Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS

Petr Kadleček

Haptic rendering for 6/3-DOF haptic devices

Department of Software and Computer Science Education

Supervisor of the master thesis:	Mgr. Petr Kmoch
Study programme:	Informatics
Specialization:	Software Systems

Prague 2013

I would like to thank my supervisor, Mgr. Petr Kmoch, for his support, availability and constructive suggestions. He has guided me in the correct directions during my master thesis and helped me with his feedbacks. I would also like to express my gratitude to my family and my girlfriend for their continuous support and patience.

Název práce: Haptická vizualizace pro zařízení se 6/3 stupni volnosti

Autor: Petr Kadleček

Katedra: Kabinet software a výuky informatiky

Vedoucí diplomové práce: Mgr. Petr Kmoch, KSVI

Abstrakt: Využití haptických zařízení se rozšířilo do oblastí jako je virtuální simulace výroby, virtuální montáž, či simulace lékařských zákroků. Pokrok ve výrobě umožnil velké rozšíření haptických zařízení s pohybem v šesti stupních volnosti, avšak s omezenou zpětnou vazbou jen v translaci - tzv. nesymetrická zařízení se 6/3 stupni volnosti. Dosavadní algoritmy pro haptickou vizualizaci pracují však správně jen pro symetrická zařízení. Tato práce analyzuje algoritmy haptické vizualizace pro zařízení se třemi a šesti stupni volnosti a na základě analýzy navrhuje algoritmus haptické vizualizace pro zařízení se 6/3 stupni volnosti s podporou pseudo-haptického vnímání. Algoritmus je na základě analýzy implementován a otestován v uživatelské studii.

Klíčová slova: haptická vizualizace, 6/3 stupňů volnosti, pseudo-haptika

Title: Haptic rendering for 6/3-DOF haptic devices

Author: Petr Kadleček

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Petr Kmoch, KSVI

Abstract: Application of haptic devices expanded to fields like virtual manufacturing, virtual assembly or medical simulations. Advances in development of haptic devices have resulted in a wide distribution of assymetric 6/3-DOF haptic devices. However, current haptic rendering algorithms work correctly only for symmetric devices. This thesis analyzes 3-DOF and 6-DOF haptic rendering algorithms and proposes an algorithm for 6/3-DOF haptic rendering involving pseudo-haptics. The 6/3-DOF haptic rendering algorithm is implemented based on the previous analysis and tested in a user study.

Keywords: haptic rendering, 6/3 degrees of freedom, pseudo-haptics

Contents

1. Introduction

1.1 What is haptics?

The word haptic originates from Greek “hapt-esthai” or “haptó” which means “to feel” or “to touch”. Haptic technology makes it possible to use the sense of touch with computers. It is another modality of human-computer interaction. Haptic perception is usually divