converted to an implicit surface 'on-the-fly' (a gradient to an implicit surface was estimated at every time step). Two different methods were presented for doing this. In the first method, the implicit surface was obtained by radially expanding each point and smoothing the resulting volume. In the second method, the surface normal vector for any given point was calculated using total least-squares.

A quite different direction is taken in [85] where 2D images were augmented with haptic

features such as texture feedback and geometry feedback (from a depth camera). In that work, the depth data was overlaid directly onto the image without first converting the data to a point cloud.

To the best of the authors' knowledge, there exists no prior work where 6-DOF haptic rendering has been applied to streaming point cloud data. The transition to 6-DOF rendering has been challenging mainly because of the computational requirements involved in the rigid body simulation with a potentially large point cloud. An initial step was taken in [86] where an operator of a robot was provided with 6-DOF haptic feedback based on static point cloud data derived from a depth image (using the approach in [71]). We presented the first method for 6-DOF haptic interaction with streaming point cloud data in [4].

### *2.2 Teleoperation and Virtual Fixtures*

A bilateral teleoperation approach [20] has been taken to achieve remote touch using a remotely controlled robot. A drawback with this method is that the control loop introduces delay in the haptic feedback. This delay arises from the controller as it usually takes a certain time before a manipulator reaches the desired location. But the main drawback with this method is that force feedback only can be sent after collision with the remote environment has occurred. In this dissertation, we are interested in how we can convey haptic information without directly using a teleoperated manipulator, thus bypassing the controller and eliminating this delay as a way to prevent collisions by generating anticipatory force feedback.

A different approach to manipulator control is to take the human out of the loop—to automate. Then there is no need for haptic feedback. In fact, industrial robots with pre-programmed and sequentially triggered paths and actions have successfully been used in the manufacturing and assembly industry for years. This is because industrial robots successfully can repeat tasks with very high accuracy for environments that are either static or very well-modeled. Tasks in highly dynamic environments has on the other hand proven to be very difficult. This makes it difficult to use traditional techniques for industrial robots/manipulators in these settings.

Teleoperation with a human in the loop on the other hand provides the benefits of in-

dustrial robots while being a feasible solution with current technology. A way to leverage the cognition of humans as well as the repeatability and accuracy of automated manipulators is to overlay the teleoperated manipulator movement with some limited amount of automation. This is called assisted teleoperation and is often implemented using virtual fixtures.

A *virtual fixture* is defined as "abstract sensory information overlaid on top of reflected sensory feedback from a remote environment" [87]. In telerobotics, a virtual fixture is often explained using the analogy of a virtual ruler such that the operator remotely controls a robot and receives feedback when deviating from a predetermined path, such as haptic-, auditory- or visual feedback, including any combination of these. The virtual fixture can however be arbitrarily defined according to the needs of the application. In [87] and [88], virtual fixtures were shown to increase the ability to position a remote manipulator and in [89] shown to increase performance in path following tasks.

Applying forbidden-region virtual fixtures using a mesh shape or a simple virtual wall is a well investigated field. However there does not appear to be any prior work using forbidden-region virtual fixtures for protecting streaming point cloud representations of physical objects. In [90], a set of definitions were developed for virtual fixtures of guidance- and forbidden-region type, and they studied the stability of the same. A guidance virtual fixture typically guides the remote manipulator to a designated location in task space or along a designated trajectory whereas a forbidden-region virtual fixture acts as a no-fly zone in the task space where the manipulator should not go. If the operator violates the virtual fixture, for instance by commanding the manipulator to deviate from the designated trajectory or to enter a forbidden region, visual-, auditory- or haptic feedback can be used to notify the operator of this. Further, the system can be set up to either allow some deviations from the virtual fixture or completely reject these commands. In this work, we refer to these situations as soft- and hard constraints respectively. This dissertation focuses on using this framework for defining and updating the virtual fixtures implicitly defined by streaming point clouds derived from RGB-D sensors.

### *2.2.1 Virtual Fixtures from Computer Vision*

There exists some work on vision-based guidance virtual fixtures, *e.g.*, [91] where a predefined shape is registered to the environment using computer vision. This is different from our work where the virtual fixture is generated implicitly from the sensor data. A single CT scan was used in [92] to generate forbidden-region virtual fixtures in ENT. In [93], a sequence of CT scans of a beating heart were obtained. These were then used to generate and apply virtual fixtures sequentially during operation. CT scans were also used for generation of virtual fixtures in [94]. Real-time MRI has been used to implement guidance virtual fixtures [95]. X-ray fluoroscopy was used in [96] to define forbidden-region virtual fixtures around blood vessels. In [97], moving virtual fixtures were investigated by defining a 1-DOF virtual fixture over a moving object with a known motion.

In [98], a method for creating a forbidden-region virtual fixture based on a static point cloud (captured using stereo-vision cameras) was developed. This was done prior to operation, by selecting a region of interest and then tessellating the points in this region using multiple local surface normal estimates. An offset distance was then applied to the tessellated point cloud. During operation, the tool-tip of the slave robot was constrained from moving into the object of interest by sending a force to the master console, based on the penetration depth. This is the most relevant effort to our work within the virtual fixture literature.

### *2.2.2 Assisted manipulation in underwater applications*

The concept of virtual fixtures and assisted teleoperation is well-studied in many areas of robotics, yet underwater, remotely-operated vehicles (ROVs) lack this capability. As a result, and to the best of the authors' knowledge, there exists little work on rendering of dynamic virtual fixtures for underwater applications. However, sonars and subsequent three-dimensional imagery are often used for navigation and collision avoidance. Thus, there exists a set of relevant efforts that will be useful towards achieving our goal of implementing haptic rendering for underwater applications.

Work in underwater survey and mapping, where macro imagery is collected with under-

water cameras or sonar, often focuses on simultaneous location and mapping (SLAM) [99], [100]. Eustice et. al. [101], [102], [103] have used ROVs for ship hull inspection, developing three-dimensional models of structures underwater with both visual and sonar data. The authors of [104] used a sonar-equipped ROV to map ancient cisterns on the islands of Malta and Gozo where it is important to avoid potentially damaging collisions. Collision avoidance for AUVs (autonomous underwater vehicles) was evaluated in [105] using a 3D sonar and path planner that was updated at each time step. In [106], robust state estimation was achieved using a 3D sonar.

In this work we are focused on high-resolution, small-scale imagery to support dextrous manipulation of objects underwater. In [107], a static point cloud was constructed using a laser scanner and then used for grasp planning. Similar in spirit to rendering fixtures in a virtual environment, in [108] an offline simulator for ROV/AUV intervention was presented. In the UNION project [109], dynamic disturbance rejection as well as robust underwater color segmentation was achieved. In [110] and [111], a method for underwater target localization using sonar and vision was presented. This information was then used to position an ROV or AUV relative to the target.

# Chapter 3

# THE MAIN IDEA

In this dissertation, we present several novel ideas involving rendering of haptic feedback from streaming point clouds using our novel non-contact sensor approach as well as ideas involving using these techniques for rendering of virtual fixtures. Although the algorithms and methods presented in Chapter 4 through 7 are very different, they all fit into the same underlying idea. In this chapter, we present the framework for this idea and how it fits into prior art.

## *3.1 Haptic Rendering of Streaming Sensor Data*

When the haptic virtual environment is constructed using sensor data the user is no longer interacting with a purely virtual world. Essentially, the virtual environment becomes a virtual representation of the real world as seen by the sensor. If the sensor streams the data to the virtual environment in real-time, the user is in fact haptically interacting with the real world at a distance. We call this *remote touch* and the architecture for this is illustrated in Fig. 3.1. It should be noted that it is currently not possible to render complete five finger haptic interaction with kinesthetic and tactile feedback identical to real world interaction. Instead, many rendering methods focus on rendering the most important features of haptic interaction such as kinesthetic feedback and tactile feedback. An analogy is how live television in a similar fashion offers limited remote vision, i.e., you can not turn your head or dynamically adjust focus.

Even though it subjectively feels no different to interact with streaming sensor data compared to interaction with a purely virtual environment, there are a few fundamental differences. For instance, consider a moving object in a rigid body simulation where the configuration of the object in theory is completely known at any given time. This is however not true for streaming sensor data where a moving object (in the real world) is captured

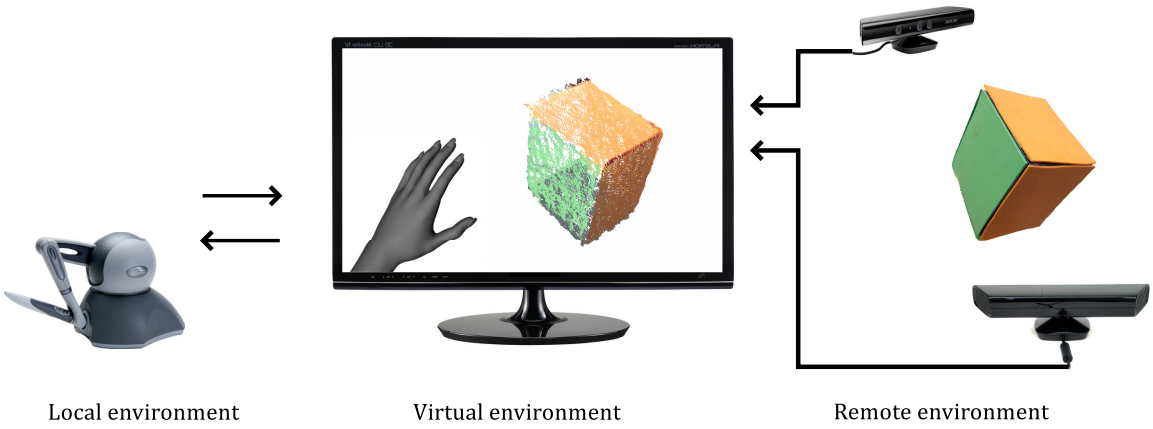


Figure 3.1: This figure illustrates the concept of remote touch where an operator in the local environment haptically interacts with a remote environment using a haptic device. The object, as captured in real-time by one or more combined, range and video, sensors in the remote environment, is imported into the virtual environment. The virtual environment renders haptic feedback to the operator by simulating object interaction using, in this case, a virtual hand model.

at a fairly low rate, relative to the receptors in our skin and muscles, where the object motion becomes discretized. That is, to allow for interaction with streaming sensor data the rendering algorithm needs to be robust to temporal discontinuities. Another challenge is the timing constraint as the sensor data has to be available quickly after it is captured. Even though we ideally would like the time between an event in the real world and the representation of that event in the virtual world to be zero. Since this is impossible, the goal instead becomes to minimize this delay. Finally, the haptic rendering also needs to be able to render small as well as sharp features in order for the interaction to feel realistic.

Haptic interaction with streaming sensor data, or streaming point clouds, can be used in several applications. For instance, consider having a combined range and video sensor (from this point on referred to as an RGB-D sensor) in a remotely located museum. As the sensor is being carried around, people around the world would be able to, using a haptic device, remotely touch the artifacts as held up in front of the sensor, including any moving parts. In another scenario, an underwater range sensor such as a sonar is placed on the seabed such that children in classrooms can use a haptic device to remotely touch fish or subsea

structures. Our 3-DOF haptic rendering method provides point-like haptic interaction which is sufficient in some applications. However, for most realistic haptic interactions a 6-DOF method, providing haptic feedback corresponding to both translations as well as rotations in the virtual environment, is needed.

One of the most important components in our work is our computationally efficient implementation of haptic rendering. It is sufficiently fast to process and render streaming point cloud data in real-time. Although some prior implementations were fast, they were not sufficiently fast to allow for remote touch or rendering of virtual fixtures.

### *3.2 Haptic Feedback in Telemanipulation*

One of the main challenges in implementing virtual fixtures on a remote manipulation system is to define the virtual fixture regions and update these as the environment changes. This dissertation addresses this challenge by proposing that virtual fixtures can be defined implicitly by the RGB-D sensor data and that virtual fixture feedback can be generated using a haptic rendering method for streaming RGB-D data. Strictly, this idea falls under the category of model-based haptic feedback where there in addition to the operators environment and the remote manipulator environment also exists a virtual environment. This environment consists of a real-time manipulator simulation registered to the RGB-D sensor data (with the implicitly defined virtual fixtures) and essentially predicts the outcome of the operator commands allowing for anticipatory haptic feedback. This concept is illustrated in Fig. 3.2.

Our 3-DOF virtual fixture rendering algorithm works best when the remote manipulator is a narrow tip or another very small structure. The algorithm then correctly renders virtual fixtures with respect to this tip (but not with respect to the shaft). To rendering virtual fixture with respect to the full geometry of a remote end effector, our 6-DOF version needs to be used. In addition, this method provides both force as well as torque feedback to the user.

This technology is applicable to many manipulation tasks where operations has to be executed in remote or inaccessible environments, this includes offshore maintenance and operations, satellite inspection, robotic surgery and mining. The techniques and meth-

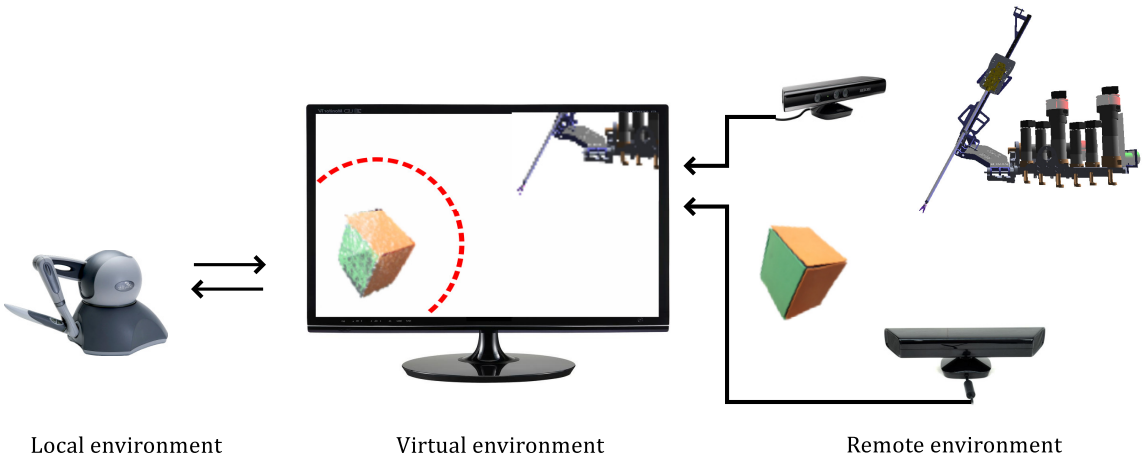


Figure 3.2: The concept of using a haptic rendering method to implement virtual fixture for remote manipulation. The operator controls the remote manipulator using a haptic device in the local environment. The virtual environment consists of a real-time anticipatory manipulator simulation in which streaming sensor data is imported and virtual fixtures formed. The haptic feedback is then rendered based on the interaction between the virtual manipulator and the virtual fixtures.

ods described in this dissertation are readily applicable to many robotic and manipulator systems.

Chapter 4

# 3 DEGREE-OF-FREEDOM HAPTIC INTERACTION WITH STREAMING POINT CLOUDS[1]

This paper presents a method for real-time 3-DOF haptic rendering of streaming point clouds. Using point clouds to represent physical objects has recently gained popularity because of the availability of inexpensive RGB-D cameras such as Kinect (Microsoft Corp.). The point clouds representing physical objects are captured and streamed to a computer. Using haptic interacting with these point clouds, *one-way remote touch* is achieved.

There exist several challenges in haptic interaction with streaming point cloud data. The most critical challenge is the latency induced in the haptic rendering. This has to be kept to a minimum in order for the haptic rendering to be perceived by the user as being in real time. This imposes a computational speed requirement on any haptic rendering method for moving objects, used in real-time interactions. A second challenge is to correctly render the smallest possible geometries for a given point cloud density (spatial sampling rate). The haptic resolution (the level of detail the user will be able to feel) is highly dependent on the density of the point cloud. Haptic interaction with point cloud representations of physical objects is non-trivial since no information is given regarding which object a point belongs to and since no information about surface normals is given. A third challenge in the design of a haptic rendering algorithm is to be robust to point cloud movements as well as noise.

In this paper we present a method for haptic rendering that addresses the aforementioned challenges by extending the idea of a haptic proxy to streaming point clouds. These challenges were first addressed in [1] where an iterative proxy method was implemented. This paper extends that work by presenting a new method for choosing a proxy step size as well as a novel method for estimating point cloud velocity. The use of a variable proxy step size

---

[1]F. RYDÉN, AND H.J. CHIZECK. "A PROXY METHOD FOR REAL-TIME 3-DOF HAPTIC RENDERING OF STREAMING POINT CLOUD DATA." IEEE TRANSACTIONS ON HAPTICS (2013). 

yields improved haptic accuracy as well as fewer iterations (and hence faster computation). The point cloud velocity estimation reduces the risk for incorrect haptic rendering, and *pop-through* when in contact with a moving point cloud, by correcting the proxy position at every point cloud update.

## *4.1 Challenges in Haptic Interaction with Streaming Point Cloud Data*

The use of streaming point cloud data in haptic rendering imposes some new challenges as well as opportunities. The main opportunity is that it allows us to interact with dynamic and/or moving objects in real-time. The goal is to design a haptic rendering method for streaming point clouds with performance comparable to methods for polygons while meeting the timing constraints for real-time applications.

### *4.1.1 Challenge 1: Timing/Latency*

The haptic rendering method should not induce excessive latency if it is to be considered real-time. It has been shown that 99% of the population is unable to perceive a delay between visual and haptic feedback if the delay is less than $54ms$ [112]. For our purpose we denote our haptic rendering method to be real-time when it is perceived as real-time. The main sources of latency are the processing on the RGB-D camera and the computational effort in the haptic rendering. In this work, the challenge of minimizing the latency involved in the haptic rendering method is addressed.

### *4.1.2 Challenge 2: Interaction with Moving Objects*

A physical object moving at a constant velocity is represented in a streaming point cloud as points moving with discrete jumps. At higher velocities this might cause a problem as the jumps become larger. In addition to this, the point cloud is typically subject to measurement noise. The challenge is to construct a proxy method that is robust to both movements and noise in the point cloud.

### *4.1.3 Challenge 3: Haptic Resolution*

The resolution for point cloud haptic rendering is highly dependent on the density of the captured point cloud as well as the size and geometry of the proxy. The goal is to design a haptic rendering method that allows the user to distinguish the smallest possible features for a given spatial sampling rate. This work focuses on rendering the correct geometry of objects given that the point cloud is dense (no up-sampling is needed). The representation of surface mechanical properties, such as friction, are beyond the scope of this paper.

## *4.2 The Haptic Rendering Method for Streaming Point Clouds*

We have developed a method for haptic rendering that addresses the challenges mentioned in the previous section. Our initial approach was presented in [1] where the main idea was an adaptation of the proxy method to point clouds by iteratively moving the proxy towards the HIP while estimating surface normals at every proxy step (see Fig. 4.1). This was done to prevent the proxy from moving through points and instead remain on an estimated surface. The force sent to the haptic device was calculated as a virtual spring-damper coupling between the HIP and the proxy. An extended and improved version of that haptic rendering method is presented in this section.

The adaptation of prior proxy method to point clouds is done by extending the traditional spherical proxy with three different regions. These regions are used to detect the current *proxy state* while moving in the point cloud.

The proxy is moved iteratively by a proxy movement algorithm which starts when a new point cloud is captured or when the haptic device moves. The proxy movement algorithm moves the proxy in small steps until it converges (i.e., reaches the desired position in the point cloud). The force sent to the haptic device is calculated as a virtual spring-damper between the HIP position and the proxy position (as given by the proxy movement algorithm).

In prior work the proxy would *pop through* if the point cloud moved too fast. In this section we present a method for preventing pop through by estimating and correcting for the point cloud velocity at each point cloud update. This essentially enforces an upper bound

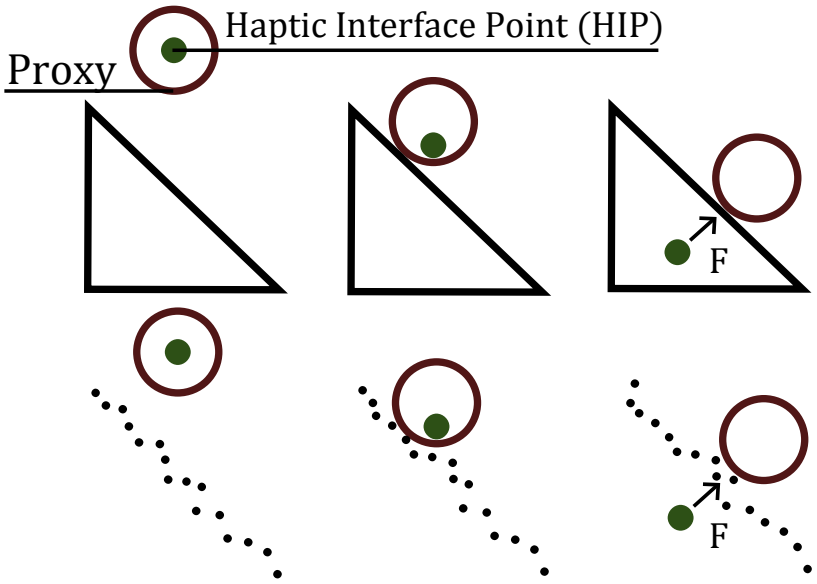


Figure 4.1: A two-dimensional illustration of the similarities between traditional haptic rendering methods and our proposed method for haptic rendering of streaming point clouds. Just as the proxy is constrained by the polygon (top), the proxy is constrained by the point cloud (bottom). [1]

on point cloud acceleration rather than velocity. However the proxy might pop through at lower accelerations due to the effects of noise.

The presented method involves many variables and definitions, and the most important ones are listed in Table 4.1.

### *4.2.1 A Proxy in a Point Cloud Environment*

The classic proxy as developed in [36] is defined as a sphere which perfectly serves the purpose in a virtual environment consisting of polygons with well-defined surfaces. For the proxy to move in a noisy point cloud environment, the proxy is defined here to have three radii $r_1 < r_2 < r_3$ extending out from its center (see Fig. 4.2). The region inside $r_1$ is used to detect when the proxy is about to pop through a point cloud. The region between $r_1$ and $r_2$ is used to detect contact with a point cloud, since the proxy most likely will be in contact with noisy point clouds rather than well defined surfaces. The contact point is defined to be in the middle of this region; that is $r_c = \frac{r_1+r_2}{2}$. All the points within $r_3$ are used for the

Table 4.1: Variables and definitions

| | |
|---|---|
| $f_c$ | The frame rate of the point cloud source |
| $j = 0, 1, 2...$ | Numbering of point cloud frames |
| $k = 0, 1, 2...$ | Numbering of proxy steps |
| $l = 0, 1, 2...$ | Numbering of force frames |
| $\mathbf{P}_k$ | The proxy position (at step k) |
| $r_{\{1,2,3,c\}}$ | Radii defining proxy regions |
| $\mathbf{u}_k$ | The vector between the proxy and the HIP (in the beginning of step k) |
| $\mathbf{s}_k$ | The change in proxy position (at step k) |
| $d_k$ | The step size calculated in Section 4.2.4 |
| $\xi$ | A function determining the step size when moving large steps in contact |
| $\gamma$ | A scaling factor for small proxy steps when the proxy is in contact |
| $\theta$ | A constant to determine proxy convergence |
| $\hat{\mathbf{n}}_k$ | The estimated normal vector at $\mathbf{P}_k$ |
| $\psi(r)$ | The modified Wendland weighting function |
| $\mathbf{s}_j$ | The correction in proxy position due to point cloud velocity (at frame j) |
| $\hat{\mathbf{v}}_\mathbf{j}$ | The estimated point cloud velocity vector |
| $v_j, \hat{v}_j$ | The true and estimated point cloud velocity respectively |
| $\mathbf{F}_l$ | The force sent to the haptic device (at frame l) |
| $\mathbf{u}_l$ | The most recent $\mathbf{u}_k$ at force frame $l$ |

normal estimation (see Fig. 4.3).

Thus during haptic interaction with a point cloud the proxy will be in one of three

possible states [1]:

1. In *free motion* if there are no points within $r_2$ from the proxy.

2. In *contact* if there are points between $r_1$ and $r_2$ but no points within $r_1$ of the proxy.

3. In *entrenchment* if there are points within $r_1$ of the proxy. This can be thought of as the proxy digging into the point cloud.

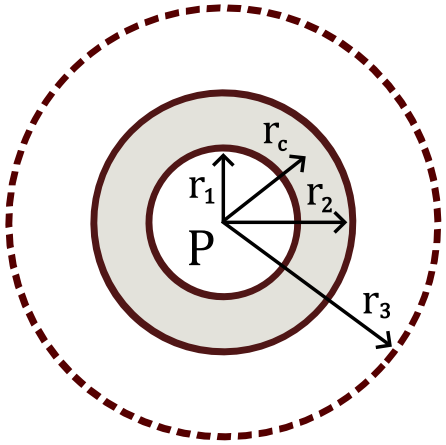


Figure 4.2: 2-dimensional cross-section of the proxy. Proxy regions are defined by $r_1 < r_c < r_2 < r_3$.

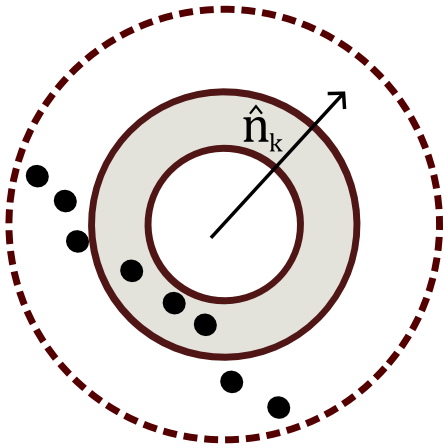


Figure 4.3: The normal vector estimate $\hat{\mathbf{n}}_k$.

### *4.2.2 Local Surface Normal Estimation*

The proxy should ideally never move through points in order to achieve correct haptic rendering. This can be enforced by using the points to form a linear constraint on the proxy movement and then moving the proxy iteratively perpendicular to this constraint. Note that when a spherical object (such as a proxy) comes into contact with any continuous surface, the normal vector at any contact point points towards the center of the sphere. That is, the geometry of the proxy can be used to estimate a normal vector to the surface. The alternative to this approach would be to compute a normal vector for each point using total

least squares, as done for point cloud surface reconstruction in [113], which would involve an eigenvalue decomposition for each point.

Our approach avoids this repeated eigenvalue decomposition. It also produces the expected result when in contact with a single point. To further improve speed, the constraints from the points inside radius $r_3$ are weighted and grouped to form a single planar constraint to the proxy; this weighting assumes that the points approximately are uniformly sampled within $r_3$. In many situations it is sufficient to group the movement constraints into a single constraint as long as the step size is small enough (which it will have to be anyway due to linearization of the constraints). Another implication of this approach is that the point cloud can be touched from any direction.

Every time before the proxy moves, this normal vector estimate $\hat{\mathbf{n}}_k$ is found as follows:

1. Let $\mathbf{x}_i,\ i = 1, 2, ..., N$ be the points within radius $r_3$ of the proxy.

2. Let

$$\hat{\mathbf{n}}'_k = \sum_{i=1}^{N} \frac{\mathbf{P}_k - \mathbf{x}_i}{\|\mathbf{P}_k - \mathbf{x}_i\|_2} \psi(\|\mathbf{P}_k - \mathbf{x}_i\|_2) \tag{4.1}$$

   $\hat{\mathbf{n}}_k$ is then found by normalizing $\hat{\mathbf{n}}'_k$.

In (4.1), $\psi(r)$ is a modified version of the Wendland function [114] such that:

$$\psi(r) = \begin{cases} 1 & \text{if } r \leq r_1 \\ \left(1 - \frac{r - r_1}{r_3 - r_1}\right)^4 \left(\frac{4(r - r_1)}{r_3 - r_1} + 1\right) & \text{if } r_1 < r < r_3 \\ 0 & \text{if } r \geq r_3 \end{cases} \tag{4.2}$$

This smooth weighting function is monotonically decreasing between $r = r_1$ and $r = r_3$. As a result, new points are introduced gradually (starting with low weight) as the proxy moves in the point cloud. The Wendland function was suggested for point cloud haptic rendering in [84].

The transformation of points from an RGB-D frame to a Cartesian coordinate is done in parallel utilizing general purpose GPU computing. Consequently timing issues of concern in [1] are resolved. This method decreases CPU load and does not increase GPU load since all points would have to be transformed for visualization anyway.

For a large point cloud it can be computationally expensive to find the points that are in the neighborhood of the proxy. Luckily, most point clouds derived from RGB-D cameras are sorted horizontally and vertically in an array according to some coordinate frame. By transforming the proxy position to this frame, the neighboring points can be found. Details regarding finding points in the neighborhood of the proxy can be found in [1].

If the number of points in the neighborhood around the proxy is relatively small, a brute-force search of this neighborhood is sufficient. That is what we have done in this paper. For very dense point clouds, the points in a (larger) neighborhood can be ordered in a tree-structure at every point cloud update. Unfortunately, this will bound the distance that the proxy can move until the next point cloud update, as collision outside the scope of this tree structure will be missed.

### *4.2.3 Proxy Movement Algorithm*

This algorithm is, based on the normal vector estimate and the state of the proxy, moving the proxy to its new position in small steps of varying size (this step size is described in Section 4.2.4). When the point cloud updates (and potentially moves), the algorithm corrects for the point cloud velocity by moving the proxy along an estimated velocity vector (this velocity is estimated in Section 4.2.5). The algorithm starts when either the haptic device moves or when the point cloud updates, and it iterates until the proxy has converged.

1. If the point cloud was updated since the last iteration, correct for this by moving the proxy one step

$$\mathbf{s}_j = \frac{1}{f_c}\hat{v}_j\hat{\mathbf{n}}_{k-1} \tag{4.3}$$

   where $\hat{v}_j$ is the point cloud velocity to be estimated in Section 4.2.5. This velocity is assumed to be in the direction of the previous normal vector.

2. Determine the proxy state.

3. If the proxy is in free motion (as described in Section 4.2.1 and determined by a brute

force search), move the proxy one step

$$\mathbf{s}_k = d_k \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|_2} \tag{4.4}$$

along the vector between the proxy and the HIP ($d_k$ is the step size determined in Section 4.2.4).

4. Else if the proxy is entrenched, move the proxy one step

$$\mathbf{s}_k = d_k \hat{\mathbf{n}}_k \tag{4.5}$$

in the direction of the normal vector estimate.

5. Else if the proxy is in contact and the HIP is inside the estimated surface (see Fig. 4.4), project $\mathbf{u}_k$ on the plane defined by $\hat{\mathbf{n}}$ (denote this vector $\mathbf{u}_{k,p}$). Then move the proxy one step

$$\mathbf{s}_k = d_k \frac{\mathbf{u}_{k,p}}{\|\mathbf{u}_{k,p}\|_2} \tag{4.6}$$

in the direction of $\mathbf{u}_{k,p}$.

6. Else if the proxy is in contact and the HIP is outside the estimated surface, move the proxy one step

$$\mathbf{s}_k = d_k \hat{\mathbf{u}}_k \tag{4.7}$$

which moves the proxy towards the HIP and eventually out of contact (see Fig. 4.5).

7. Iterate steps 1 to 7 until the proxy has converged. If the proxy is in contact, it has converged if

$$\cos^{-1}\left(\frac{-\mathbf{u}_k^T \hat{\mathbf{n}}_k}{\|\mathbf{u}_k\|_2}\right) < \theta \tag{4.8}$$

where $\theta$ is a constant representing a small angle between $-\mathbf{u}_k$ and $\hat{\mathbf{n}}_k$. If the proxy is in free motion, it has converged if the HIP position equals the proxy position.

There will most likely be several iterations of this algorithm for every haptic frame (indexed with $l = 0, 1, 2 \dots$) and even more iterations for every point cloud update (indexed with $j = 0, 1, 2 \dots$). One iteration of the proxy movement algorithm is referred to as a *proxy step*, and $\mathbf{P}_k$ should therefore be interpreted as the proxy position at step $k = 0, 1, 2 \dots$.

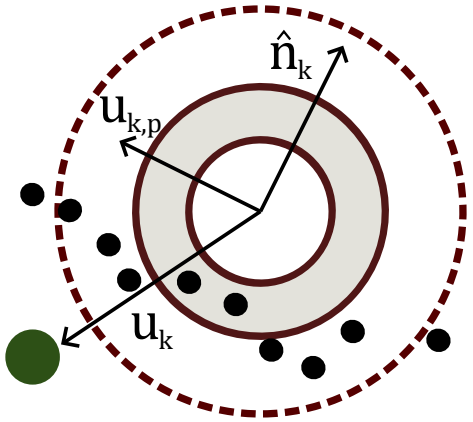


Figure 4.4: The proxy is in contact and the HIP is inside the estimated surface. In this case the proxy will move in the direction of $\mathbf{u}_{k,p}$.

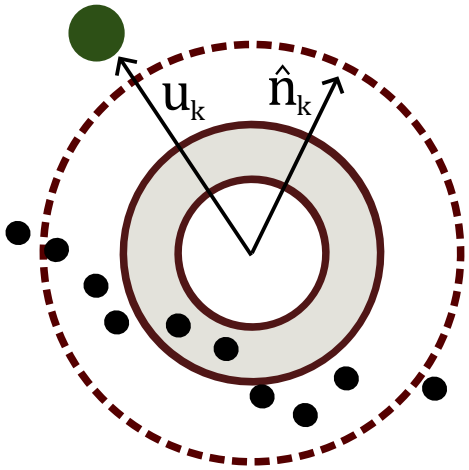


Figure 4.5: The proxy is in contact. Since the HIP is moving out of the surface the proxy will move in the direction of $\mathbf{u}_k$.

### *4.2.4 Variable Step Size for Improved Performance*

In [1], the proxy was moved with fixed size steps. In order to achieve good haptic resolution, the step size had to be small. Because of this, many iterations were required for the proxy to move longer distances even if no points were constraining the proxy's path. It also caused the proxy to jitter up and down since it almost never made perfect contact. In this paper, we address these problems by considering a variable step size $d_k$ (associated with proxy step $k$) determined by the points in the neighborhood of the proxy.

#### *Step Size in Free Motion*

The desired step size for a proxy in free motion is the largest distance the proxy can move in a straight line towards the HIP without passing through any points, see Fig. 4.6. This step

size is found by considering each point as an individual constraint on the proxy movement. For each point, the distance that the proxy could move towards the HIP without passing through the point is calculated. The desired step size is chosen as the smallest of those distances (which takes all contraints into account).

Let $\mathbf{x}_i,\ i = 1, 2...M$ be the set of M points that could potentially constrain the proxy movement. The step size associated with the i-th point ($d_i$) is then found by solving

$$r_c - \left|\left|\mathbf{x}_i - \mathbf{P}_k - d_i \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|_2}\right|\right|_2 = 0 \tag{4.9}$$

for the solution with the smallest magnitude (this has to be done M times). Step size is then chosen as $d_k = \min_i d_i$. If $d_k$ is larger than the distance the proxy would have to move to perfectly match the position of the HIP, simply set $d_k = \|\mathbf{u}_k\|_2$.

*Step Size in Entrenchment*

For an entrenched proxy the desired step size is the smallest distance the proxy has to move to get all points out of entrenchment. Let $\mathbf{x}_i,\ i = 1, 2...M$ be the entrenched points (all points within region 1 of the proxy). The step size associated with the i-th point ($d_i$) can be found as the positive solution of

$$r_c - \|\mathbf{x}_i - \mathbf{P}_k - d_i \hat{\mathbf{n}}_k\|_2 = 0 \tag{4.10}$$

Then choose $d_k = \max_i d_i$ (see Fig. 4.7).

*Step Size in Contact*

When the proxy is in contact the goal becomes to move the proxy towards the ideal position on the surface. This ideal position can be thought of as the position locally minimizing the distance between the proxy and HIP, or finding a position such that the angle between $-\mathbf{u}_k$ and $\hat{\mathbf{n}}_k$ is 0 (in practice this angle is set to a small constant $\theta$). Consider thinking of $\mathbf{u}_{k,p}$ as a gradient estimate to the distance between the proxy and the HIP (which it is for some coordinate frame). The movements towards the ideal position could then be found using numerical optimization by approximating a Hessian at every proxy step. However this is

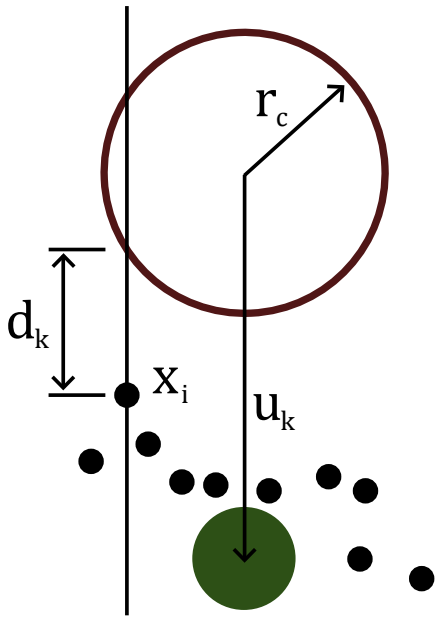


Figure 4.6: $d_k$ is the largest distance the proxy can move (in the direction of $\mathbf{u}_k$) without passing through any points.

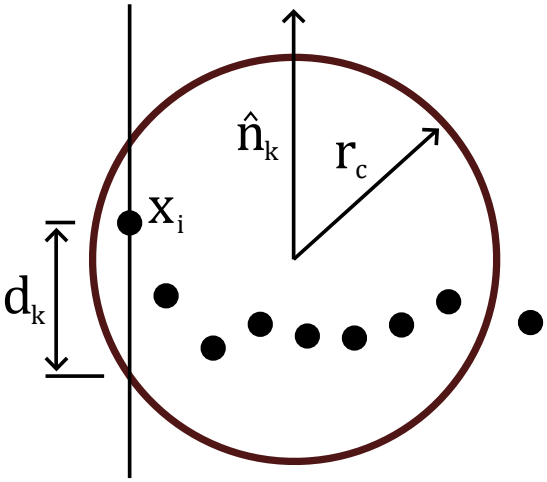


Figure 4.7: $d_k$ is the shortest distance the proxy has to move (in the direction of $\hat{\mathbf{n}}_k$) to get all the points out of entrenchment.

not a good idea for haptic rendering with a streaming point cloud since the point cloud does not represent an underlying smooth function.

Instead, when a proxy is in contact with the surface and closer than $r_1$ to the ideal position (as determined by the projection $\mathbf{u}_{k,p}$), the step size $d_k$ should equal the magnitude of $\mathbf{u}_{k,p}$ scaled by a factor $\gamma \leq 1$ to suppress errors in the normal vector estimate. That is, when $\|\mathbf{u}_{k,p}\|_2 < r_1$ choose

$$d_k = \gamma \|\mathbf{u}_{k,p}\|_2 \tag{4.11}$$

where $\gamma$ can be chosen as a function of the number of proxy steps taken while closer than $r_1$ to the ideal position. This way the proxy steps become smaller as the number of iterations increase.

When the proxy is in contact and farther away than $r_1$ from the ideal position on the

point cloud, the step size is chosen as

$$d_k = \xi r_1 \tag{4.12}$$

where $\xi < 1$ is a constant. This step size is chosen after observing that a proxy in contact might become entrenched after a movement of size $< r_1$ but realizing that this will be corrected for in the next proxy step. *The important thing is that the proxy never will be able to pass through a point cloud in a single step.*

### *4.2.5 Point Cloud Velocity Estimation*

A physical object moving at a constant velocity $v$ is represented in a point cloud as a set of points moving in discrete steps of size $v/f_c$. Upon haptic interaction with a point cloud moving towards the proxy at a low velocity, the proxy becomes entrenched at every point cloud update. This is then corrected for in the next proxy step when the proxy moves in the direction of the estimated normal vector. In this setting, point cloud movements larger than $r_1$ per update will cause the subsequent normal estimate (if any) to point in the wrong (opposite) direction. As a result the proxy will move out of entrenchment in the wrong direction. This is referred to as a *pop through*.

One could argue that a solution to this would be to keep track of the points where the proxy is in contact and then constrain the proxy to always be on the correct side of these points. However, this is not feasible in a point cloud without any information regarding which physical object or surface a point belongs to.

Instead, consider correcting for the point cloud velocity at every point cloud update by moving the proxy along a local velocity vector estimate (as done in step 1 of Section 4.2.3). Even though it is called a velocity vector, only the velocity component towards the proxy is estimated, since this is the only component that could potentially cause a pop through. That is, it is perfectly fine for a point cloud to *slide* sideways under the proxy, as the velocity component towards the proxy is zero in that case. If the proxy becomes entrenched at the point cloud update (but after the velocity correction), it will be interpreted as a positive point cloud acceleration (in a finite difference sense). Conversely, if the proxy is in free motion after the point cloud update, it will be interpreted as a negative point cloud

acceleration. In either case, the velocity estimate is corrected using the length of proxy step taken right after the point cloud update ($\mathbf{s}_k$), projected onto the normal vector that was estimated just before point cloud update ($\hat{\mathbf{n}}_{k-1}$), scaled by the point cloud update rate ($f_c$). These quantities are illustrated in Fig. 4.8. Given this local velocity, estimated at point cloud update $j-1$ (assume $v_0 = 0$), the velocity estimate at point cloud update $j$ becomes

$$\hat{v}_j = \hat{v}_{j-1} + \hat{\mathbf{n}}_{k-1}^T \mathbf{s}_k f_c \tag{4.13}$$

If the estimated velocity becomes negative, set $\hat{v}_j = 0$ since there is no risk for pop through when a point cloud is moving away from the proxy.

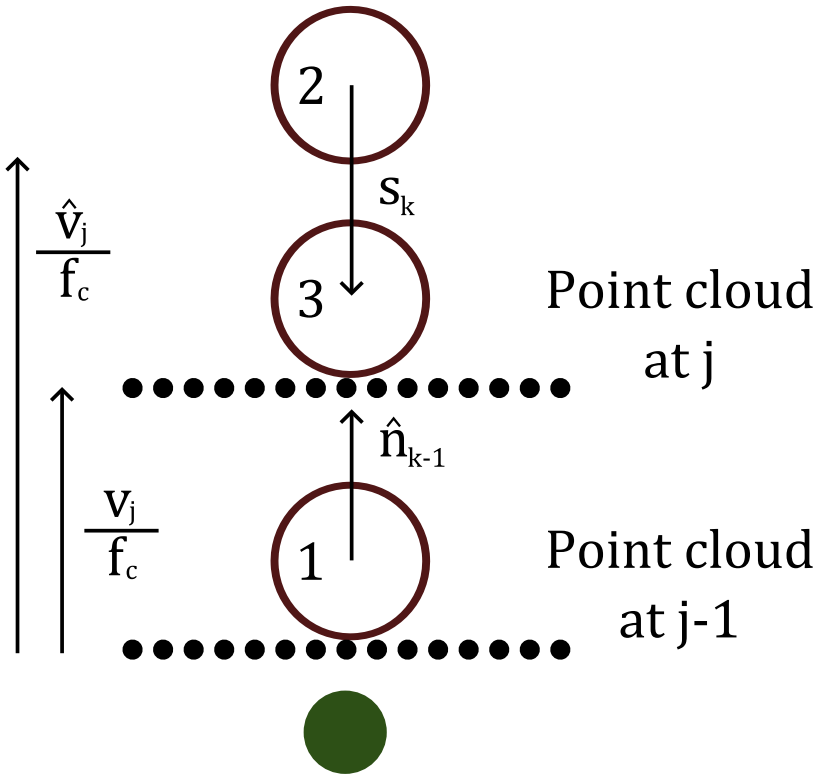


Figure 4.8: 1) The proxy in contact at proxy step $k-1$. 2) Before proxy step $k$ the point cloud updates and the velocity vector $\hat{\mathbf{v}}_\mathbf{j}/f_c$ is added to the proxy position. This causes the proxy to move too far since the magnitude of the true velocity was less than the estimated velocity. 3) One iteration of the proxy movement algorithm brings the proxy down to the surface. The velocity estimate is then updated using the projection of $\mathbf{s}_k$ onto $\hat{\mathbf{n}}_{k-1}$ as given by (4.13).

For haptic interaction with a dense and perfectly flat point cloud surface moving towards the proxy, the upper bound on point cloud acceleration becomes

$$a < r_1 f_c^2 \tag{4.14}$$

This way of estimating the point cloud velocity assumes small deviations in the point cloud trajectory between each frame. This assumption is valid for point clouds sampled at a sufficient rate.

### *4.2.6 Force Rendering*

The force on the haptic device is calculated once every 1 $ms$ as a virtual spring-damper coupling between the HIP and the contact region of the proxy. Let $\mathbf{u}_l$ be the vector between the most recent proxy and the HIP (the most recent $\mathbf{u}_k$ at the time of force rendering). If this vector would be used in the force rendering, a stuttering force would be sent to the haptic device upon interacting with point clouds moving with higher velocities. This is because the proxy moves along the estimated velocity vector at every point cloud update ($f_c$ times per second). By adjusting $\mathbf{u}_l$ for the estimated velocity, the estimated vector between the proxy and the HIP can be written as

$$\hat{\mathbf{u}}_l = \mathbf{u}_l + (t - t_j)\hat{\mathbf{v}}_j \tag{4.15}$$

where $t_j$ is the time at which the most recent point cloud was updated. The force on the haptic device can then be calculated as

$$\mathbf{F_l} = -\frac{\hat{\mathbf{u}}_l}{\|\hat{\mathbf{u}}_l\|_2}\left(k_s\left(\|\hat{\mathbf{u}}_l\|_2 - r_c\right) - k_d f_h(\|\hat{\mathbf{u}}_l\|_2 - \|\hat{\mathbf{u}}_{l-1}\|_2)\right) \tag{4.16}$$

where $f_h$ is the rate of which the force is rendered and the following term should be interpreted as the one-sided finite difference of the distance between the proxy and the HIP (a velocity). This force is sent to the haptic device *only* when the HIP is outside the radius $r_c$ of the proxy and only when the proxy is in contact (see Fig. 4.9). As a result, the user will feel a force only after the HIP has passed through the points rather than directly when the proxy becomes constrained by the points. In practice, the computation of 4.16 is done

in a separate thread running at a fixed $1000Hz$ rate. This has to be done since convergence can not be guaranteed in an arbitrary point cloud environment.

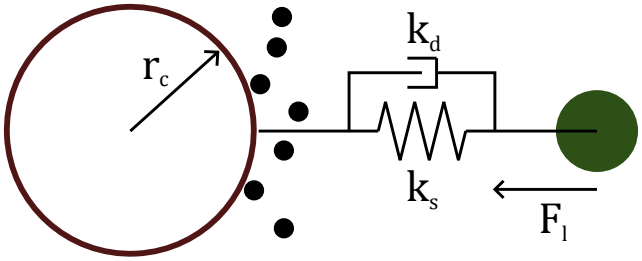


Figure 4.9: Illustration of the force acting on the HIP that is being sent to the haptic device. The force is calculated as if there was a virtual spring-damper coupling between the HIP and the proxy.

### *4.3 Results*

This haptic rendering method successfully renders haptic forces upon haptic interaction with streaming point clouds. The efficient structure allows the haptic rendering to be executed on a common desktop computer. The point cloud velocity estimation presented in Section 4.2.5 significantly reduces the risk for pop through at point cloud updates. The use of the velocity estimate for force rendering in Section 4.2.6 smooths the force sent to the haptic device. The method successfully renders haptic forces for interaction with sharp corners. Haptic interaction with concave shapes can be accurately rendered if the curvature is less than the curvature of the proxy.

The performance of this haptic rendering method is evaluated in a virtual 3D environment simulated on a computer (Intel i7-2675QM CPU with AMD Radeon HD 6750M GPU) running Windows 7. The HIP in this environment was controlled by either a SensAble PHANToM Omni haptic device or by automating the HIP movement (for evaluation). The point cloud can either be imported from a RGB-D camera or by sampling an implicit function.

The correctness of our haptic rendering method is evaluated by constructing a synthetic point cloud with grid spacing $s_{xz} = 0.3r_c = 0.0015m$ (which is roughly what we would

expect at $1m$ distance from an RGB-D camera) which is updated at $f_c = 30Hz$. These values were chosen to match typical properties of point clouds derived from commercially available RGB-D cameras. Since these synthetic datasets are not derived from a depth camera, a brute-force search was used to find a normal vector, proxy state and step size. The HIP is automated in this point cloud to simulate haptic interaction for purposes of observing the proxy movement under different point cloud conditions. These evaluations focus on the proxy movement rather than the force sent to the haptic device with the assumption that correct proxy movements result in a correct force on the haptic device. Finally, the feasibility of haptic rendering with an RGB-D camera is shown using point clouds from an Xbox Kinect.

The parameters used in the evaluation are shown in Table 4.2. The results in this section are dependent on these parameters and in particular dependent on the three radii associated with the proxy. The results are therefore normalized using $r_c = 0.005m$. That is, the results for the synthetic dataset holds for varying $r_c$ as long as the radii $r_{1,2,3}$ as well as the grid spacing are scaled accordingly. In experiments it was found that no damping was needed in the virtual coupling, since the damping in the haptic device was sufficient. In this test implementation, the proxy movement algorithm does not stop upon convergence, this since it is not feasible to start and stop threads at the required rate on a non real-time operating system.

### *4.3.1 Flat Point Cloud Surface*

The HIP is first automated on a 100x100 point cloud (see Fig. 4.10) such that it moves with a constant velocity of $2r_c$ per second ($v = 0.01m/s$) in positive tangential direction to the plane. The proxy is first moved $1s$ above the points, followed by $3s$ below the points and then again $1s$ above the points. The y-coordinates of the proxy and the HIP is shown in the top trace of in Fig. 4.11. The bottom trace shows a closer look at the proxy trajectory; with this zoom level a periodic pattern can be observed. This is expected since the proxy movement algorithm moves the proxy with respect to the points and not to an assumed underlying surface. This can be thought of as the proxy *squeezing* in between the points.

Table 4.2: Parameters used in experimental evaluation

| | |
|---|---|
| $r_c$ | $5 \cdot 10^{-3}m$ |
| Proxy radius $r_1$ | $r_c - 0.01 \cdot 10^{-3}$ |
| Proxy radius $r_2$ | $r_c + 0.01 \cdot 10^{-3}$ |
| Proxy radius $r_3$ | $2r_c$ |
| Convergence constant $\theta$ | $0.1\pi/180$ |
| Step size scaling factor $\gamma$ | $3/10$ |
| Step size scaling factor $\xi$ | $1/10$ |
| Spring constant $k_s$ | $600N/m$ |
| Damping constant $k_d$ | $0Ns/m$ |
| Point grid spacing $s_{xz}$ | $0.3r_c$ |
| Point cloud update rate $f_c$ | $30Hz$ |

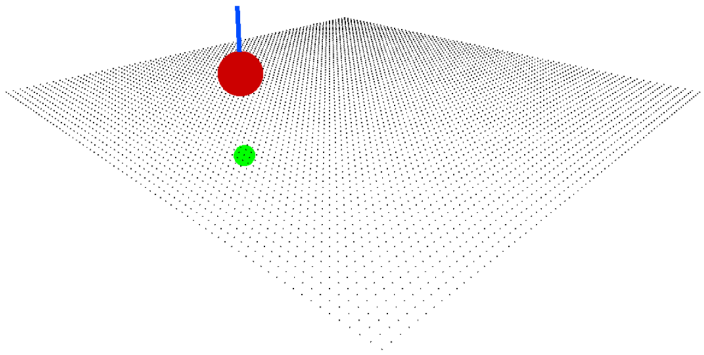

Figure 4.10: Haptic interaction with a 100x100 point cloud grid with equal spacing. The larger sphere representing the proxy and the smaller sphere below the point cloud representing the position of the HIP. The pin sticking out of the proxy illustrates the estimated normal vector.

The amplitude of this periodic pattern is $0.01r_c = 5 \cdot 10^{-5}m$. The step-like trajectory arises from the fact that the contact region allows the proxy to become slightly entrenched (or

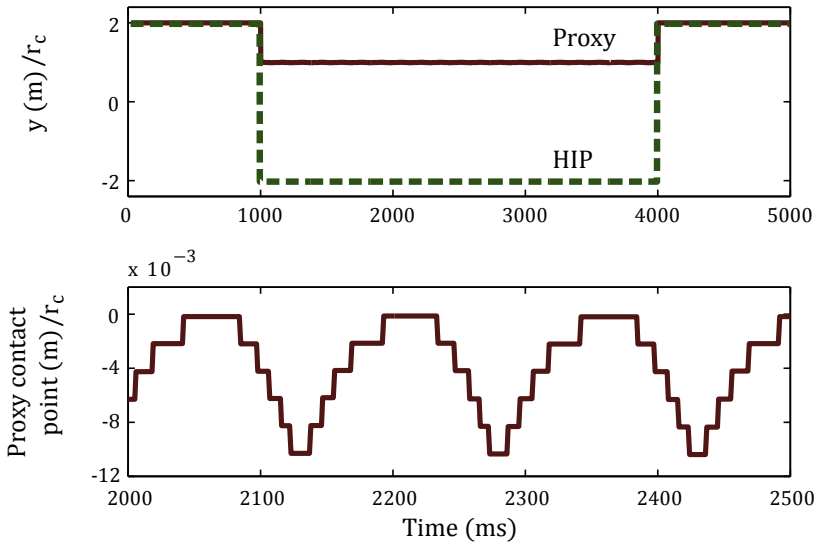


Figure 4.11: To validate that the presented method results in a correct haptic proxy, the HIP is automated on an 100x100 point cloud grid. The top trace shows the y-coordinate of the proxy and the HIP as the HIP is moved at a constant velocity of $2r_c$ per second ($v = 0.01m/s$) in positive tangential direction to the plane. After $1000ms$ the HIP moves below the surface and then after $4000ms$, it moves back up again. The bottom trace shows the y-coordinate of the proxy contact point which shows a periodic pattern with amplitude $0.01r_c = 5 \cdot 10^{-5}m$.

in free motion) before correcting and moving up (or down). Upon haptic interaction using a haptic device, the surface feels like a frictionless plane. When the HIP makes a discrete jump at $1000ms$, it takes the proxy less than $1ms$ to move down to the point set surface. Hence, a correct force can be sent to the haptic device in the next haptic cycle (after $1ms$). This means that this method of haptic rendering meets the timing goal of *Challenge 1* in Section 4.1.1.

### *4.3.2 Moving Point Cloud Surface*

The performance of the proxy in contact with moving point clouds is evaluated by constructing a 200x200 flat point cloud with varying acceleration and noise. Initially, the proxy and the HIP are placed at the origin and the point cloud is centered $10r_c = 0.05m$ below the origin. The point cloud is then lifted (in the direction of the normal vector) with

constant velocity $20r_c$ per second ($v_0 = 0.1m/s$) until it reaches the origin, after which a constant acceleration $a$ is applied for 10 frames (1/3 second) (see Fig. 4.12). To simulate measurement noise from an RGB-D camera, zero-mean Gaussian noise (standard deviation $\sigma$) is applied to each Cartesian coordinate of each point. During the acceleration, the HIP is kept $6r_c = 0.03m$ below the point cloud surface. If the proxy is located below the point cloud after the lift is finished, we denote it a pop through. This procedure was repeated 5396 times for different values of point cloud acceleration and point cloud noise. The result is shown in Fig. 4.13 where a filled square indicates that a pop through *did not* occur. These results are a major improvement compared to [1] where velocities above $16r_c$ per second ($v = 0.08m/s$) resulted in pop-through. The goals regarding moving point clouds in *Challenge 2* in Section 4.1.2 are therefore met.

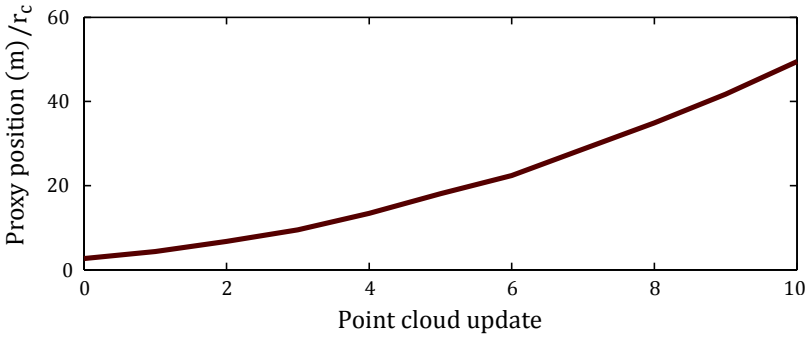


Figure 4.12: The proxy y-coordinate for each point cloud update when the point cloud is lifted with constant acceleration of $600r_c$ ($a = 3m/s^2$) and under the influence of noise with $\sigma = 0.6r_c = 0.003$.

### *4.3.3 Point Cloud Representing Sharp Corners*

The accuracy of this haptic rendering method is evaluated by moving the HIP in a 300x100 grid with the shape of a triangle wave with increasing amplitude and observing how well the proxy can track this surface. Fig. 4.14 shows how the proxy tracks the surface over a few convex and concave corners. As can be seen, the convex part is rendered very well as a sharp edge. For the concave part of the wave the rendering is ultimately dependent on the size of the proxy. When moving into a narrow concave shape, the proxy becomes

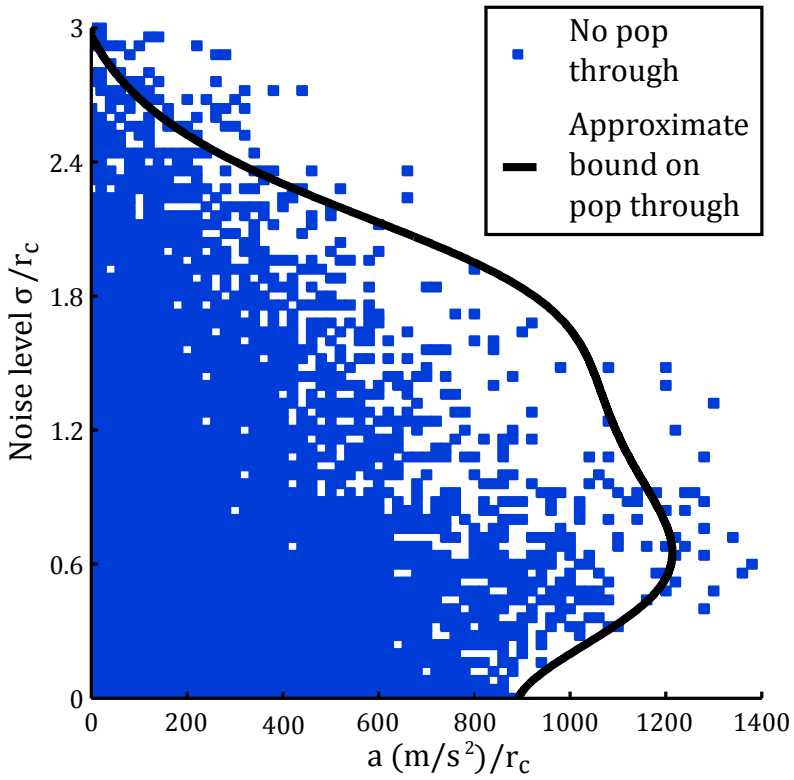


Figure 4.13: The result of 5396 point cloud lifts for varying values of noise level $\sigma$ and acceleration $a$. The filled squares indicates a successful lift, the white portion of the plot illustrates the values for which the proxy popped through the point cloud. The solid line illustrates the mathematically derived approximate bound on pop through obtained in Section 4.4.2. The axes are scaled by $r_c$.

constrained by the points on the sides and therefore the proxy can not move to the bottom of the triangle wave. The goals of *Challenge 3* in Section 4.1.3 are met since narrow features and sharp corners can be distinguished.

### *4.3.4 Streaming Point Cloud from an RGB-D camera*

Haptic interaction with a streaming point cloud is evaluated using an Xbox Kinect RGB-D camera capturing point clouds at $f_c = 30Hz$. Fig. 4.15 shows a point cloud representation of a physical object on a table where the RGB data is superimposed on the depth data. For purposes of evaluating the accuracy of haptic interaction with a noisy point cloud, the HIP

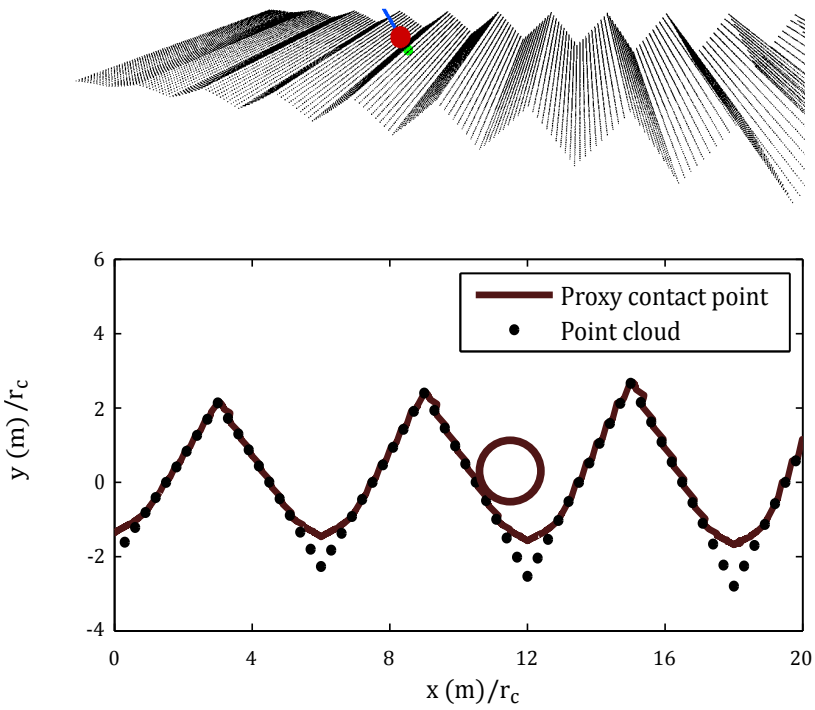


Figure 4.14: A 300x100 point set surface shaped as a triangle wave with increasing amplitude (top). A two dimensional cross section of the point cloud (dots) and the contact point of the proxy (line) for a small portion of this triangle wave (bottom). The line ultimately illustrates the surface the user will feel when interacting with this point cloud. The circle has radius $r_c$.

was moved at constant velocity in positive z-direction. The top trace of Fig. 4.16 shows the proxy trajectory in the y-z plane as the proxy moves with respect to the point cloud representation of the physical object. The bottom trace shows the magnitude of the force sent to the haptic device for each HIP position.

Upon haptic interaction using a SensAble PHANToM Omni haptic device, the force sent to the haptic device remains stable and without oscillations despite of the evident measurement noise in the point cloud data. The use of the velocity estimate in Section 4.2.6 results in smooth haptic interaction with moving point clouds even when the proxy

moves in large discrete steps at every point cloud update. The method successfully renders forces at different locations in the scene (with different point cloud resolutions) as long as the point cloud is dense enough for the proxy not to fall in between points. The proxy typically converges in $1-2$ proxy steps each time the haptic device moves. The proxy movement algorithm typically runs at a maximum rate of $30kHz$ (this includes finding proxy state, normal vector as well as step size).

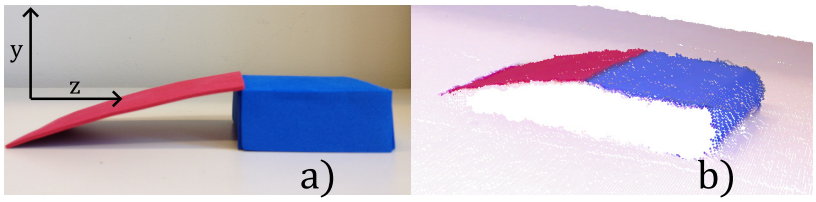


Figure 4.15: A physical object consisting of a ramp and a box is captured by the Xbox Kinect camera for purposes of evaluating haptic interaction with streaming point clouds. a) A photo of the physical objects. The camera was located outside the right upper corner of this photo. b) The point cloud representation of the same scene as captured by the Kinect camera.

The speed of the algorithm scales approximately linearly with the number of points in the neighborhood of the proxy as collision detection, normal vector calculation and the search for step size are the computationally most intensive parts of the algorithm. In actual implementation, these three are performed in the same loop where each individual task involves a brute-force search of the neighborhood around the proxy. This will probably be the main limitation of a 6-DOF version as the virtual tool typically will be larger than the small spherical proxy in this work. If the virtual tool is larger than the spherical proxy used here, the computational load of this 6 DOF haptic rendering algorithm will increase. The force is rendered independently of the proxy algorithm as the rate is fixed at 1000 Hz.

## *4.4 Analysis*

### *4.4.1 A bound on pop through*

In Section 4.3.2, it was shown that our haptic rendering method significantly reduces the risk for pop through upon haptic interaction with noisy point clouds moving at high accel-

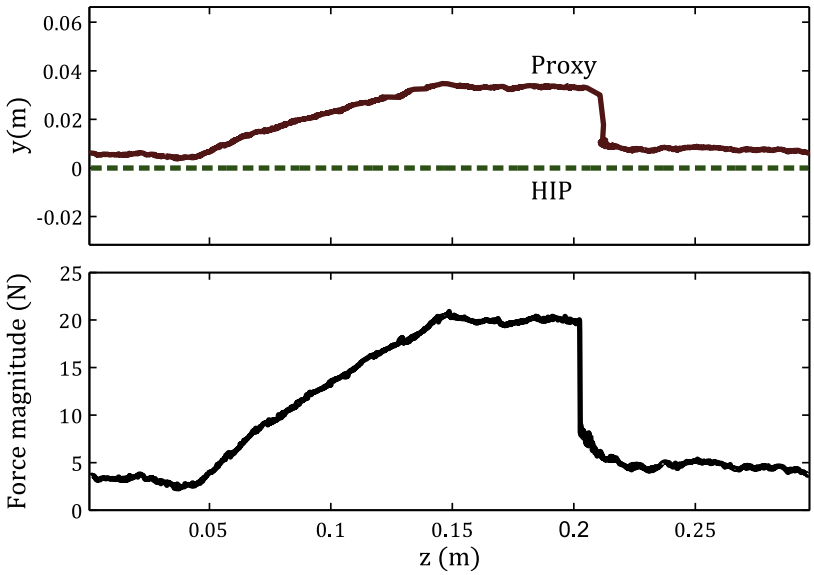


Figure 4.16: The top trace shows the z-coordinates plotted versus the y-coordinates for the proxy and the HIP when moving in the point cloud representation of the physical object. The bottom trace shows the magnitude of the force sent to the haptic device.

erations. Under the influence of noise, the condition for pop through becomes a bit more complex than described by (4.14). Noise affects the pop through problem mainly in two ways. First because it results in an erroneous normal vector estimate. An error that propagates to the proxy movement algorithm where it results in an incorrect point cloud velocity correction. Secondly, it might cause the point cloud to become thicker, which reduces the risk for pop through. In fact, for a given estimated velocity vector $\hat{\mathbf{v}}$ and a point cloud thickness at $j-1$ (denoted $h_{j-1}$), correct haptic interaction with respect to pop through can be expected if

$$\frac{\hat{\mathbf{v}}_j^T \mathbf{v}_j}{f_c \|\mathbf{v}_j\|_2} + r_1 + h_{j-1} > \frac{v_j}{f_c} \tag{4.17}$$

This inequality is illustrated in Fig. 4.17.

For a point cloud moving in the direction of the true normal vector, (4.17) can be rewritten in terms of the true and estimated normal vector as well as the point cloud

acceleration $a_j$ (since $v_j = v_{j-1} + \frac{a_j}{f_c}$) which yields

$$r_1 + h_{j-1} > \frac{a_j}{f_c^2} + \frac{1}{f_c}\left(v_{j-1} - \hat{v}_j \mathbf{n}_{k-1}^T \hat{\mathbf{n}}_{k-1}\right) \tag{4.18}$$

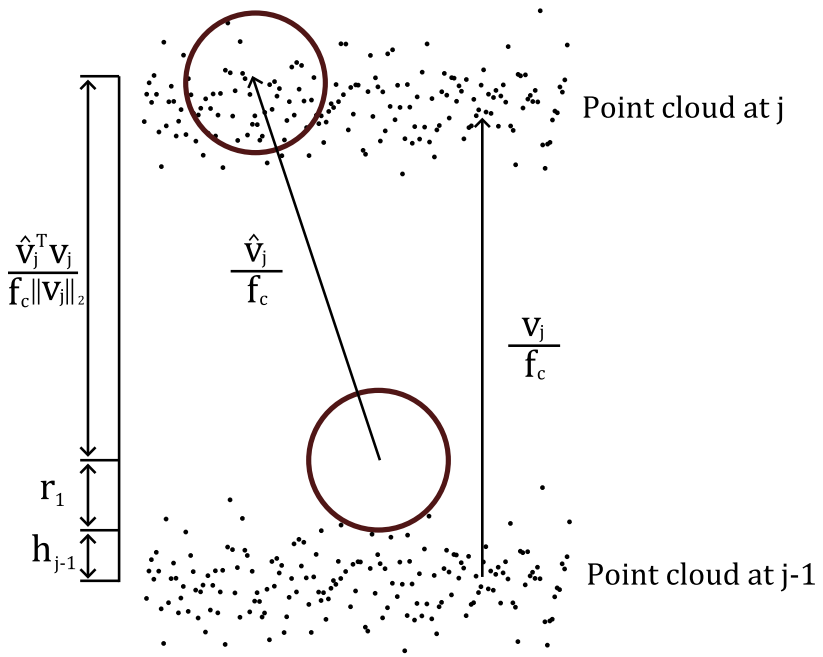


Figure 4.17: The proxy will pop through the point cloud if the sum of the true velocity is greater than the sum of the estimated velocity projected onto the true velocity vector, the inner radius $r_1$ of the proxy and the thickness of the point cloud (denoted $h_{j-1}$).

### *4.4.2 Validation of the pop through bound*

To validate (4.17) we compare it with the experimental results from Section 4.3.2 after making some simplifying assumptions. First, note that (4.18) gives a bound on pop-through for a given point cloud update $j$ rather than a sequence of point cloud updates as in Section 4.3.2 (acceleration during 10 frames). By observing (4.18), one can argue that the risk for pop through increases with velocity as the error in the normal vector estimate becomes dominant. Therefore, only the risk for pop through at the point cloud update with the highest velocity (the 10th frame) will be considered in this analysis.

Secondly, the quantities of $h_{j-1}$, $\hat{\mathbf{n}}_{k-1}$ and $\hat{v}_j$ are unknown and values of these have to be estimated or chosen.

To get some intuition on how to choose them, consider the following sequence of events leading up to a pop through:

1. The normal vector estimate calculated just before point cloud update $j-1$ is close to the actual normal vector and the proxy does not pop through at this point cloud update.

2. The resulting velocity estimate is very good, that is

$$\hat{v}_j \approx v_{j-1} \tag{4.19}$$

3. Due to noise, the normal vector estimate right before point cloud update $j$ is poor and points in an erroneous direction.

4. The resulting proxy step at point cloud update $j$ ($\mathbf{s}_j$) will therefore be in the wrong direction. As a result, the proxy will not be moved to the right position and a pop through will most likely occur.

With this in mind, $\hat{v}$ at the 10th frame should for a given constant acceleration $a$ be chosen as

$$\hat{v} = \max_j v_{j-1} = v_9 = \frac{1}{10} + \frac{9a}{30} \tag{4.20}$$

since $v_0 = 0.1m/s$ was used in Section 4.3.2.

By rewriting (4.18) using (4.19) we get (by dropping the subscripts $j$ and $k$ since only pop through at the 10th point cloud update is considered)

$$r_1 + h > \frac{a}{f_c^2} + \frac{\hat{v}}{f_c}\left(1 - \mathbf{n}^T\hat{\mathbf{n}}\right) \tag{4.21}$$

In (4.21), $\mathbf{n}^T\hat{\mathbf{n}}$ should be interpreted as the length of the estimated normal vector projected onto the true normal vector which makes it convenient to define

$$e_{\hat{n}} = 1 - \mathbf{n}^T\hat{\mathbf{n}} \tag{4.22}$$

where $h$ and $e_{\hat{n}}$ can be thought of as functions of the random variables associated with the noise on each point. The expected values of $h$ and $e_{\hat{n}}$ can be obtained as functions of $\sigma$ by simulating point cloud lifts with zero acceleration and varying noise levels between $\sigma = 0$ and $\sigma = 2r_c = 0.01$ (since pop through never occurred in this interval in the experiment). By simulating point cloud lifts in this way and applying least-square fits to the expected values, the estimates are found to be

$$\hat{h}(\sigma) = (5.5 \cdot 10^4)\sigma^4 - (6.7 \cdot 10^3)\sigma^3 + 63\sigma^2 + \sigma \\ -4.1 \cdot 10^{-5} \tag{4.23}$$

and

$$\hat{e}_{\hat{n}}(\sigma) = (2.9 \cdot 10^{10})\sigma^5 - (6.4 \cdot 10^8)\sigma^4 + (4.8 \cdot 10^6)\sigma^3 \\ -(1.2 \cdot 10^4)\sigma^2 + 16\sigma \tag{4.24}$$

Using these estimates we can approximately say that a pop through in Section 4.3.2 should not have occurred if

$$r_1 + \hat{h}(\sigma) > \frac{a}{f_c^2} + \frac{\hat{v}}{f_c}\hat{e}_{\hat{n}}(\sigma) \tag{4.25}$$

We can evaluate the accuracy of this analysis and of (4.17) in Fig. 4.13 by comparing the approximate pop through bound on the results from the experimental results. The approximate bound is a bit loose partly because of noise as well as errors in the least square estimates $\hat{h}(\sigma)$ and $\hat{e}_{\hat{n}}(\sigma)$, but mainly because our simplified model only takes into account pop throughs that occur at the 10th frame (and not in frames leading up to the 10th frame). Note that the bound above $\sigma = 2r_c = 0.01$ is a bit more conservative because $\hat{h}(\sigma)$ and $\hat{e}_{\hat{n}}(\sigma)$ were obtained using measurements only up to $\sigma = 0.01$. The risk for pop through increases for increasing velocities as $\hat{e}_{\hat{n}}(\sigma)$ gets magnified. The risk for pop through decreases as $f_c$ increases since the discontinuities between each frame becomes smaller. An exact bound on pop through could theoretically be obtained using (4.18) if the quantities of $h_{j-1}$, $\hat{\mathbf{n}}_{k-1}$ and $\hat{v}$ were known exactly for each point cloud update.

### *4.5 Conclusion*

We have presented a haptic rendering method for streaming point cloud data by extending a traditional *virtual coupling*-based proxy method. The proxy moves iteratively with respect to the points without any need to assume a sufficiently sampled point cloud or pre-processing of points. The use of a variable proxy step size results in better accuracy for short proxy movements and faster convergence for longer movements. The method presented provides highly accurate haptic interaction for geometries in which the proxy can fit. This means that the challenges of timing as well as accuracy are resolved. This novel point cloud velocity estimation significantly reduces the risk for pop through upon haptic interaction with point clouds moving under the influence of noise. Thus this method resolves the challenge of haptic interaction with point clouds moving at high velocities. The haptic rendering method presented is very computationally efficient and can run on a Intel Core i7 or equivalent personal computer.

Chapter 5

# RENDERING OF 3-DOF VIRTUAL FIXTURES FROM STREAMING POINT CLOUDS [1]

## *5.1 Introduction*

For a remote controlled robot, a virtual fixture is to the robot what the ruler is to the pen. A virtual fixture is software for constraining or guiding a tele-operated robot in a pre-defined region. A forbidden-region virtual fixture is a method (algorithm and software) for prohibiting a robot end effector from entering a certain region. In prior work in the literature concerning virtual fixtures, these regions of interest had to be defined prior to operation by construction of mesh shapes or regions defined by implicit functions. In the method developed here, these prior computations are not required.

The use of RGB-D cameras has recently gained popularity in the field of robotics. These cameras capture RGB data as well as depth data which can be interpreted as a streaming point cloud. Constraint-based haptic rendering of dense streaming point clouds from RGB-D data (using a proxy) has been investigated by several groups and reliable methods exist.

Assuming that the object that needs to be protected is in the field of view of the RGB-D camera, the idea is to define a spherical virtual fixture around each point associated with the object of interest. The novelty in this work is an efficient method for enforcing forbidden-region virtual fixtures defined by point clouds. We modify our existing proxy method for haptic rendering of streaming point cloud data [1]. This is done without any need for pre-processing of points and without any information regarding surface properties such as surface normals. The feasibility of the method is shown by protecting a beating heart using a forbidden-region virtual fixture. This method can be implemented on any

---

[1] F. RYDÉN, AND H.J. CHIZECK. "FORBIDDEN-REGION VIRTUAL FIXTURES FROM STREAMING POINT CLOUDS: REMOTELY TOUCHING AND PROTECTING A BEATING HEART." IN INTELLIGENT ROBOTS AND SYSTEMS (IROS), 2012 IEEE/RSJ INTERNATIONAL CONFERENCE ON, PP. 3308-3313. IEEE, 2012. 

teleoperated robot equipped with a sensor capturing point clouds.

### *5.2 Forbidden-Region Virtual Fixtures defined by Point Clouds*

The idea of a forbidden-region virtual fixture can be combined with the idea of a proxy-based haptic rendering method for streaming point clouds by letting a virtual fixture be defined by multiple spherical shapes defined around each point of interest. The HIP then represents the position of the master console (the commanded robot position). The position of the slave robot can then be chosen as either the proxy position or the HIP position. If the latter is chosen, the master will be able to override the virtual fixture by resisting the force exerted by the master console (haptic device). Fig. 5.1 illustrates the proxy when constrained by the forbidden-region virtual fixture as defined by multiple points. The force on the HIP can then be calculated as a virtual coupling between the HIP and the proxy.

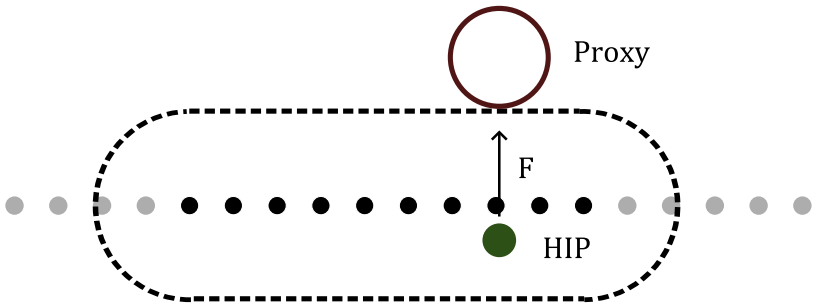


Figure 5.1: A two-dimensional illustration of a forbidden-region virtual fixture (the dashed line) defined by the black points. The proxy is constrained by the virtual fixture and can only move on the boundary of the forbidden region. The haptic interface point (HIP) shows the position of the master. The arrow illustrates the force on the HIP (which is sent to the master console).

In [98], the points within a region of interest were selected and offset by a fixed distance to prohibit movements closer than that distance. One drawback with this method is that the distance constraint only can be enforced correctly in one direction. An easy solution to this would be to create additional points at a fixed radius from the points of interest and then up-sample to get the sufficient point cloud density. This is not a good idea since many points would be required for large forbidden-regions. This would greatly increase the

computational requirements of the algorithm.

Instead, consider defining individual forbidden-region virtual fixtures around each point that needs to be protected. Let

$$\mathbf{p}_i,\ i = 1, 2, \ldots, N \tag{5.1}$$

be the points in the point cloud. Assign each point with a corresponding forbidden-region radius

$$r_{f,i},\ i = 1, 2, \ldots, N \tag{5.2}$$

as well as a corresponding stiffness

$$k_{s,i},\ i = 1, 2, \ldots, N \tag{5.3}$$

Fig. 5.2 shows a two-dimensional illustration of the resulting forbidden-region virtual fixture (the dashed line) when 3 points are assigned with the same radius. The proxy will ultimately be prohibited from moving inside the forbidden-region.

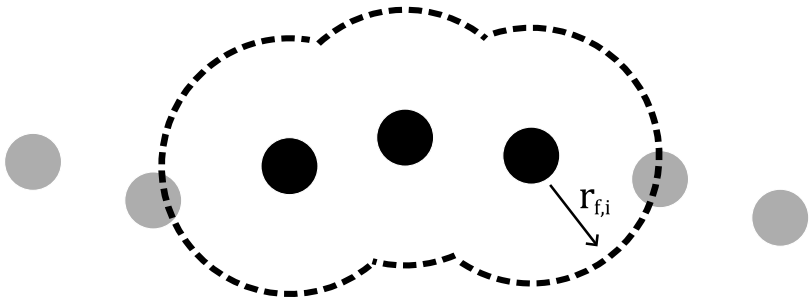


Figure 5.2: A two-dimensional illustration of the resulting forbidden-region virtual fixture obtained when superimposing the spherical virtual fixture associated with each of the three points.

To prohibit the proxy from penetrating the virtual fixture we start by thinking of the forbidden regions defined by the points as spheres and then only allowing the proxy to move on the surface of these spheres. Therefore, every time before the proxy moves, a normal vector estimate is obtained. Thus, only movements on or away from the plane defined by

the estimated normal vector are allowed. Due to the linearization in the proxy movement, the proxy sometimes will become entrenched (penetrating the forbidden region). Since this happens very rapidly and is corrected for in the next proxy step, it should be considered an intermediate state and not used for force rendering or position control. The resulting stiffness is then calculated as a weighted average of the active constraints.

The method for enforcing the forbidden region virtual fixtures can be summarized as follows:

1. Capture a dense point cloud using an RGB-D camera. If the point cloud is sparse (relatively the size of the proxy), up-sampling techniques can be used, but this will degrade the accuracy of the rendering.

2. Use some method (such as object recognition) for assigning values of $r_{f,i}$ and $k_{s,i}$ to each point in the point cloud.

3. Move the proxy iteratively with respect to the virtual fixture constraints. This is done by at every step determine the proxy state, estimate a normal vector, determine a step size and only allow movements on or away from an estimated plane.

4. Send a force to the master console (haptic device) based on the distance between the proxy and the HIP.

5. Set the position of the slave robot to either the proxy position or the HIP position.

This work builds upon the work on point cloud haptic rendering in [1], [3]. However the method presented in that work cannot be applied to virtual fixtures without appropriate modifications. The modifications required to render virtual fixtures are presented below.

### *5.2.1 Determining Proxy State*

We will use the notion of the proxy being in either *free motion*, *in contact* or *entrenched* as well as a proxy being defined by three radii $r_1$, $r_2$ and $r_3$ (see Fig. 5.3) extending out from the proxy position (denoted $\mathbf{P}$). The contact point is defined to be in the middle

between $r_1$ and $r_2$ such that $r_c = (r_1 + r_2)/2$. For a proxy in an environment with multiple forbidden-region virtual fixtures, these states will have to be interpreted a bit differently. In this work, the proxy is said to be in free motion if the proxy is not on the boundary or inside *any* forbidden-regions. The proxy is said to be in contact if one (or more) forbidden-regions boundaries are inside $r_2$ but no boundaries are inside $r_1$ of the proxy. The proxy is said to be entrenched if one or more forbidden-region boundaries are inside $r_1$ of the proxy.

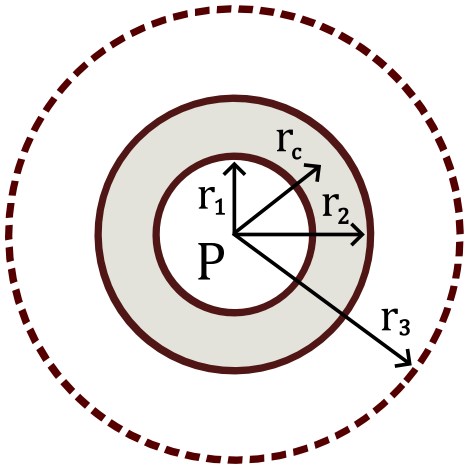


Figure 5.3: 2-dimensional cross-section of the proxy. Proxy regions are defined by $r_1 < r_2 < r_3$. [1]

### *5.2.2 Obtaining a Normal Vector Estimate and a Virtual Fixture Stiffness*

Every time before the proxy moves, a normal vector estimate will be obtained as a weighted average of the directions to the forbidden regions in the neighborhood of the proxy. That is, every time before the proxy moves, calculate a normal vector estimate according to the following steps

1. Let $\mathbf{p}_i,\ i = 1, 2, \ldots, M$ be the points satisfying

$$r_{n,i} \leq r_3 \tag{5.4}$$

where

$$r_{n,i} = \|\mathbf{P} - \mathbf{p}_i\|_2 - r_{f,i} \tag{5.5}$$

As a result, this set will consist of all the points whose constraints are active, plus some extra points (to yield a smoother normal vector estimate). These extra points will only be used in the normal vector estimate and not for calculating a step size for the proxy.

2. The points $\mathbf{p}_i$ will be weighted using a modified version of the Wendland function [114] such that

$$\psi(r_{n,i}) = \begin{cases} 1 \text{ if } r_{n,i} \leq r_1 \\ \left(1 - \frac{r_{n,i}-r_1}{r_3-r_1}\right)^4 \left(\frac{4(r_{n,i}-r_1)}{r_3-r_1} + 1\right) \\ \text{if } r_1 < r_{n,i} < r_3 \\ 0 \text{ if } r_{n,i} \geq r_3 \end{cases} \tag{5.6}$$

This weighting function was used for haptic rendering in [3] and a similar function was initially suggested for haptic rendering in [84]. This function is smoothly decreasing between $r_1$ and $r_3$ which results in a smoother normal vector estimated as new points are being introduced.

3. The normal vector can then be calculated in the same fashion as in [3]

$$\hat{\mathbf{n}} = \frac{\sum_{i=1}^{M} \frac{\mathbf{P}-\mathbf{p}_i}{\|\mathbf{P}-\mathbf{p}_i\|_2} \psi(r_{n,i})}{M \sum_{i=1}^{M} \psi(r_{n,i})} \tag{5.7}$$

Here the denominator terms are simply a way to normalize $\hat{\mathbf{n}}$.

4. Using the same idea, the weighted virtual fixture stiffness will be calculated as

$$k_s = \frac{\sum_{i=1}^{M} k_{s,i} \psi(r_{n,i})}{M \sum_{i=1}^{M} \psi(r_{n,i)}} \tag{5.8}$$

### *5.2.3 Determining an Optimal Proxy Step Size*

In [1] the proxy was moved with fixed steps of a small size. In [3] the proxy was moved with respect to the points in the neighborhood of the proxy. Now, consider the following efficient method for moving the proxy with respect to the forbidden-region virtual fixtures in the neighborhood of the proxy.

For the case when the proxy is in free motion and a forbidden-region virtual fixture is intersecting the straight line trajectory between the proxy and the HIP, it is desirable to move directly to the virtual fixture in one step. Let $\mathbf{p}_i,\ i = 1, 2, \ldots, M$ be the set of points in the neighborhood of the proxy. It is important to select this neighborhood large enough such that any points that could constrain the proxy movement are taken into account. The largest step size the proxy can move without violating any constraints can then be found as $d_{free-motion} = \min_i d_i$ where $d_i$ is found by solving

$$\|\mathbf{P} - \mathbf{p}_i + d_i \frac{\mathbf{u}}{\|\mathbf{u}\|_2}\|_2 = r_{f,i} + r_c \tag{5.9}$$

and only considering real solutions (since an imaginary solution implies that the forbidden region associated with the $i$-th point is not constraining the proxy path); $\mathbf{u}$ in (5.9) denotes the vector between the proxy and the HIP. The geometrical interpretation of $d_i$ is illustrated in Fig. 5.4.

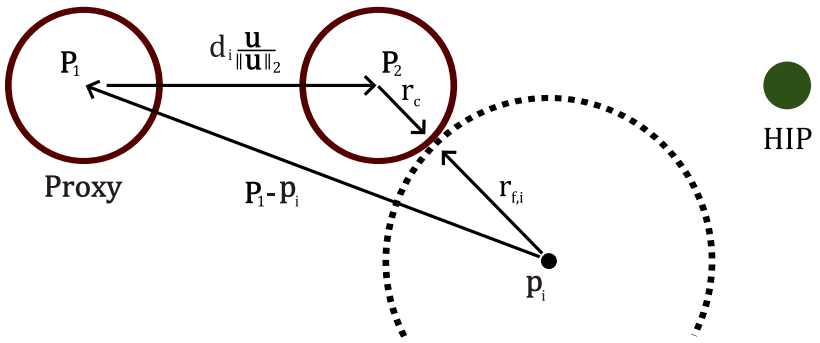


Figure 5.4: A two-dimensional illustration of (5.9). $d_i$ is the largest step that can be taken towards the HIP from $P_1$ without violating the virtual fixture region defined by $\mathbf{p}_i$ and $r_{f,i}$. The resulting proxy position after this step will be $P_2$.

For the case when the proxy is entrenched, let $\mathbf{p}_i,\ i = 1, 2, \ldots, M$ be the points satisfying

$$r_{n,i} < r_1 \tag{5.10}$$

where $r_{n,i}$ is defined by (5.5). These points are the points that either are entrenching the proxy or whose forbidden-regions are entrenching the proxy. The desired step size can then be found as $d_{entrenched} = \max_i d_i$ where $d_i$ is found by solving

$$\|\mathbf{P} - \mathbf{p}_i + d_i\hat{\mathbf{n}}\|_2 = r_{f,i} + r_c \tag{5.11}$$

where $d_i$ is the step size required to move the proxy out of the $i$-th constraint. By selecting $d_{entrenched}$ as the maximum of all $d_i$, the proxy is guaranteed to move out of all constraints in one step.

Since modifications have been made to the calculation of the normal vector and the definitions of when the proxy is contact, the step size from [3] can be used directly. That is, if $\|\mathbf{u}_p\|_2 \leq r_1$ (where $\mathbf{u}_p$ is the vector $\mathbf{u}$ projected onto the plane defined by the estimated normal vector) then choose

$$d_{in-contact} = \gamma\|\mathbf{u}_p\|_2 \tag{5.12}$$

Otherwise, if $\|\mathbf{u}_p\|_2 > r_1$ choose

$$d_{in-contact} = \xi r_1 \tag{5.13}$$

where $\gamma$ and $\xi$ are design parameters affecting rate of convergence and accuracy.

### *5.2.4 The Proxy Movement Algorithm*

The proxy movement algorithm [1], [3] can be used directly since appropriate modifications have been made to the definition of states, the normal vector estimate, the calculation of stiffness and the calculation of step size. A short summary of that algorithm is listed here for completeness. The following algorithm starts when the proxy is updated or when the HIP moves:

1. Determine the proxy state.

2. If the proxy is in free motion, move the proxy one step

$$\mathbf{s} = \frac{\mathbf{u}}{\|\mathbf{u}\|_2} d_{free-motion} \tag{5.14}$$

where $d_{free-motion}$ is the step size determined in Section 5.2.3.

3. If the proxy is entrenched, move the proxy one step

$$\mathbf{s} = \hat{\mathbf{n}} d_{entrenched} \tag{5.15}$$

4. If the proxy is in contact and the HIP is inside the estimated surface, project $\mathbf{u}$ on the plane defined by $\hat{\mathbf{n}}$ (denote this vector $\mathbf{u}_p$). Then move the proxy one step

$$\mathbf{s} = \frac{\mathbf{u}_p}{\|\mathbf{u}_p\|_2} d_{in-contact} \tag{5.16}$$

5. If the proxy is in contact and the HIP is outside the estimated surface, move the proxy one step

$$\mathbf{s} = (r_2 - r_1)\,\hat{\mathbf{n}} \tag{5.17}$$

which moves the proxy out of contact and into free motion.

6. Iterate steps 1 to 7 until the proxy has converged. If the proxy is in contact, it has converged if

$$\cos^{-1}\left(\frac{-\mathbf{u}^T\hat{\mathbf{n}}}{\|\mathbf{u}\|_2}\right) < \theta \tag{5.18}$$

where $\theta$ is a constant representing a small angle between $-\mathbf{u}$ and $\hat{\mathbf{n}}$. If the proxy is in free motion, it has converged if the HIP position equals the proxy position.

An entrenched proxy should not be considered in force rendering or when commanding the slave robot, since it is in fact violating the forbidden-region constraint. Since it will be corrected in the next proxy step, this will introduce only a very small latency.

### *5.2.5 Force Rendering*

The force on the haptic device can then be rendered as a virtual coupling between the HIP and the contact point on the proxy using the stiffness $k_s$ from (5.8). Additional details regarding this can be found in [1]. Note that the force can be rendered using any function resulting in a force on the HIP. For instance, it would be trivial to implement the results from [115]. Stability analysis is not in the scope of this paper, but any analysis for traditional proxy or 'god-object' haptic rendering is also valid for this work.

## *5.3 Results and Discussion*

The method presented in this paper successfully constrains the proxy to stay on the boundaries of the forbidden-region virtual fixtures. This is evaluated in simulation by controlling the HIP using a PHANTOM (Sensable Inc.) haptic device and then observing the proxy movement under different conditions. First, the method is verified using a point cloud with a single point. Then we show how the proxy can transition between virtual fixtures with different properties. Finally, we show using the Xbox Kinect that the presented method can be used to generate a virtual fixture protecting a beating heart in real-time.

In the evaluations the following parameters were used: $r_1 = 0.00499m$, $r_2 = 0.00501m$, $r_3 = 0.01m$, $\gamma = 0.3$, $\xi = r_1/3$, $\theta = \pi/180$.

### *5.3.1 Single Point Virtual Fixture*

To illustrate that the presented method results in a proxy that does not violate any virtual fixture constraints we construct a point cloud consisting of a single point $\mathbf{p}_0$ placed at the origin. This point is assigned virtual fixture radius $r_{f,0} = 0.1m$. Fig. 5.5 shows the distance between the proxy and $\mathbf{p}_0$ (which should be greater or equal to $r_{f,0} + r_c = 0.1 + 0.005$) as well as the distance between the HIP and $\mathbf{p}_0$. As can be seen, the proxy never moves closer than $0.1m$ to $\mathbf{p}_0$.

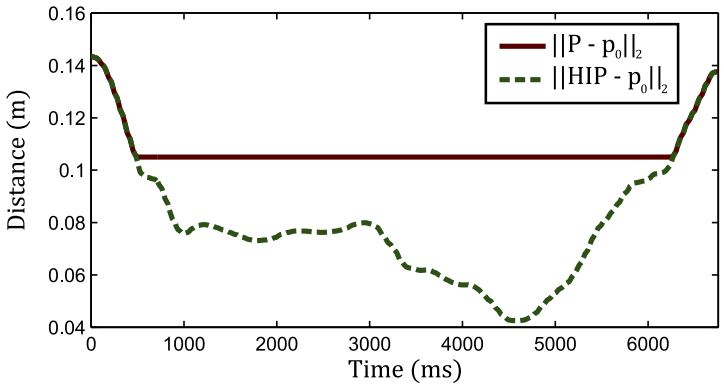


Figure 5.5: The solid line shows the distance between the proxy and the point cloud (consisting of a single point at the origin). The dashed line shows the distance between the HIP and the same point. The proxy never penetrates the spherical virtual fixture region defined by the point. When the HIP penetrates the virtual fixture region, a force based on the distance between the proxy and the HIP is sent to the master console.

### *5.3.2 Virtual Fixtures with Varying Radius*

To illustrate the transition between virtual fixtures with different properties, a 100x100 flat point cloud was constructed (in the X-Z-plane). For $x < 0$ all virtual fixture radii were set to $r_{f,i} = 0m$ and $k_{s,i} = 600N/m$. For $x \geq 0$ all radii were set to $r_{f,i} = 0.01m$ and $k_{s,i} = 600N/m$. The top trace of Fig. 5.6 shows the location of the proxy contact point when the HIP was moved with an automated constant velocity in the positive x-direction in the plane (at $y = 0$). The bottom trace shows the norm of the rendered force for each x-value. As can be seen, the force builds up before the proxy *pops up* on the virtual fixture.

### *5.3.3 Virtual Fixtures Protecting a Beating Heart*

To evaluate the feasibility of the forbidden-region virtual fixtures developed in this paper in a more realistic setting we use RGB-D data of a beating pig heart. Fig. 5.7 shows the setup for capturing the 640x480 2.5D point cloud at $30Hz$ using Xbox Kinect (Microsoft Corp.). A fixed heart rate was maintained using the Ramphal Cardiac Surgery Simulator (Univ. of the West Indies). For simplicity, all the points in the resulting point cloud were assigned

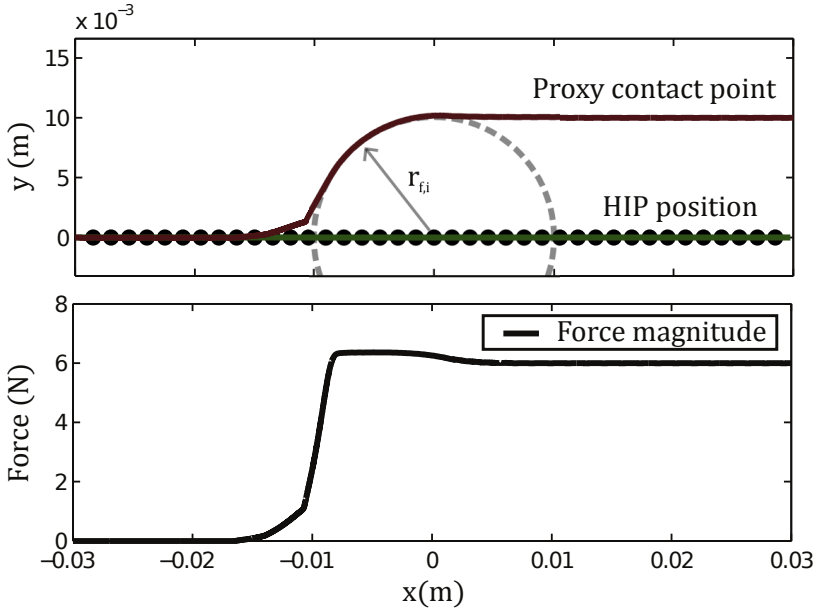


Figure 5.6: The top trace shows a two-dimensional cross-section (X-Y) of the proxy contact point and the HIP position while moving on the virtual fixture with varying stiffness. The dashed circle illustrates a single spherical virtual fixture at $x = 0m$ and the dots illustrates the location of the points in the point cloud. The bottom trace shows the magnitude of the force sent to the HIP for a given x-coordinate.

with the same virtual fixture radius. The HIP was placed manually (using a haptic device) just inside the point cloud representation of the beating heart. The proxy position was then recorded for $15s$. For the first $5s$, the virtual fixture radius was set to zero. Between $5s$ and $10s$ the virtual fixture radius was increased linearly from $r_{f,i} = 0m$ to $r_{f,i} = 0.02m$. After $10s$, the virtual fixture radius was kept at $r_{f,i} = 0.02m$. Fig.5.8 shows the y-coordinate of the proxy versus time as the radius of the virtual fixture was varied. This illustrates how we can transition from point cloud haptic rendering (touching points) to point cloud forbidden-region virtual fixtures (protecting points) using the exact same method. Fig. 5.9 shows the proxy while maintaining $0.02m$ distance to the point cloud representation of the beating heart. The video accompanied with this paper shows haptic- and virtual fixture rendering of the beating pig heart.


Figure 5.7: Capturing the RGB-D data. The Kinect camera (A) was located directly above and looking down at the heart (B). This position and orientation was chosen since it resulted in a very good depth estimation (a dense point cloud). The haptic device is not shown in this figure.

### *5.4 Conclusion*

We have presented a method for implementing forbidden-region virtual fixtures defined by streaming point clouds using a proxy method by placing a spherical virtual fixture around each point. Therefore, the virtual fixtures are implicitly defined by the point cloud as it is

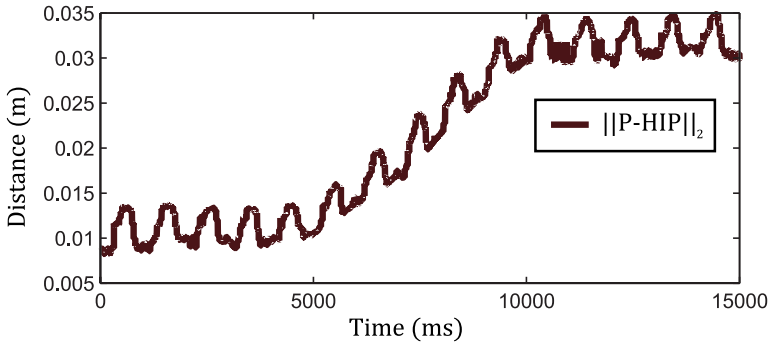


Figure 5.8: The distance between the proxy and the HIP as the proxy moves with the beating heart. Initially, the virtual fixture region was set to zero which means that the proxy was allowed to be in contact with, but not move through the heart. After $5s$, the virtual fixture radius was increased linearly from $r_{f,i} = 0m$ to $r_{f,i} = 0.02m$ in $5s$. The virtual fixture radius was then kept at $r_{f,i} = 0.02m$. The increasing offset in the trace is due to the increasing virtual fixture radius. The increasing virtual fixture radius illustrates the transition from haptic rendering to rendering of virtual fixtures.

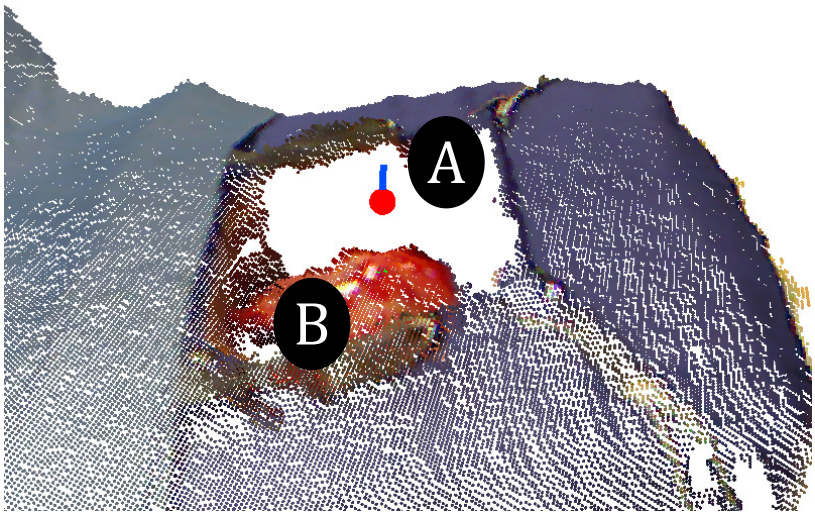


Figure 5.9: The proxy (A) in the point cloud while maintaining $0.02m$ distance to the beating heart (B). The pin sticking out of the proxy illustrates the estimated normal vector to the virtual fixture.

being updated. This method allows for new approaches to force feedback in tele-operation tasks. We show feasibility by implementing a forbidden-region virtual fixture protecting a

beating heart.

### *5.5 Acknowledgments*

Our thanks to Dr. Thomas S. Lendvay, Samuel Park and Cameron St. Hilaire at UW ISIS for their time and assistance with the beating heart. We would also like to thank Professor Blake Hannaford, Sina Nia Kosari and Nicklas A. Gustafsson for their insights and suggestions.

Chapter 6

# SIX DEGREE-OF-FREEDOM HAPTIC INTERACTION WITH STREAMING POINT CLOUDS [1]

## *6.1 Introduction*

Haptic interaction has traditionally been a purely virtual task. But recent advancements in depth sensing have made it possible to stream depth information in real-time and at low cost. There have been multiple examples demonstrating haptic interaction with a streaming depth image (or a streaming point cloud) in three degrees-of-freedom (3-DOF), but to the best of the authors' knowledge, there exists no six degree-of-freedom (6-DOF) haptic rendering method for streaming point cloud data.

6-DOF haptic interaction with streaming point clouds could be used for applications where a 6-DOF haptic rendering device, such as the delta.6 (Force Dimension) is used. This includes situations where the object being *remotely touched* can be moving. For example, consider remotely petting a dog, as in Fig 6.1.

When haptic interaction is combined with manual control of a robot (or robotic end effector) in co-robotic tasks, such as tele-robotic surgery, a 6-DOF capability could add versatility and precision to this co-robotic interaction. For example, it has also recently been shown that the techniques of point cloud haptic rendering with 3 DOFs can be useful in tele-robotics for implementation of virtual fixtures. That is, the user receives a force when the tele-operated robot is near collision with the environment. A 6-DOF haptic rendering would be preferable in this scenario since the entire end-effector could be used as the virtual tool, rather than just modeling the tool tip as a small 3-DOF sphere. A 6-DOF haptic rendering method could be useful even though many haptic devices, such as the Phantom

---

[1] F. RYDÉN, AND H.J. CHIZECK. "A METHOD FOR CONSTRAINT-BASED SIX DEGREE-OF-FREEDOM HAPTIC INTERACTION WITH STREAMING POINT CLOUDS" IN ROBOTICS AND AUTOMATION (ICRA), 2013 IEEE INTERNATIONAL CONFERENCE ON. IEEE, 2013. 

Omni (Sensable Inc.), only support 3-DOF actuation (but has 6-DOF sensing).

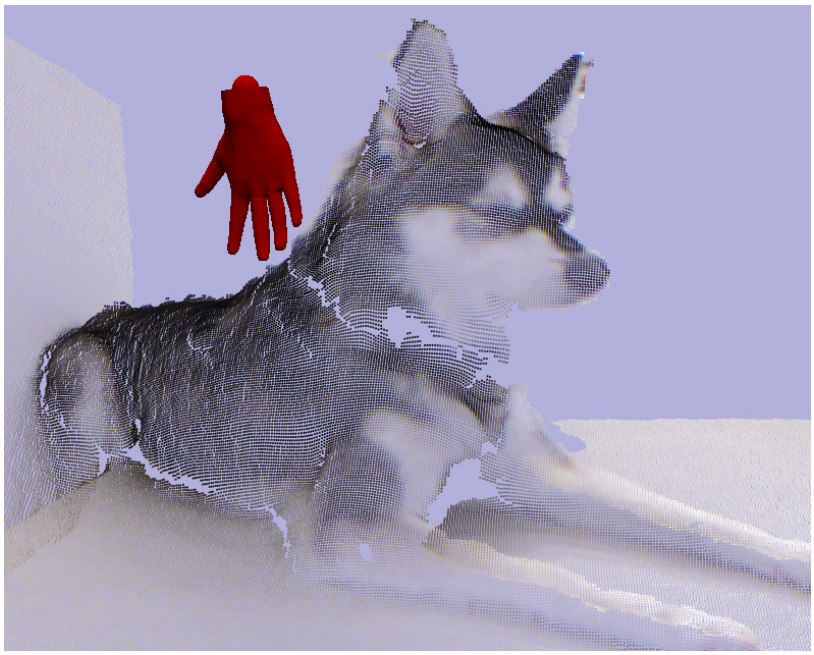

Figure 6.1: An application to six degree-of-freedom haptic rendering from streaming point clouds: Petting a point cloud representation of a dog captured using the Kinect. The virtual tool representing the haptic device in the virtual environment has the form of a human hand.

In this work, we demonstrate a constraint-based haptic rendering method for interaction between a voxelized polygon and a streaming point cloud. Since no surface information (other than color information) can be directly obtained from depth sensor systems, we only consider the geometry of the point cloud. That is, the point cloud is considered to be a set of fixed and infinitely stiff points in space.

The main contributions of this work can be summarized as:

1. A novel haptic interaction method that allows for direct interaction between the tool and the point cloud without first converting the point cloud to an intermediate data structure, thus allowing for faster rendering.

2. A method for real-time 6-DOF haptic interaction with discontinuous streaming point clouds derived from depth sensors by iteratively resolving collisions at each point cloud

update.

### *6.1.1 Haptic Interaction with Streaming Point Clouds*

To the best of the authors' knowledge, there exists no prior work where 6-DOF haptic rendering has been applied to streaming point clouds. The recent literature includes several approaches to haptic interaction with streaming point clouds for 3-DOF [1], [84], [85] and [2]. In these papers, the haptic interaction is obtained without pre-processing the point cloud (for example, converting it into a mesh). The main approach has been to simulate the virtual tool as a sphere (often referred to as a *proxy*) or by implicitly filling in the gaps and represent the virtual tool as a point (a *god-object*). The transition to 6-DOF rendering has been challenging mainly because of the computational requirements involved in the rigid body simulation with a potentially large point cloud. An initial step was taken in [86] where an operator of a robot was provided with 6-DOF haptic feedback based on static point cloud data derived from a depth image (using the approach in [71]).

## *6.2 Method*

The task of producing realistic forces for six degree-of-freedom haptic interaction is essentially the task of performing a rigid-body simulation of a virtual object at a very high rate. In the 6-DOF literature, this virtual object is referred to as a *virtual tool* (rather than a HIP). It is controlled by the user of the haptic device during interaction with the virtual environment. More specifically, the virtual tool matches the position and orientation (the configuration) of the haptic device while avoiding collisions in the virtual environment. A common approach to this problem is an iterative quasi-static simulation where the virtual tool, for purposes of calculating its constrained motion while in contact, is considered to be at rest at every time step.

In a virtual environment consisting of a point cloud, the virtual tool can be in either one of the following three states:

1. *Free motion* when no points are constraining the virtual tool.

2. *In contact* when there are points on the boundary of the virtual tool but no points are penetrating the tool.

3. *In collision* when there are points penetrating the virtual tool.

It can be determined if the virtual tool is in the first state simply by running a collision detection algorithm. If a collision is detected, the virtual tool can be either in contact or in collision. When the virtual tool is in contact, the points of contact are used to form motion constraints for the virtual tool. These constraints are then used to compute a constrained motion that will move the virtual tool towards the configuration of the haptic device without violating any contact constraints. If the virtual tool is in collision, the collision can be resolved by moving away from the set of collision constraints.

When a new depth image (or depth frame) is captured, it is transformed into a point cloud where every point in Cartesian space corresponds to a pixel. The point cloud is then filtered and a surface normal is calculated for every point in the point cloud. It should be noted that this normal estimation is different from the one performed for 3-DOF point cloud haptic interaction in [1] where the normal vector of any given point was determined by its location relative to the spherical tool. The benefit of the approach in this paper is that a collision constraint always points in the correct direction, even if the point cloud surface is touched from the back side. Finally, the force on the haptic device is calculated as a function of the *kinetic distance* between the configuration of the virtual tool and the haptic device.

### *6.2.1 Real-Time Processing of the Streaming Point Cloud*

Depth data can be captured using depth sensors such as Kinect (Microsoft Corp.). The Kinect captures data at 30 Hz and stores the depth measurements in a depth image where every pixel corresponds to a depth value. The pixel coordinates combined with the depth value can then be used to calculate Cartesian coordinates for every point (using a transformation usually found in the camera specification).

With the points transformed to a Cartesian frame, all depth values are filtered using a bilateral filter similar in fashion to [116]. The points are weighted using a Gaussian kernel

(with standard deviation $\sigma_f$) as a function of Euclidean distances in a Cartesian frame. This is done to preserve depth discontinuities as a neighboring pixel only contributes to the filtered value if its depth value is sufficiently close.

A normal vector is then calculated by fitting a plane to the points in a small neighborhood around each point. The neighboring points are weighted using the smooth Wendland weighting function [114] (with radius $r_w$) as done in [84]. This weighted total least squares problem has a closed form solution as it can be solved with a $3 \times 3$ eigenvalue decomposition for every point. This results in two solutions where the one pointing towards the depth sensor is chosen.

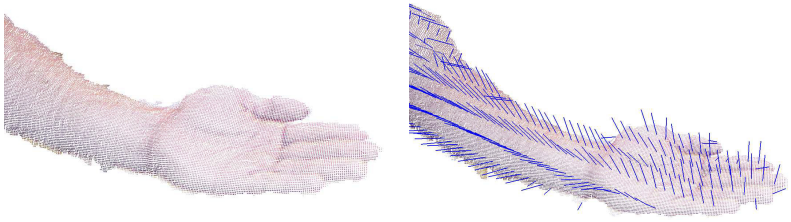

Figure 6.2: A human hand captured using Kinect and visualized as a point cloud (Left). The same human hand now augmented with normal vectors for each point (Right). A normal vector is calculated for every point in the point cloud, but only every 100th normal vector is shown for clarity.

This per-pixel data parallel work is ideal for graphics processors (GPUs). In this work, the transformation, filtering and eigenvalue decomposition are all done for every pixel in parallel using OpenCL (Khronos Group Inc.). Several optimization techniques can be used to speed this process. For instance a batch of pixels can be copied to local GPU memory and be processed in parallel by a work group of computational units that leverages from 1-2 orders of magnitude faster memory access. This is especially useful in filtering operations where many neighboring pixels are accessed. It should be noted that some redundant work is done due to the overlapping nature of a filtering operation. This redundancy can be minimized using a *loop unrolling* technique, but currently this is difficult because of local memory limitations (typically 32- or 64 Kb).

### *6.2.2 Collision Detection with the Virtual Tool*

In order for the virtual tool to move with respect to the points in the point cloud, a quick collision detection method is needed. In this work, the virtual tool is voxelized (on a $0.5mm \times 0.5mm \times 0.5mm$ grid) and stored in local memory using a simple scan-line technique. To find out if a specific point in the point cloud is colliding with the virtual tool, we simply transform the point into the tool frame and then query this point against the three-dimensional voxel matrix (which essentially is a collision/no collision lookup table). A more memory-efficient method can be used for storing the voxel data, such as a sphere-tree data structure as in [67]. This was however not necessary in this work as the RAM was more than sufficient to store the voxel data (less than a few hundred megabytes). It should be noted that this approach to collision detection is the dual of the algorithm presented in [50], where the virtual tool was represented by points and the virtual environment by voxels.

It can be very time-consuming to search the whole point cloud for potential collisions. In this work, we approximated the process by only checking for collision with points in the neighborhood of the virtual tool. These neighboring points are found using a technique similar to [1] where a two-dimensional bounding box is placed around the projection of the virtual tool onto the depth image. This approach assumes that the virtual tool is smaller than the span of the point cloud.

### *6.2.3 Finding a Constrained Motion while In Contact*

Using this collision detection method, the virtual tool is moved towards the configuration of the haptic device and stopped at the first point of contact. To find a feasible movement for the virtual tool, that will not violate the contact constraint further, we apply the Gauss' least constraints principle as done for rigid body simulation in [48] and later used for 6-DOF polygon haptic rendering in [69]. A brief description of that method is given here, but we refer to [48] for a complete derivation. The key concept is the use of a *quasi-static* simulation which greatly simplifies the calculations as the velocity is considered to be zero at every time step.

Let $\mathbf{a_g}$ be the generalized acceleration such that

$$\mathbf{a_g} = \begin{pmatrix} \mathbf{a} \\ \alpha \end{pmatrix} \tag{6.1}$$

Further, denote the generalized unconstrained acceleration

$$\mathbf{a_{gu}} = \begin{pmatrix} \mathbf{a}_u \\ \alpha_{\mathrm{u}} \end{pmatrix} \tag{6.2}$$

as the acceleration that would make the configuration of the virtual tool match the configuration of the haptic device in one unit time step. However if the virtual tool is in contact, this step can not be taken without violating the point cloud constraints. That is, each point of contact (as found by the collision detection) introduces a linear constraint of the form

$$\mathbf{n}^T\mathbf{a} + (\mathbf{r} \times \mathbf{n})^T\alpha \geq d \tag{6.3}$$

where $\mathbf{r}$ is the vector from the center of mass to the contact point, $\mathbf{n}$ is the normal vector at the point of contact and $d$ is the penetration depth of the contact point (that is $d = 0$ when in perfect contact).

Given $\mathbf{a}_{gu}$, the objective is to calculate a generalized constrained acceleration $\mathbf{a}_{gc}$ which minimizes the kinetic distance between the virtual tool and the haptic device with respect to the linearized constraints. More formally, the generalized constrained acceleration is found as

$$\mathbf{a}_{gc} = \frac{1}{2}\operatorname*{argmin}_{\mathbf{a_g}}(\mathbf{a}_{gu} - \mathbf{a}_g)^T\mathbf{M}(\mathbf{a}_{gu} - \mathbf{a}_g) \tag{6.4}$$

subject to

$$\mathbf{J}\mathbf{a}_g \geq 0 \tag{6.5}$$

where $\mathbf{M}$ is the generalized mass matrix for the virtual tool and $\mathbf{J}$ is the matrix of the constraints formed by (6.3).

This is a quadratic programming problem, but as described by [69], this problem can be solved using Wilhelmsen's nearest point algorithm [117]. It should be noted that this generalized constrained acceleration is only valid in a small neighborhood around the current configuration because of the linearized contact constraints combined with the fact that any movement might introduce new constraints.

### *6.2.4 Resolving Collisions*

Object interpenetration is avoided in [70] by using interval arithmetic and in [71] by ensuring that the step size of any part on the virtual tool always is less than the smallest feature size. In this work, the latter approach is applied with the same smallest feature size chosen as half the tool voxel size. Bounding the virtual tool movement ensures that no collisions are missed in a single point cloud frame. However, this does not hold when the point cloud updates, as the locations of points in the most recent point cloud are arbitrary. Therefore, a method of resolving collisions is needed.

We propose that the problem of resolving collisions can be solved using a simplified quasi-static version of the method for resolving collisions in [48]. We first note that the desired configuration for the virtual tool is to remain at rest. That is

$$\mathbf{a}_{gu} = \mathbf{0} \tag{6.6}$$

Secondly we note that $d$ in (7.4) has to be non-zero in order for the minimization to move the virtual tool out of the constraint. The value of $d$ should ideally match the penetration depth at each point of collision. But since a planar constraint such as (6.3) only is valid in a small neighborhood of the virtual tool and since movements of the virtual tool might introduce new constraints, we argue that an approximate solution to this problem can be obtained by weighting the constraints equally and updating the configuration of the virtual tool iteratively. This approach can be summarized as

$$\mathbf{a}_{gc} = \frac{1}{2} \underset{\mathbf{a_g}}{\operatorname{argmin}} \mathbf{a}_g^T \mathbf{M} \mathbf{a}_g \tag{6.7}$$

subject to

$$\mathbf{J}\mathbf{a}_g \geq \mathbf{1} \tag{6.8}$$

The right hand side of (6.8) can be chosen as any vector with identical elements since this only will change the magnitude and not the direction of the result from (6.7).

### *6.2.5 Algorithm for moving the virtual tool*

Given that the point cloud is processed as in Section 6.2.1, the linear motion constraints can be found by the collision detection algorithm in Section 6.2.2. The minimization problem

of moving the virtual tool can then be solved with respect to these constraints using the approach in Section 6.2.3 and 6.2.4.

That is, if the virtual tool is in free motion (no contact points and no interpenetrations), the virtual tool can be moved in the direction of the unconstrained acceleration $\mathbf{a}_{gu}$ until the first contact occurs. The virtual tool is in practice moved in small discrete steps to ensure that no contact is missed, as discussed in Section 6.2.4. Upon first contact, bisection is applied as in [71] to further refine the point of contact until it is within $\epsilon_f$.

If the virtual tool is in contact, a constrained acceleration is calculated as in Section 6.2.3 which provides a feasible movement of the virtual tool. If the magnitude of this feasible movement is greater than a fixed constant $s_{max}$, the magnitude of the movement is limited to $s_{max}$ (since the constraints only are valid in a local neighborhood of the virtual tool).

Finally, if there are points in collision with the virtual tool, a constrained acceleration is calculated as in Section 6.2.4. The virtual tool is then moved in small steps such that no feature is missed (as for the case when the virtual tool is in free motion). This procedure is repeated until collision is resolved. Bisection is then again applied until the collision is resolved with an accuracy of $\epsilon_c$.

### *6.2.6 Calculating the Force*

The force sent to the haptic device is simply calculated as a function of the difference between the virtual tool and haptic device configuration. That is, the translational and rotational forces are calculated as:

$$\mathbf{f} = \mathbf{K}\mathbf{M}\mathbf{a}_{gu} \tag{6.9}$$

where $\mathbf{K}$ is a diagonal matrix containing the spring constants. The translational component of $\mathbf{f}$ corresponds to the first three elements and the rotational component (the torque) corresponds to the last three elements.

Stability/passivity of haptic systems is a well studied field, e.g., [25], [30] and we argue that those results also hold for the presented method since a virtual coupling architecture is being used.

## *6.3 Results*

This 6-DOF haptic rendering method was implemented on a desktop computer (AMD Phenom II X6 with Radeon HD 6990) running Ubuntu 11.10. The force is calculated asynchronously at $1000Hz$ in a separate thread, regardless of whether the virtual tool has reached a local minimum or not. This has to be done since no guarantees regarding convergence can be given for an iterative method of this type. During typical interaction, the collision detection algorithm ran at $15kHz$. The point cloud was filtered and normal vectors were calculated for every point on the GPU. Real-time processing was achieved using a neighborhood of $9 \times 9$ points for filtering as well as normal vector calculation. The position of the haptic device was both controlled automatically (for purposes of producing accurate results) as well as with a Sensable PHANToM Omni haptic device. Using the latter, only translational forces could be perceived by the user (since this device only provides 3 DOFs of sensation).

The correctness of the haptic rendering method is first verified using a synthetic noise free data set. The method was then evaluated using streaming point cloud data from a depth sensor (Xbox Kinect) with two different virtual tools. In the evaluation, the following parameters were used: $\sigma_f = 0.025$, $r_f = 0.014$, $\epsilon_f = \epsilon_c = 0.1mm$ and $s_{max} = 1mm$.

### *6.3.1 Interaction with a Synthetic Point Cloud*

To evaluate the presented haptic rendering method in a noise free environment, a synthetic point cloud box (with 5 sides but no top) was constructed. The normal vectors were set perpendicular to each side, pointing into the box. In this evaluation, a model of a *UW husky* was used as the virtual tool. The position of the haptic device was then automated to move 2s into the box, rotate in the positive direction for $2s$, rotate in the opposite direction for $2s$, and then move for $2s$ out of the box. The first and the second plots of Fig. 6.3 show the resulting displacement as well as rotation along the y-axis. The last plot of Fig. 6.3 shows the number of contact points (after any collision was resolved) over time. As can be seen, the virtual tool becomes constrained by the sides of the box and interacts with as many as 6 contact points simultaneously. In Fig. 6.3, the configuration of the haptic device and the

virtual tool is presented rather than the force on the haptic device. This is done with the understanding that a correct virtual tool configuration will result in a correct force on the haptic device as computed in (6.9).

### *6.3.2 Interaction with a Streaming Point Cloud*

Fig. 6.4 shows two examples of haptic interaction with streaming point cloud where arbitrary polygon models can be used as virtual tools. The performance of the haptic rendering method is ultimately dependent on the number of points in the neighborhood of the virtual tool. Most of the computational time is spent on collision detection and forming movement constraints and as a result, the algorithm scales approximately linearly in the number of neighboring points. This ultimately limits the size of the virtual tool as the neighboring set increases with increasing tool size. To improve performance in interaction with large tools, the point cloud can be down-sampled. For neighborhoods of size less than 2500 points, the haptic interaction feels transparent. To increase transparency in interaction with denser point clouds, we propose using parallel computing on the CPU, as the collision detection currently is performed on a single core. The accuracy of the rendering could also be improved using interval arithmetic as done in [69].

The attached video demonstrates interaction with point cloud representations of objects on a table as well as with the synthetic point cloud.

## *6.4 Conclusion*

We have presented a method for 6-DOF haptic rendering of moving objects, in real-time. This method renders haptic interactions using an arbitrary voxelized polygon tool and a streaming point cloud derived from a depth sensor. The depth image is captured in real-time at 30 Hz using a Kinect depth camera. Every frame is filtered and surface normals are calculated in real-time on the GPU. Using this method, a user can interact with both dynamic as well as static point cloud representations of real objects. The performance of the haptic rendering is dependent on the of the density of the point cloud, as collision detection is performed as a linear search in the neighborhood of the virtual tool.

***Acknowledgments***

The authors would like to thank Professor Blake Hannaford and graduate students Sina Nia Kosari and Timothy Haines (and his dog Kelty) for their insights and suggestions. We would also like to acknowledge Advanced Micro Devices (AMD) for providing us with the GPUs used to implement the algorithm presented in this paper.

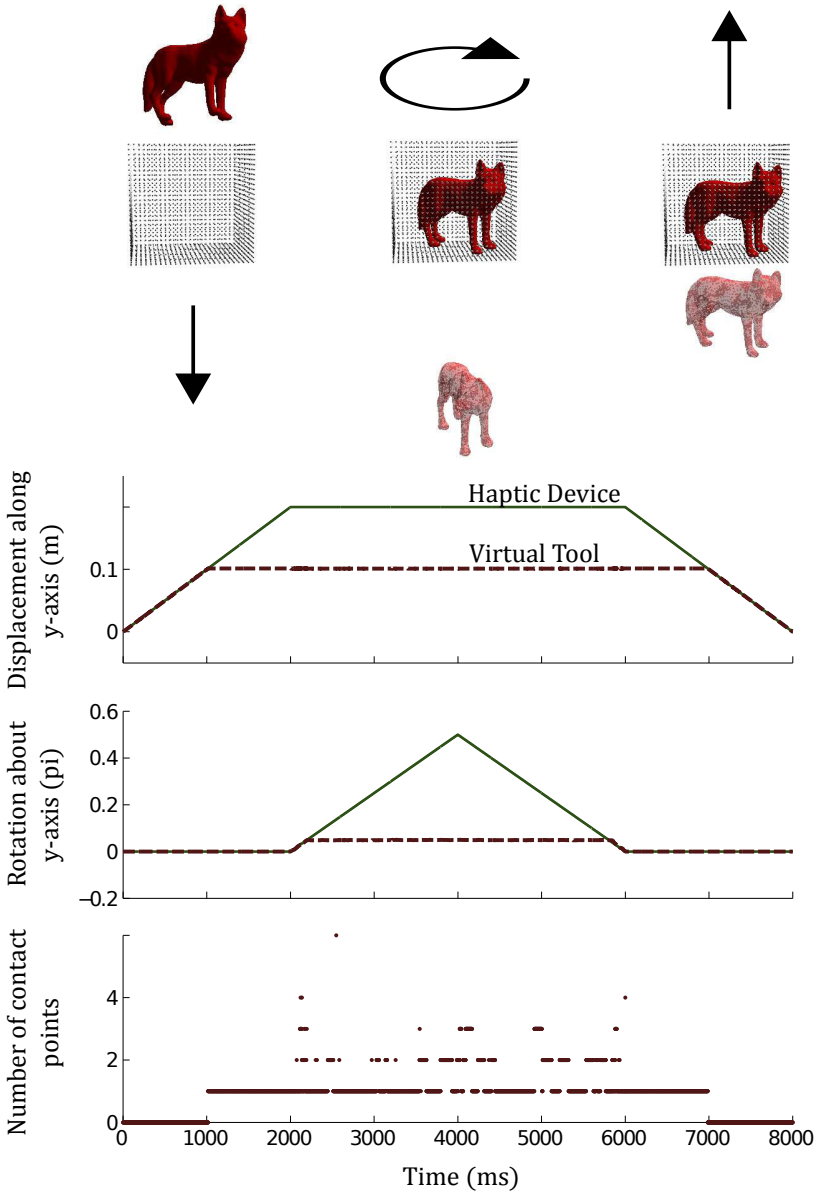


Figure 6.3: Six degree-of-freedom haptic interaction with point cloud representation of a narrow box using a *UW husky* model as a virtual tool. The top plot shows the displacement from the starting position along the y-axis for the haptic device and the virtual tool as the haptic device moves. The middle plot shows the rotation about the y-axis for the device and tool. Note that the position of the haptic device was automated for purposes of evaluation, which resulted in perfect trajectories (the solid lines). The bottom plot shows the number of contact points during interaction with the point cloud box.

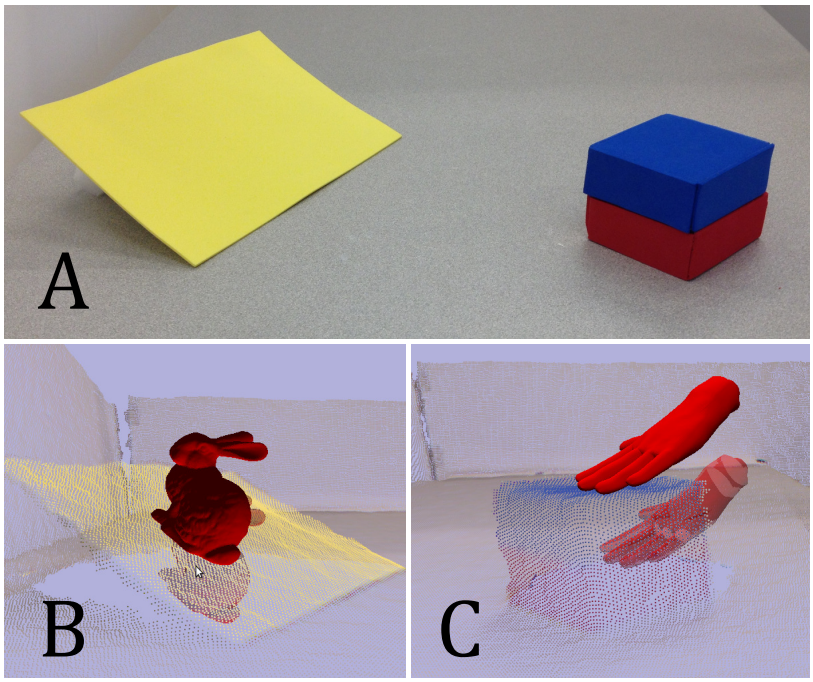


Figure 6.4: Some real world objects that are captured using an Xbox Kinect depth camera (A). Haptic interaction with a streaming point cloud using both the *Stanford bunny* (B) and a hand model (C) as virtual tool. The configuration of the virtual tool is illustrated in the virtual environment as a filled polygon and the configuration of the haptic device as a transparent polygon.

Chapter 7

# RENDERING OF SIX DEGREE-OF-FREEDOM VIRTUAL FIXTURES FOR UNDERWATER MANIPULATION [1]

## *7.1 Introduction*

When tasks must be carried out in the deep ocean or hazardous marine environments, teleoperated, robotic systems are necessary. In most applications a pilot resides at the surface while a remotely-operated vehicle (ROV) carries out functions below. Munitions remediation, oil and gas operations, studies of hydrothermal vent colonies, and construction of ocean observatories are a subset of underwater applications that require teleoperation. In particular, manipulation is often necessary for operating tools, opening valves, mating cables, or simply moving objects from one place to another. Subsea manipulation is a challenging and expensive endeavor that often relies on expert pilots with substantial experience. The cost of operating ships and robotic systems often drives crews to operate 24 hours/day and underscores the importance of avoiding mistakes. Advanced tools that lower the burden on pilots, increasing safety, reliability and efficiency of operations, has potential to dramatically reduce cost and increase feasibility of complex tasks performed underwater.

Perception is paramount to efficiently and safely carrying out complex tasks remotely. We make use of *virtual fixtures* to provide tactile feedback to a remote operator of a robotic device underwater. Haptic virtual fixtures are computer-generated constraints that limit the motion of a robot in space. Violation of these constraints is opposed by a force, communicated to the pilot by a haptic device (input device that conveys reaction forces to the user). *Forbidden regions* act as *force fields* that push back on the end effector to avoid constraints, allowing for sensitive objects to be protected. If the pilot commands the end effector into a forbidden region, a resistive force is rendered – preventing violation of the

[1] F. RYDÉN, ANDREW STEWART AND H.J. CHIZECK. "ADVANCED TELEROBOTIC UNDERWATER MANIPULATION USING VIRTUAL FIXTURES AND HAPTIC RENDERING." IN OCEANS 2013. IEEE, 2013. 

constraint. *Guidance virtual fixtures* help the pilot move toward a desired object to be manipulated.

In this work we demonstrate our ability to generate virtual fixtures in an underwater environment from non-contact (camera) sensors to provide tactile feedback to a remote pilot of a subsea manipulator. We focus on the application of remotely opening and closing valves underwater. This task is representative of operations that would be carried out by ROVs on oil rigs or in subsea infrastructure maintenance. It may be necessary to service one valve while not disturbing an adjacent valve. When valves are close in proximity this presents a challenge to the pilot. By using both guidance fixtures and forbidden regions we are able to assist the pilot in servicing a valve while avoiding undesired contact or collisions with other objects in the environment.

Our approach makes use of a 3D camera (or sonar) to create a model of the environment. The haptic rendering algorithm runs in the virtual environment to compute reaction forces which are fed to the pilot. This leverages the natural perceptive capability of pilots and simplifies remote manipulation tasks.

We note that many ROVs have been equipped with tools for haptic feedback, but suffer from limitations. The standard approach to providing tactile feedback requires contact forces to be measured *after* contact. For subsea applications, latency and a subsequent instability arise driving many ROV pilots to switch off the haptic feedback function. Even when functioning properly, a significant burden is placed on pilots as they must predict points of contact by inferring three-dimensional structure through a series of two-dimensional displays. With our approach, the pilot receives feedback before contact is made. Further, virtual fixtures can be used to assist a pilot in and around tight corners, or toward targets (i.e. handles, knobs, interfaces), as well as away from sensitive objects.

The contributions of this paper include:

- A method for rendering of six degree-of-freedom (6-DOF) forbidden-region haptic and visual virtual fixtures from streaming point cloud data. This means that virtual fixtures can be rendered with respect to the full geometry as well as configuration of the end-effector.

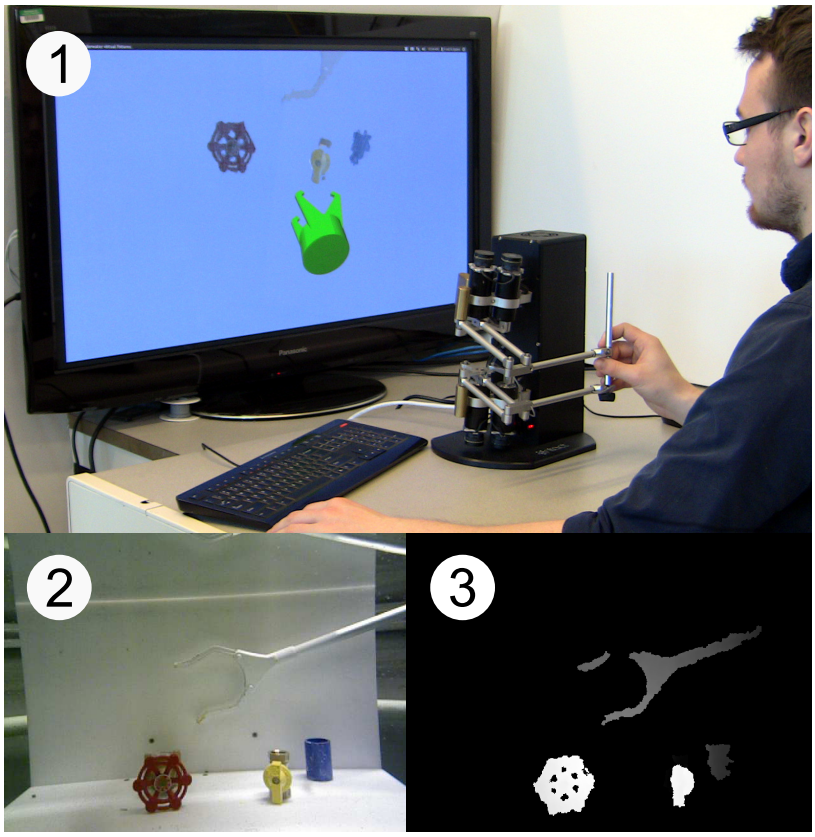


Figure 7.1: Interaction with haptic virtual fixtures generated from underwater sensor data. The operator interacts with the virtual environment using a W5D haptic device (Entact Robotics) (1). RGB data of the underwater scene (2). Depth data of the same underwater scene (3).

- A method for rendering of 6-DOF guidance virtual fixtures generated from streaming point cloud data, in combination with forbidden region virtual fixtures.

- A demonstration of virtual fixture rendering for a relevant underwater tool positioning task.

### *7.2 Six Degree-of-Freedom Haptic Virtual Fixtures in Underwater Environments*

In this paper, we present a novel method for rendering of 6-DOF haptic forbidden-region virtual fixtures and guidance virtual fixtures based on real-time sensor data. Essentially

this is done by running a real-time simulation of robot motions in the actual environment to calculate the effect of haptic virtual fixtures. This is then presented to the operator. This work extends the method of rendering 3-DOF forbidden-region virtual fixtures of [2], to the 6-DOF case. In [2], the haptic feedback was based solely on the desired position of the robotic end effector, not taking rotations or torques into account. This was done by approximating the robotic end effector as a simple bounding sphere, which is not always a good approximation. The forbidden-regions were further defined by superimposing spherical regions where each sphere corresponded to a point in the point cloud captured by the a depth sensor overlooking the robot task space.

In this work, the three-dimensional structure of the robot's task space is captured in real time. The forbidden-regions are defined in a similar fashion by finding the points that correspond to a virtual fixture and then produce a forbidden region sphere around each of those points. Rendering of haptic virtual fixtures then simply corresponds to *conventional* haptic rendering of spheres (typically a large number of time-varying spheres) using a virtual-coupling method. The guidance virtual fixture can then simply be implemented as a spring to a desired target configuration.

There are several real-world challenges in rendering of haptic virtual fixtures that depend upon properties of the robot task space. These include:

1. **Registration:** The environment around the robot needs to be captured using some type of sensor with depth sensing capabilities. This sensor further has to be registered to the robot base frame. When the sensor is mounted on the robot itself, this is trivial (as the displacement can be easily measured). When, instead, the sensor is at another location, registration markers can be used.

2. **Processing of sensor data:** The sensor data needs to be processed (transformation to base frame and filtering etc.) and the virtual fixtures must be defined in real-time. Parallel processing is typically necessary to achieve this.

3. **Defining virtual fixtures:** The virtual fixtures have to be defined implicitly by the sensor data. Therefore, a method for choosing the data that corresponds to a guidance

virtual fixture or a forbidden-region virtual fixture is needed. For instance, this can be a simple color segmentation algorithm.

4. **Speed of haptic rendering:** In order to meet the $1000Hz$ recommended haptic rendering rate, an efficient rendering method is needed. As haptic rendering can be seen as a simplified physics simulation, a fast collision detection method is required. A parallel collision detection method can be used in combination with a sphere tree data structure [118].

To generate virtual fixtures for an underwater task space, sensor data was captured underwater using an RGB-D sensor (RGB + depth). The data was transformed to the robot base frame and spatially low pass filtered. A color segmentation algorithm was used to define the haptic virtual fixtures and package them in a sphere tree data structure that could efficiently be used for haptic rendering. The haptic rendering then uses the desired configuration of the robot end effector in combination with the virtual fixtures to present the force to the operator.

As this work focuses on rendering of virtual fixtures rather than system integration on a robot system, the results here are presented in simulation. However, the objects involved are obtained from streaming point clouds from the camera. Our justification for this is that the virtual fixture rendering essentially runs a real-time simulation of the robot plus its surroundings. In the next phase of this work we will be incorporating a five degree of freedom submersible manipulator. If the registration and the simulation is correct we can assume that the robot is very close to where the operator commands it to be using the haptic device.

### *7.3 Capturing and Processing Underwater Data*

Before virtual fixtures can be constructed and haptic feedback can be given to the operator, the RGB-D sensor data has to be processed. This is done in two steps by first transforming and filtering the data and then defining the virtual fixtures.

### *7.3.1 Processing*

Since image processing (both depth and RGB) is highly parallel work, it is efficient to utilize modern graphics processors (GPUs). Our method is to first transform the depth image into a point cloud using the camera parameters and then spatially low-pass filter using a bilateral filtering approach [116] such that the depth values are weighted using the Cartesian distance to neighboring points. That is, for a given depth value $v_0$ corresponding to a point in space $\mathbf{p_0}$, the filtered depth value $v_0'$ is calculated as:

$$v_0' = \frac{\sum_{k=0}^{N-1} v_k \phi(k)}{\sum_{k=0}^{N-1} \phi(k)} \tag{7.1}$$

where N is the number of neighboring points and $\phi(k)$ are the weights such that

$$\phi(k) = e^{\frac{-(\mathbf{p_k}-\mathbf{p_0})^T(\mathbf{p_k}-\mathbf{p_0})}{2\sigma^2}} \tag{7.2}$$

where $\sigma$ is a design parameter. The spatial bilateral filtering approach is preferable as it does not induce any time delay (unlike temporal filtering).

### *7.3.2 Defining virtual fixtures*

With the data filtered and transformed into a point cloud, virtual fixtures can be assigned to either protect certain regions, or guide the operator towards certain regions. A color segmentation method is used to find the point that should be assigned to either a forbidden-region virtual fixture or a guidance virtual fixture. This is done by converting the RGB (red, green and blue) values to corresponding HSL (hue, saturation and lightness) values and then selecting points based on their hue value (which is orthogonal to the lightness value). This technique is very useful in environments with varying lighting conditions. As a result, a forbidden-region can for instance be defined around all *red* points and a guidance virtual fixture can for instance be generated based on all *yellow* points. Due to local memory constraints on typical GPUs, the data is processed in patches of 16x16 pixel which due to required overlapping results in some redundant work. This approach is chosen as it is one to two orders of magnitude faster to read from local memory than from global memory.

*Defining the Forbidden-Region Virtual Fixture*

For each point (corresponding to some physical object) that is assigned as a forbidden-region virtual fixture, a spherical forbidden region (with a radius chosen by the application) is constructed. Thus, a distance constraint can be maintained in all directions. The final forbidden-region virtual fixture is then constructed by superimposing multiple spherical regions, as was done in [2], see Fig 7.2. For purposes of rapidly being able to query this virtual fixture using a tree structure, a bounding sphere is generated around each 16x16 patch. Rather then generating a perfectly fitting sphere, a simple bounding sphere is generated such that the center is at the mean location $\tilde{\mathbf{s}}$ of the forbidden-regions and the radius such that:

$$\tilde{r} = \max_k \|\mathbf{s_k} - \tilde{\mathbf{s}}\| + r_k \tag{7.3}$$

where $\mathbf{s_k}$ and $r_k$ is the location and radius of the k-th spherical virtual fixture respectively.

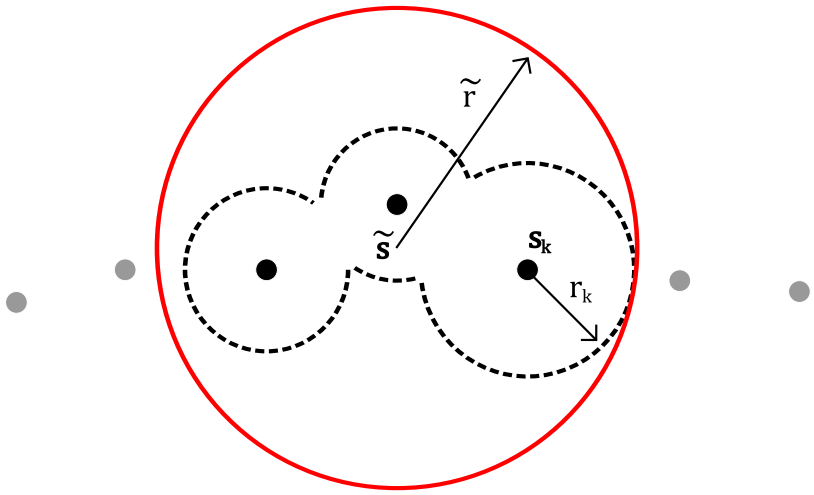


Figure 7.2: Illustration of a forbidden-region virtual fixture obtained by superimposing three spherical fixtures. The sphere defined by $\tilde{\mathbf{s}}$ and $\tilde{r}$ illustrates the bounding sphere described in (7.3).

*Defining the Guidance Virtual Fixture*

The guidance virtual fixture is implemented as a desired 6-DOF configuration (position and orientation) in space. In terms of processing, all points with a certain hue value are selected and the mean location is chosen as the location of the guidance virtual fixture. If the selected points correspond to a plane, a normal vector can be estimated and the orientation of the virtual fixture can be based on this. Otherwise, the orientation can simply be fixed relative to the robot base frame as done in this work.

## *7.4 Rendering of Haptic Virtual Fixtures*

Since the forbidden-region virtual fixture constraints are defined by superimposing spheres and since the virtual model of the robot arm can be stored in a sphere-tree (for purposes of collision detection), the most straightforward method for haptic rendering is a sphere-tree to sphere-tree haptic rendering method. The optimized sphere-tree containing the virtual tool is generated prior to operation using the Sphere Tree Toolkit [118]. The sphere-tree containing the forbidden regions is generated on-the-fly every time a frame arrives from the sensor system.

Since the haptic feedback is based on the difference (in configuration space) between the configuration of the haptic device and the configuration of an ideal virtual tool that does not penetrate any forbidden-regions, a method for moving the virtual tool is needed. Therefore, at each haptic time step ($1ms$), the virtual tool is moved iteratively towards the configuration of the haptic device. If this configuration is inside a forbidden region, the virtual tool stays at the boundary of the forbidden region and locally minimizes the *distance* (in configuration space) between the virtual tool and the configuration of the haptic device. This is done using a *quasi-static* simulation as suggested for haptic rendering by [70]. The force and torque on the haptic device is then defined by two virtual springs (one translational and one torsional) between these two configurations. When new sensor data arrives, the virtual tool might be at a configuration inside a forbidden region, in that case the virtual tool will move away from the forbidden region. This is similar to the approach for moving a voxelized tool out of point cloud constraints in [4].

The guidance virtual fixture is used to guide the operator, using force feedback, towards a desired configuration in the robot's task space. An efficient method of implementing this is to simply augment the force contributed by the forbidden-regions with the guidance force such that the operator is *pulled* towards the guidance configuration by a virtual spring.

### *7.4.1 Collision detection and forming constraints*

When the operator controls the robot (using the haptic device) and no forbidden-regions are violated, the virtual tool matches the configuration of the haptic device. But when the operators guides the robot into a forbidden region, this effectively forms a set of linear constraints on the virtual tool. These constraints can be found by recursively checking for collisions between the virtual tool (a sphere tree) and the forbidden regions (another sphere tree); denote these trees $T_{tool}$ and $T_{FR}$ respectively. This is done by first using a depth-first search of $T_{tool}$, querying against the highest level of $T_{FR}$, until the leaf node is reached. Then any occurrence of collision in $T_{FR}$ is expanded in a similar depth-first fashion by querying against the leaf nodes of $T_{tool}$. If a collision has occurred between a leaf node in $T_{tool}$ and a leaf node in $T_{FR}$, this effectively forms a linear motion constraint on the tool, of the following form:

$$\mathbf{n}^T\mathbf{a} + (\mathbf{r} \times \mathbf{n})^T\alpha \geq d \tag{7.4}$$

where $\mathbf{a}$ and $\alpha$ are the tool's unconstrained translational and rotational acceleration, respectively. $\mathbf{n}$ is the normal to the constraint, $\mathbf{r}$ is the vector from the tool's center of rotation to the point of collision and $d$ is the penetration depth (see Fig. 7.3). No constraints are formed on the velocities as the velocity is considered zero in a quasi-static simulation.

### *7.4.2 Moving the virtual tool with respect to virtual fixture constraints*

Given a haptic device configuration, a virtual tool configuration and a set of linear collision/contact constraints, the goal is to minimize the distance in configuration space between the configuration of the haptic device and the configuration of the virtual tool with respect to the virtual fixture constraints. Such a minimization process can be solved as a nearest-point problem and results in a vector pointing in a feasible direction (in configuration space).

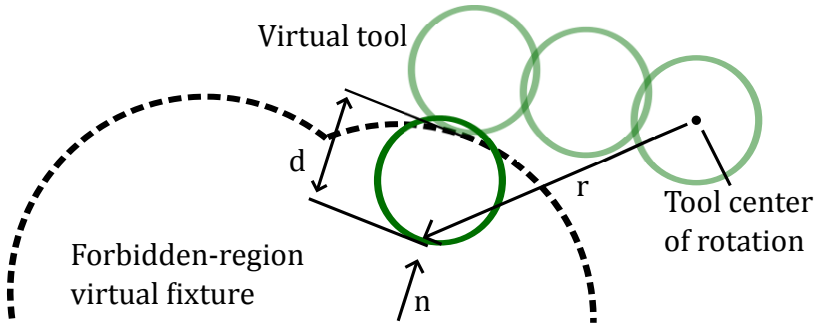


Figure 7.3: Illustration of a collision constraint between a forbidden-region virtual fixture (the dashed line) and a virtual tool constructed by a sphere tree (where solid circles represent the leaf nodes). Here **n** is the normal to the constraint, **r** is the vector from the tool's center of rotation to the point of collision and $d$ is the penetration depth. The penetration is exaggerated for clarity as the penetration typically is very small.

Unfortunately, since the constraints are linearized, this feasible direction is only valid in a small neighborhood around the tool. Therefore, this process must be repeated iteratively such that a new minimization process is performed every time a new constraint/collision occurs. The full derivation of this minimization process is outside the scope of this paper, but can be found in [48].

### *7.4.3 Calculating force feedback*

The resulting force feedback is calculated as a function of the difference between the haptic device and virtual tool configuration. That is, the force and torque are chosen as the springs that would bring the configuration of the haptic device to the configuration of the virtual tool.

## *7.5 Results*

The proposed virtual fixture rendering method was implemented on a Core i7-3930K CPU (Intel Corp.) with Radeon HD 6990 GPU (Advanced Micro Devices, Inc.). The force was rendered at 1000 Hz and sent to a W5D haptic device (Entact Robotics) with 6-DOF sensing and 5-DOF actuation (no actuation on the roll joint). The streaming point cloud data was captured in a water tank at the University of Washington Applied Physics Lab. Depth

and RGB data was captured using a Xtion Pro depth + RGB sensor (Asus) and recorded using the Robot Operating System (ROS). The GPU processed the recorded point cloud in real-time at $30Hz$ using the OpenCL API (Khronos Group); this includes transformation and filtering of depth and RGB data, color segmentation, as well as generation of sphere trees corresponding to the virtual fixtures. The collision detection was performed in parallel on four cores on the CPU.

We evaluate the proposed method by evaluating two different scenarios. First, two forbidden-region virtual fixtures and secondly a guidance virtual fixture in combination with a forbidden-region virtual fixture.

### *7.5.1 Evaluation of Forbidden-Region Virtual Fixtures*

The implementation of forbidden-region virtual fixtures is evaluated by considering an application of ROV operations that involves the opening and closing of valves. In this example task, we have a series of distinct valves in a water tank that are close to one another.

The depth and RGB data was then segmented such that only points corresponding to the two valves were selected. Two $4cm$ virtual fixtures were then defined around the two valves. As a result, the operator feels a repelling force when pushing the haptic device into the forbidden regions. Every depth sensor frame, the virtual fixtures are updated to match the most recent data, thus allowing for adaptive virtual fixtures to be generated. A sequence when the operator, using a virtual representation of a tool, first tries to push the haptic device towards the left valve and then the right valve is shown in Fig. 7.4.

### *7.5.2 Evaluation of Guidance Virtual Fixtures*

A forbidden-region virtual fixture in combination with a guidance virtual fixture is evaluated in a similar fashion such that the left valve is protected by a forbidden-region virtual fixture. A guidance virtual fixture is placed automatically, using color segmentation, in front of the right valve. It is activated by a key press from the operator. Fig. 7.5 illustrates a sequence where the operator first pushes the haptic device towards the forbidden-region and then, around $3000ms$, activates the guidance virtual fixture. As can be seen, the torque never

reaches zero even though the user moves the haptic device towards the configuration of the guidance virtual fixture. This occurs partly because the operator somewhat resisted the guidance force and mainly because the W5D haptic device can not exert force on the roll joint, resulting in a residual error that only can be corrected visually.

### *7.6 Conclusion*

We have presented a method for rendering of six-degree of freedom forbidden-region and guidance virtual fixtures defined by streaming point cloud data from an RGB-D sensor in an underwater environment. The method is applicable to many underwater robotic applications such as opening valves, positioning tools or mating cables. Forces and torques are successfully rendered at $1000Hz$ with sensor data captured at $30Hz$. We demonstrate that an operator can be guided, using haptic feedback, towards a desired tool configuration while avoiding unintended collision with the environment. Future work includes integration with subsea manipulator, sonar integration as well as improved camera design and customization.

### *Acknowledgments*

The authors would like to thank Entact Robotics for providing us with the W5D haptic device as well as Advanced Micro Devices for providing the GPU hardware. We would also like to thank Professor Weichih Wang, Professor Payman Arabshahi, Chi Leung Tsui, Fong-Ru Hsieh and Ben Estroff for their assistance with the water tank tests.

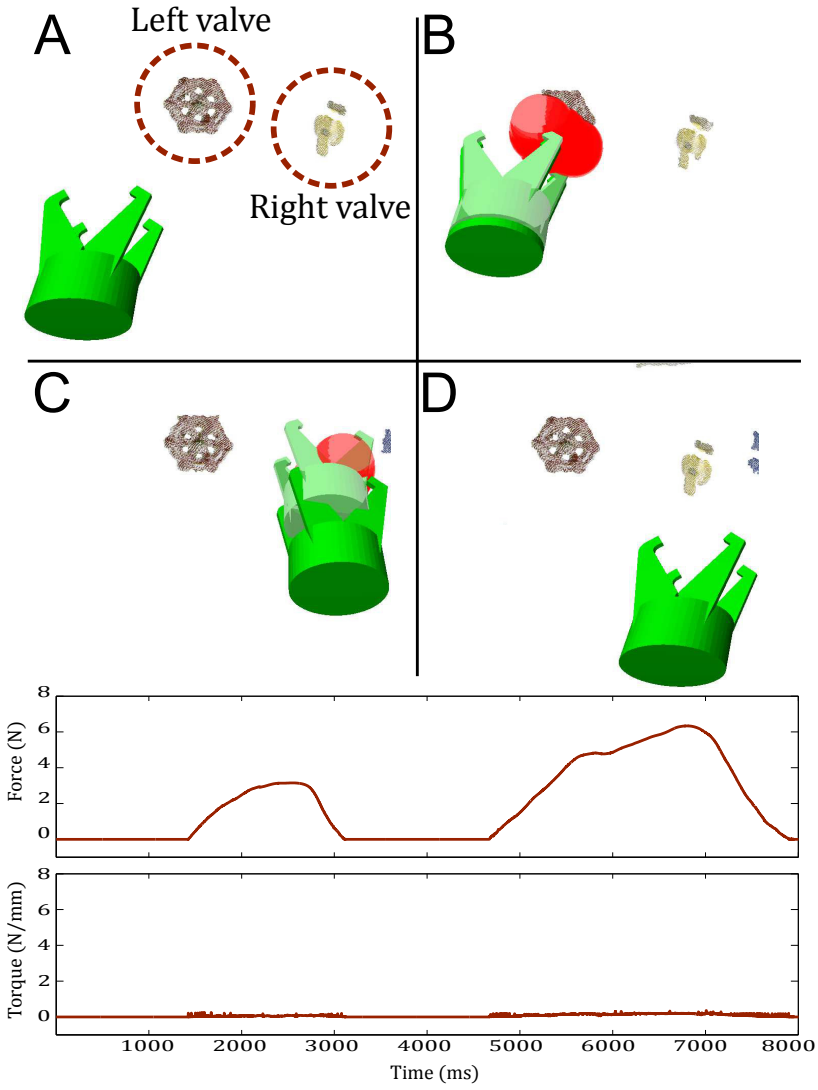


Figure 7.4: The top and bottom traces show the norm of the force and torque respectively while the operator performs the first task (A-D). A) The operator starts at the initial position. B) The operator pushes the haptic device towards the left forbidden-region virtual fixture and experiences a repelling force. C) The operator pushes the haptic device towards the right virtual fixture and experiences a similar force. D) The operator moves away from the virtual fixture.

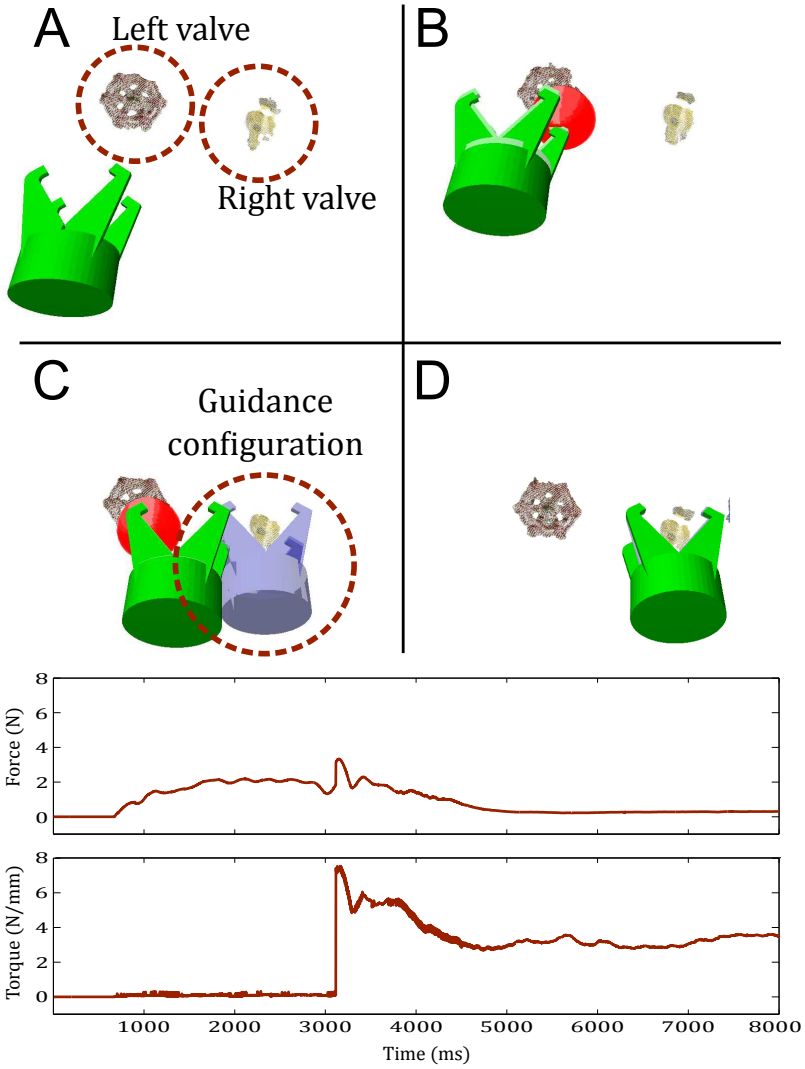


Figure 7.5: The top and bottom traces illustrate the force and torque during the second task (A-D). A) The operator is at the initial position. B) The operator moves towards the left valve which is protected by a forbidden-region virtual fixture, a repelling force is therefore sent to the haptic device. C) The guidance virtual fixture is activated with a key press which and the operator experiences forces and torques pushing towards the right valve. When activated, the guidance virtual fixture is visualized in blue. D) The haptic device at the configuration of the guidance virtual fixture.

# Chapter 8

# FUTURE WORK AND CONCLUSION

## *8.1 Problems Solved*

We have addressed and solved the problem of haptic rendering with remote physical environments using a remote non-contact sensor, so called remote touch. The proposed solution addresses to computational, accuracy and robustness challenged involved with haptic rendering. As a result, the method is perceived as transparent and has the ability to render a variety of geometry accurately, in particular sharp features. This novel approach to haptic rendering is important as it allows for haptic interaction with a remote environment without the need for a remote manipulator.

We also proposed a solution to the problem of virtual fixture rendering in telemanipulation of robots. In this approach, the virtual fixtures are defined implicitly by sensor data captured from an RGB-D sensor located near the remote manipulator. The operator then receives anticipatory force feedback, as the manipulator control loop is bypassed, when approaching for instance a forbidden-region virtual fixture. This particularly useful for manipulators with slow dynamics where the controller delay might become significant.

## *8.2 Open Questions*

An evident problem when implementing the methods presented in this work is the incomplete data in the virtual environment as most realistic scenes cannot be fully captured by a single sensor. A solution could be to fuse sensor data from two sensors to create a more complete virtual environment [119]. This would certainly improve the interaction but unfortunately not necessarily solve the problem as the number of sensors needed increases with the complexity of the scene. A solution could be to not directly haptically interact with the most recent point cloud data but rather use some method for fusing the sensor data similar to [120] and [121]. A problem with these methods is that it is hard to track

dynamic motions even though some approaches in fact detect movements as outliers in the data, they are not tracked. For instance, it is a very difficult task to track an object that for a long time appears to be static and then suddenly moves as this contradicts the best guess. A slightly different solution would be to let the sensor data essentially drive a real-time state estimation of the full environment dynamics. Recently, promising results have been presented in this direction [122]. However, tracking arbitrary objects with unknown dynamics is still an open research problem.

In our method we model kinesthetic feedback between a rigid-body and a streaming point cloud. Even though the streaming point cloud data is dynamic and updated at a fixed frame rate, it is not affected by the users input. That is, the user can interact with but never affect the remote environment, we refer to this as *one-way remote touch*. This is expected as the environment is captured using a non-contact range sensor rather than a remote manipulator. In order to address this problem and create a more realistic haptic interaction with the streaming point cloud data, the user could be allowed to modify the virtual environment. For instance with appropriate computer vision algorithms, point cloud representations of physical objects could be distinguished from each other and further be allowed to transform with respect to each other upon user interaction. For instance, a very realistic haptic interaction could be modeled using a real-time FEM. Note that this would create yet another virtual environment that would deviate from the one constructed from the raw sensor data and the task of switching between these would be non-trivial. For instance, consider when an object is moved out of the field of view of the sensor and it to the user seems like to object vanished for no apparent reason.

In interaction with rapidly moving remote physical objects as captured by a RGB-D sensor, the frame rate of the sensor is of great importance, as described in Section 4.4. A sensor with a higher frame rate would for instance allow for less interpolation in between frames. It would also make the system more robust to remote physical objects with rapid accelerations. Another sensor parameter of great importance is the internal processing time of each frame as this introduces a delay between the time the event of a physical object is captured and the time when the user haptically can experience the same event. In pure haptic interaction, this delay is negligible but in rendering of virtual fixtures this delay

becomes crucial and the accuracy of the virtual fixture rendering will decrease as this delay increases.

It is likely that future haptic devices will be designed in a very different way compared to the current state of the art stylus-like haptic devices. For instance, several researchers have investigated the use of complete exoskeletons as haptic devices, e.g., [123]. One can also imagine a haptic device built with non-rigid links using so called *soft robotics* techniques [124]. The methods presented in this paper would be highly relevant also for these mechanism, although they most likely would have to be modified. The main difference is that we would have to model the virtual tool as the full exoskeleton or the soft mechanism, which potentially could be challenging.

In teleoperation, the method for virtual fixture rendering method presented in this dissertation fails when the remote manipulator is blocking the view of the RGB-D sensor. For instance, if a forbidden-region virtual fixture is defined around a object of interest, the rendering tends to fail when the remote manipulator comes too close to this object. The methods for generating a more complete virtual environment discussed above could potentially be used in this case. The rendering method could probably also recover if the remote manipulator is equipped with a force sensor. Another solution could be to use accelerometers mounted on the manipulator [125].

In a commercial teleoperated system, for instance in oil and gas work, the system would also have to robustly handle sequential tasks. This requires a way of according to some specification sequentially lock/unlock forbidden-region virtual fixtures and activate/deactivate virtual fixtures. At an early stage, this is most likely done using manual operator inputs. In future work, we will develop an automatic system for verifying that necessary operations has been executed and use this to progress the sequential operation. This system can also be used to notify the operator of missed steps or procedural violations.

### *8.3 Conclusion*

The methods for haptic rendering presented in this dissertation provides an important stepping stone for future research on remote touch by showing that some aspects of the sense of touch in fact can be transmitted over long distances, without the need for expensive

remote contact sensors. Even though our methods might not be able to in a convincing way produce remote touch equivalent to real physical haptic interaction between a grand parent and a grand child, we believe that our methods can be extended, mainly using computer vision methods, to in the near future reach that goal.

Teleoperated robots and manipulators will provide us with the means to access previously inaccessible locations. In addition to allowing access to dangerous or harsh locations where humans go, teleoperation also allows scaling as a way to manipulate very small or very big objects. Methods for assisted teleoperation such as the ones presented in this dissertation will provide means for operating more efficiently and safer for tasks that cannot be purely automated.

# BIBLIOGRAPHY

[1] F. Rydén, S. Nia Kosari, and H. J. Chizeck, "Proxy method for fast haptic rendering from time varying point clouds," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 2614–2619.

[2] F. Rydén and H. J. Chizeck, "Forbidden-region virtual fixtures from streaming point clouds: Remotely touching and protecting a beating heart," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 3308–3313.

[3] ——, "A Proxy Method for Real-Time 3-DOF Haptic Rendering of Streaming Point Cloud Data," *IEEE Transactions on Haptics*, p. 1, 2013.

[4] ——, "A Method for Constraint-Based Six Degree-of-Freedom Haptic Interaction with Streaming Point Clouds," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2013.

[5] M. Minsky, O.-y. Ming, O. Steele, F. P. Brooks Jr, and M. Behensky, "Feeling and seeing: issues in force display," *ACM SIGGRAPH Computer Graphics*, vol. 24, no. 2, pp. 235–241, 1990.

[6] T. H. Massie and J. K. Salisbury, "The phantom haptic interface: A device for probing virtual objects," in *Proceedings of the ASME winter annual meeting, symposium on haptic interfaces for virtual environment and teleoperator systems*, vol. 55, no. 1. Kluwer, 1994, pp. 295–300.

[7] K. Salisbury, D. Brock, T. Massie, N. Swarup, and C. Zilles, "Haptic rendering: Programming touch interaction with virtual objects," in *Proceedings of the 1995 symposium on Interactive 3D graphics*. ACM, 1995, pp. 123–130.

[8] W. R. Mark, S. C. Randolph, M. Finch, J. M. Van Verth, and R. M. Taylor II, "Adding force feedback to graphics systems: Issues and solutions," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 447–452.

[9] I. E. Sutherland, "The ultimate display," *Multimedia: From Wagner to virtual reality*, 1965.

[10] B. E. Insko, "Passive haptics significantly enhances virtual environments," Ph.D. dissertation, University of North Carolina, 2001.

[11] C. Ho, C. Basdogan, M. Slater, N. Durlach, and M. A. Srinivasan, "An experiment on the influence of haptic communication on the sense of being together," in *BT Presence Workshop*. Citeseer, 1998.

[12] J. Kim, H. Kim, B. K. Tay, M. Muniyandi, M. A. Srinivasan, J. Jordan, J. Mortensen, M. Oliveira, and M. Slater, "Transatlantic touch: a study of haptic collaboration over long distance," *Presence: Teleoperators & Virtual Environments*, vol. 13, no. 3, pp. 328–337, 2004.

[13] C. V. Edmond Jr, D. Heskamp, D. Sluis, D. Stredney, D. Sessanna, G. Wiet, R. Yagel, S. Weghorst, P. Oppenheimer, J. Miller, and Others, "ENT endoscopic surgical training simulator." *Studies in health technology and informatics*, vol. 39, p. 518, 1997.

[14] S. Gibson, J. Samosky, A. Mor, C. Fyock, E. Grimson, T. Kanade, R. Kikinis, H. Lauer, N. McKenzie, S. Nakajima, and Others, "Simulating arthroscopic knee surgery using volumetric object representations, real-time volume rendering and haptic feedback," in *CVRMed-MRCAS'97*. Springer, 1997, pp. 367–378.

[15] H. M. Hoffman, D. Stredney, and S. J. Weghorst, "Force interactions in laparoscopic simulations: Haptic rendering of soft tissues," *Medicine Meets Virtual Reality: Art, Science, Technology: Healthcare (R) Evolution*, vol. 50, p. 385, 1998.

[16] D. Morris, C. Sewell, F. Barbagli, K. Salisbury, N. H. Blevins, and S. Girod, "Visuohaptic simulation of bone surgery for training and evaluation," *Computer Graphics and Applications, IEEE*, vol. 26, no. 6, pp. 48–57, 2006.

[17] C. Basdogan, M. Sedef, M. Harders, and S. Wesarg, "VR-based simulators for training in minimally invasive surgery," *Computer Graphics and Applications, IEEE*, vol. 27, no. 2, pp. 54–66, 2007.

[18] M. Ouh-Young, M. Pique, J. Hughes, N. Srinivasan, and F. P. Brooks Jr, "Using a manipulator for force display in molecular docking," in *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*. IEEE, 1988, pp. 1824–1829.

[19] F. P. Brooks Jr, M. Ouh-Young, J. J. Batter, and P. Jerome Kilpatrick, "Project GROPEHaptic displays for scientific visualization," in *ACM SIGGraph Computer Graphics*, vol. 24, no. 4. ACM, 1990, pp. 177–185.

[20] B. Hannaford, "A design framework for teleoperators with kinesthetic feedback," *Robotics and Automation, IEEE Transactions on*, vol. 5, no. 4, pp. 426–434, 1989.

[21] W. McMahan, J. Gewirtz, D. Standish, P. Martin, J. A. Kunkel, M. Lilavois, A. Wedmid, D. I. Lee, and K. J. Kuchenbecker, "Tool contact acceleration feedback for telerobotic surgery," *Haptics, IEEE Transactions on*, vol. 4, no. 3, pp. 210–220, 2011.

[22] J. E. Colgate and J. M. Brown, "Factors affecting the z-width of a haptic display," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*. IEEE, 1994, pp. 3205–3210.

[23] J. E. Colgate and G. Schenkel, "Passivity of a class of sampled-data systems: Application to haptic interfaces," in *American Control Conference, 1994*, vol. 3. IEEE, 1994, pp. 3236–3240.

[24] R. J. Adams and B. Hannaford, "A two-port framework for the design of unconditionally stable haptic interfaces," in *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, vol. 2. IEEE, 1998, pp. 1254–1259.

[25] ——, "Stable haptic interaction with virtual environments," *Robotics and Automation, IEEE Transactions on*, vol. 15, no. 3, pp. 465–474, 1999.

[26] B. E. Miller, J. E. Colgate, and R. A. Freeman, "Guaranteed stability of haptic systems with nonlinear virtual environments," *Robotics and Automation, IEEE Transactions on*, vol. 16, no. 6, pp. 712–719, 2000.

[27] B. Hannaford and J.-H. Ryu, "Time domain passivity control of haptic interfaces," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 2. IEEE, 2001, pp. 1863–1869.

[28] J. Abbott and A. M. Okamura, "Stable forbidden-region virtual fixtures for bilateral telemanipulation," *Journal of dynamic systems, measurement, and control*, vol. 128, p. 53, 2006.

[29] N. Diolaiti, G. Niemeyer, F. Barbagli, J. K. Salisbury, and C. Melchiorri, "The effect of quantization and coulomb friction on the stability of haptic rendering," in *Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005. World Haptics 2005. First Joint*. IEEE, 2005, pp. 237–246.

[30] N. Diolaiti, G. Niemeyer, F. Barbagli, and J. K. Salisbury, "Stability of haptic rendering: Discretization, quantization, time delay, and coulomb effects," *Robotics, IEEE Transactions on*, vol. 22, no. 2, pp. 256–268, 2006.

[31] M. A. Otaduy and M. C. Lin, "Introduction to haptic rendering," in *ACM SIGGRAPH 2005 Courses*. ACM, 2005, p. 3.

[32] K. Salisbury, F. Conti, and F. Barbagli, "Haptic rendering: introductory concepts," *Computer Graphics and Applications, IEEE*, vol. 24, no. 2, pp. 24–32, 2004.

[33] C. Basdogan and M. A. Srinivasan, "Haptic rendering in virtual environments," *Handbook of virtual environments*, pp. 117–134, 2002.

[34] J. E. Colgate, M. C. Stanley, and J. M. Brown, "Issues in the haptic display of tool use," in *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, vol. 3. IEEE, 1995, pp. 140–145.

[35] C. B. Zilles and J. K. Salisbury, "A constraint-based god-object method for haptic display," in *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, vol. 3. IEEE, 1995, pp. 146–151.

[36] D. C. Ruspini, K. Kolarov, and O. Khatib, "The haptic display of complex graphical environments," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1997, pp. 345–352.

[37] D. Ruspini and O. Khatib, "Collision/contact models for the dynamic simulation of complex environments," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 82. Citeseer, 1997.

[38] K. Salisbury and C. Tarr, "Haptic rendering of surfaces defined by implicit functions," in *ASME Dynamic Systems and Control Division*, vol. 61, 1997, pp. 61–67.

[39] L. Kim, A. Kyrikou, G. S. Sukhatme, and M. Desbrun, "An implicit-based haptic rendering technique," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 3. IEEE, 2002, pp. 2943–2948.

[40] Y. Adachi, T. Kumano, and K. Ogino, "Intermediate representation for stiff virtual objects," in *Virtual Reality Annual International Symposium, 1995. Proceedings.* IEEE, 1995, pp. 203–210.

[41] C.-H. Ho, C. Basdogan, and M. A. Srinivasan, "Ray-based haptic rendering: force and torque interactions between a line probe and 3D objects in virtual environments," *The International Journal of Robotics Research*, vol. 19, no. 7, pp. 668–683, 2000.

[42] D. E. Johnson and P. Willemsen, "Six degree-of-freedom haptic rendering of complex polygonal models," in *Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2003. HAPTICS 2003. Proceedings. 11th Symposium on.* IEEE, 2003, pp. 229–235.

[43] D. E. Johnson, P. Willemsen, and E. Cohen, "Six degree-of-freedom haptic rendering using spatialized normal cone search," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 11, no. 6, pp. 661–670, 2005.

[44] A. Gregory, A. Mascarenhas, S. Ehmann, M. Lin, and D. Manocha, "Six degree-of-freedom haptic display of polygonal models," in *Proceedings of the conference on Visualization'00*. IEEE Computer Society Press, 2000, pp. 139–146.

[45] D. D. Nelson, D. E. Johnson, and E. Cohen, "Haptic rendering of surface-to-surface sculpted model interaction," in *ACM SIGGRAPH 2005 Courses*. ACM, 2005, p. 97.

[46] D. Baraff, "Analytical methods for dynamic simulation of non-penetrating rigid bodies," in *ACM SIGGRAPH Computer Graphics*, vol. 23, no. 3. ACM, 1989, pp. 223–232.

[47] ——, "Fast contact force computation for nonpenetrating rigid bodies," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM, 1994, pp. 23–34.

[48] S. Redon, A. Kheddar, and S. Coquillart, "Gauss' least constraints principle and rigid body simulations," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 1. IEEE, 2002, pp. 517–522.

[49] ——, "Fast continuous collision detection between rigid bodies," in *Computer graphics forum*, vol. 21, no. 3. Wiley Online Library, 2002, pp. 279–287.

[50] W. A. McNeely, K. D. Puterbaugh, and J. J. Troy, "Six degree-of-freedom haptic rendering using voxel sampling," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1999, pp. 401–408.

[51] ——, "Voxel-based 6-dof haptic rendering improvements," *Haptics-e*, vol. 3, no. 7, 2006.

[52] M. Wan and W. A. McNeely, "Quasi-static approximation for 6 degrees-of-freedom haptic rendering," in *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*. IEEE Computer Society, 2003, p. 34.

[53] S. Hasegawa and M. Sato, "Real-time Rigid Body Simulation for Haptic Interactions Based on Contact Volume of Polygonal Objects," in *Computer Graphics Forum*, vol. 23, no. 3. Wiley Online Library, 2004, pp. 529–538.

[54] M. A. Otaduy and M. C. Lin, "Sensation preserving simplification for haptic rendering," in *ACM SIGGRAPH 2005 Courses*. ACM, 2005, p. 72.

[55] ——, "Stable and responsive six-degree-of-freedom haptic manipulation using implicit integration," in *Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005. World Haptics 2005. First Joint*. IEEE, 2005, pp. 247–256.

[56] ——, "A modular haptic rendering algorithm for stable and transparent 6-DOF manipulation," *Robotics, IEEE Transactions on*, vol. 22, no. 4, pp. 751–762, 2006.

[57] J. Barbič and D. James, "Time-critical distributed contact for 6-dof haptic rendering of adaptively sampled reduced deformable models," 2007.

[58] J. Barbič and D. L. James, "Six-dof haptic rendering of contact between geometrically complex reduced deformable models," *Haptics, IEEE Transactions on*, vol. 1, no. 1, pp. 39–52, 2008.

[59] M. Kolesnikov and M. Žefran, "Energy-based 6-DOF penetration depth computation for penalty-based haptic rendering algorithms," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on.* IEEE, 2007, pp. 2120–2125.

[60] ——, "Generalized penetration depth for penalty-based six-degree-of-freedom haptic rendering," *Robotica*, vol. 26, no. 04, pp. 513–524, 2008.

[61] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha, "Six-degree-of-freedom haptic display using localized contact computations," in *Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2002. HAPTICS 2002. Proceedings. 10th Symposium on.* IEEE, 2002, pp. 209–216.

[62] X. He and Y. Chen, "Six-degree-of-freedom haptic rendering in virtual teleoperation," *Instrumentation and Measurement, IEEE Transactions on*, vol. 57, no. 9, pp. 1866–1875, 2008.

[63] B. Mirtich and J. Canny, "Impulse-based simulation of rigid bodies," in *Proceedings of the 1995 symposium on Interactive 3D graphics.* ACM, 1995, pp. 181—-ff.

[64] B. Chang and J. E. Colgate, "Real-time impulse-based simulation of rigid body systems for haptic display," in *Proc. Symp. on Interactive 3D Graphics*, 1997, pp. 200–209.

[65] D. Constantinescu, S. E. Salcudean, and E. A. Croft, "Haptic rendering of rigid body collisions," in *Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2004. HAPTICS'04. Proceedings. 12th International Symposium on.* IEEE, 2004, pp. 2–8.

[66] ——, "Haptic rendering of rigid contacts using impulsive and penalty forces," *Robotics, IEEE Transactions on*, vol. 21, no. 3, pp. 309–323, 2005.

[67] E. Ruffaldi, D. Morris, F. Barbagli, K. Salisbury, and M. Bergamasco, "Voxel-based haptic rendering using implicit sphere trees," in *Haptic interfaces for virtual environment and teleoperator systems, 2008. haptics 2008. symposium on.* IEEE, 2008, pp. 319–325.

[68] P. J. Berkelman, R. L. Hollis, and D. Baraff, "Interaction with a real time dynamic environment simulation using a magnetic levitation haptic interface device," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 4. IEEE, 1999, pp. 3261–3266.

[69] M. Ortega, S. Redon, and S. Coquillart, "A six degree-of-freedom god-object method for haptic display of rigid bodies," in *Virtual Reality Conference, 2006.* IEEE, 2006, pp. 191–198.

[70] ——, "A six degree-of-freedom god-object method for haptic display of rigid bodies with surface properties," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 13, no. 3, pp. 458–469, 2007.

[71] S. Chan, F. Conti, N. H. Blevins, and K. Salisbury, "Constraint-Based Six Degree-Of-Freedom Haptic Rendering of Volume-Embedded Isosurfaces," in *Proc. Of the 2011 IEEE International World Haptics Conference*, 2011.

[72] X. Zhang, D. Wang, Y. Zhang, and J. Xiao, "Configuration-based optimization for six degree-of-freedom haptic rendering using sphere-trees," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on.* IEEE, 2011, pp. 2602–2607.

[73] D. Wang, X. Zhang, Y. Zhang, and J. Xiao, "Configuration-based optimization for six degree-of-freedom haptic rendering for fine manipulation," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on.* IEEE, 2011, pp. 906–912.

[74] D. Wang, S. Liu, X. Zhang, Y. Zhang, and J. Xiao, "Six-degree-of-freedom haptic simulation of organ deformation in dental operations," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on.* IEEE, 2012, pp. 1050–1056.

[75] L. Linsen, "Point cloud representation," *University of Karlsruhe, Germany Technical Report, Faculty of Informatics*, 2001.

[76] J. K. Lee and Y. J. Kim, "Haptic rendering of point set surfaces," in *EuroHaptics Conference, 2007 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics 2007. Second Joint.* IEEE, 2007, pp. 513–518.

[77] D. Levin, "The approximation power of moving least-squares," *Mathematics of computation*, vol. 67, no. 224, pp. 1517–1532, 1998.

[78] N. R. El-Far, N. D. Georganas, and A. El Saddik, "An algorithm for haptically rendering objects described by point clouds," in *Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on*, May 2008, pp. 1443–1448.

[79] K. G. Sreeni and S. Chaudhuri, "Haptic rendering of dense 3D point cloud data," in *Haptics Symposium (HAPTICS), 2012 IEEE*. IEEE, 2012, pp. 333–339.

[80] K. G. Sreeni, K. Priyadarshini, A. K. Praseedha, and S. Chaudhuri, "Haptic rendering of cultural heritage objects at different scales," in *Haptics: Perception, Devices, Mobility, and Communication*. Springer, 2012, pp. 505–516.

[81] K. G. Sreeni and S. Chaudhuri, "Haptic rendering of variable density point cloud through local kernel bandwidth estimation," in *Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing*. ACM, 2012, p. 26.

[82] J. Cha, S. Kim, I. Oakley, J. Ryu, and K. Lee, "Haptic interaction with depth video media," *Advances in Multimedia Information Processing-PCM 2005*, pp. 420–430, 2005.

[83] J. Cha, M. Eid, and A. El Saddik, "DIBHR: Depth Image-Based Haptic Rendering," *Haptics: Perception, Devices and Scenarios*, pp. 640–650, 2008.

[84] A. Leeper, S. Chan, and K. Salisbury, "Point clouds can be represented as implicit surfaces for constraint-based haptic rendering," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 5000–5005.

[85] S. Rasool and A. Sourin, "Haptic interaction with 2D images," in *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*. ACM, 2011, pp. 13–22.

[86] A. Leeper, S. Chan, K. Hsiao, M. Ciocarlie, and K. Salisbury, "Constraint-based haptic rendering of point data for teleoperated robot grasping," in *Haptics Symposium (HAPTICS), 2012 IEEE*, Mar. 2012, pp. 377–383.

[87] L. B. Rosenberg, "Virtual fixtures: Perceptual tools for telerobotic manipulation," in *Virtual Reality Annual International Symposium, 1993., 1993 IEEE*, Sep. 1993, pp. 76–82.

[88] S. Payandeh and Z. Stanisic, "On application of virtual fixtures as an aid for telemanipulation and training," in *Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2002. HAPTICS 2002. Proceedings. 10th Symposium on*. IEEE, 2002, pp. 18–23.

[89] K.-W. Kwok, G. P. Mylonas, L. W. Sun, M. Lerotic, J. Clark, T. Athanasiou, A. Darzi, and G.-Z. Yang, "Dynamic active constraints for hyper-redundant flexible robots," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2009*. Springer, 2009, pp. 410–417.

[90] J. Abbott, P. Marayong, and A. Okamura, "Haptic virtual fixtures for robot-assisted manipulation," *Robotics Research*, pp. 49–64, 2007.

[91] A. Bettini, P. Marayong, S. Lang, A. M. Okamura, and G. D. Hager, "Vision-assisted control for manipulation using virtual fixtures," *Robotics, IEEE Transactions on*, vol. 20, no. 6, pp. 953–966, 2004.

[92] M. Li and R. H. Taylor, "Spatial motion constraints in medical robot using virtual fixtures generated by anatomy," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 2. IEEE, 2004, pp. 1270–1275.

[93] J. Ren, R. V. Patel, K. A. McIsaac, G. Guiraudon, and T. M. Peters, "Dynamic 3-D virtual fixtures for minimally invasive beating heart procedures," *Medical Imaging, IEEE Transactions on*, vol. 27, no. 8, pp. 1061–1070, 2008.

[94] B. Davies, M. Jakopec, S. J. Harris, F. y Baena, A. Barrett, A. Evangelidis, P. Gomes, J. Henckel, and J. Cobb, "Active-constraint robotics for surgery," *Proceedings of the IEEE*, vol. 94, no. 9, pp. 1696–1704, 2006.

[95] N. V. Navkar, Z. Deng, D. J. Shah, K. E. Bekris, and N. V. Tsekos, "Visual and force-feedback guidance for robot-assisted interventions in the beating heart with real-time MRI," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 689–694.

[96] J. W. Park, J. Choi, Y. Park, and K. Sun, "Haptic virtual fixture for robotic cardiac catheter navigation," *Artificial Organs*, vol. 35, no. 11, pp. 1127–1131, 2011.

[97] T. L. Gibo, L. N. Verner, D. D. Yuh, and A. M. Okamura, "Design considerations and human-machine performance of moving virtual fixtures," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 671–676.

[98] T. Yamamoto, N. Abolhassani, S. Jung, A. M. Okamura, and T. N. Judkins, "Augmented reality and haptic interfaces for robot-assisted surgery," *The International Journal of Medical Robotics and Computer Assisted Surgery*, 2011.

[99] D. Ribas, P. Ridao, J. D. Tardos, and J. Neira, "Underwater SLAM in a marina environment," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 1455–1460.

[100] S. B. Williams, P. Newman, G. Dissanayake, and H. Durrant-Whyte, "Autonomous underwater simultaneous localisation and map building," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 2, pp. 1793–1798 vol.2.

[101] A. Kim and R. M. Eustice, "Real-time visual {SLAM} for autonomous underwater hull inspection using visual saliency," *IEEE Transactions on Robotics*, 2013.

[102] F. S. Hover, R. M. Eustice, A. Kim, B. Englot, H. Johannsson, M. Kaess, and J. J. Leonard, "Advanced perception, navigation and planning for autonomous in-water ship hull inspection," *International Journal of Robotics Research, Special Issue on 3D Exploration, Mapping, and Surveillance*, vol. 31, no. 12, pp. 1445–1464, Oct. 2012.

[103] N. Carlevaris-Bianco and R. M. Eustice, "Multi-view registration for feature-poor underwater imagery," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Shanghai, China, May 2011, pp. 423–430.

[104] C. White, D. Hiranandani, C. S. Olstad, K. Buhagiar, T. Gambin, and C. M. Clark, "The Malta cistern mapping project: Underwater robot mapping and localization within ancient tunnel systems," *Journal of Field Robotics*, vol. 27, no. 4, pp. 399–411, 2010. [Online]. Available: http://dx.doi.org/10.1002/rob.20339

[105] K. Teo, K. W. Ong, and H. C. Lai, "Obstacle detection, avoidance and anti collision for MEREDITH AUV," in *OCEANS 2009, MTS/IEEE Biloxi-Marine Technology for Our Future: Global and Local Challenges*. IEEE, 2009, pp. 1–10.

[106] S.-C. Yu, T.-W. Kim, G. Marani, and S. K. Choi, "Real-time 3D sonar image recognition for underwater vehicles," in *Underwater technology and workshop on scientific use of submarine cables and related technologies, 2007. Symposium on*. IEEE, 2007, pp. 142–146.

[107] M. Prats, J. J. Fernández, and P. J. Sanz, "Combining template tracking and laser peak detection for 3D reconstruction and grasping in underwater environments," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 106–112.

[108] M. Prats, J. Pérez, J. J. Fernández, and P. J. Sanz, "An open source tool for simulation and supervision of underwater intervention missions," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 2577–2582.

[109] V. Rigaud, E. Coste-Maniere, M. J. Aldon, P. Probert, M. Perrier, P. Rives, D. Simon, D. Lang, J. Kiener, A. Casal, J. Amar, P. Dauchez, and M. Chantler, "UNION: underwater intelligent operation and navigation," *Robotics Automation Magazine, IEEE*, vol. 5, no. 1, pp. 25–35, Mar. 1998.

[110] G. Marani, S. K. Choi, and J. Yuh, "Underwater autonomous manipulation for intervention missions AUVs," *Ocean Engineering*, vol. 36, no. 1, pp. 15–23, 2009.

[111] G. Marani and S. Choi, "Underwater target localization," *Robotics & Automation Magazine, IEEE*, vol. 17, no. 1, pp. 64–70, 2010.

[112] L. M. Batteau, A. Liu, J. B. A. Maintz, Y. Bhasin, and M. W. Bowyer, "A study on the perception of haptics in surgical simulation," *Medical Simulation*, pp. 185–192, 2004.

[113] N. J. Mitra and A. Nguyen, "Estimating surface normals in noisy point cloud data," in *Proceedings of the nineteenth annual symposium on Computational geometry*. ACM, 2003, pp. 322–328.

[114] H. Wendland, "Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree," *Advances in computational Mathematics*, vol. 4, no. 1, pp. 389–396, 1995.

[115] R. Kikuuwe, N. Takesue, and H. Fujimoto, "A control framework to generate nonenergy-storing virtual fixtures: Use of simulated plasticity," *Robotics, IEEE Transactions on*, vol. 24, no. 4, pp. 781–793, 2008.

[116] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Computer Vision, 1998. Sixth International Conference on*. IEEE, 1998, pp. 839–846.

[117] D. R. Wilhelmsen, "A Nearest Point Algorithm," *Mathematics of computation*, vol. 30, no. 133, pp. 48–57, 1976.

[118] G. Bradshaw and C. O'Sullivan, "Adaptive medial-axis approximation for sphere-tree construction," *ACM Transactions on Graphics (TOG)*, vol. 23, no. 1, pp. 1–26, 2004.

[119] D. A. Butler, S. Izadi, O. Hilliges, D. Molyneaux, S. Hodges, and D. Kim, "Shake'n'sense: reducing interference for overlapping structured light depth cameras," in *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*. ACM, 2012, pp. 1933–1936.

[120] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments," in *the 12th International Symposium on Experimental Robotics (ISER)*, vol. 20, 2010, pp. 22–25.

[121] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*. IEEE, 2011, pp. 127–136.

[122] J. Schulman, A. Lee, J. Ho, and P. Abbeel, "Tracking Deformable Objects with Point Clouds," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013.

[123] J. Rosen and J. C. Perry, "Upper limb powered exoskeleton," *International Journal of Humanoid Robotics*, vol. 4, no. 03, pp. 529–548, 2007.

[124] R. Pfeifer, M. Lungarella, and F. Iida, "The challenges ahead for bio-inspired'soft'robotics," *Communications of the ACM*, vol. 55, no. 11, pp. 76–87, 2012.

[125] K. J. Kuchenbecker, J. Gewirtz, W. McMahan, D. Standish, P. Martin, J. Bohren, P. J. Mendoza, and D. I. Lee, "Verrotouch: high-frequency acceleration feedback for telerobotic surgery," in *Haptics: Generating and Perceiving Tangible Sensations*. Springer, 2010, pp. 189–196.

# VITA

Fredrik Rydén was born in 1988 and grew up in Borlänge, Sweden. In 2010, he earned a Bachelor of Science degree in Automation and Mechatronics from Chalmers University of Technology in Göteborg, Sweden. Same year, he started graduate school at the University of Washington and joined the BioRobotics Laboratory. In 2012, he earned a Master of Science degree in Electrical Engineering and was awarded with the departmental Outstanding Research Award. In August 2013, he earned a Doctor of Philosophy degree in Electrical Engineering from the University of Washington.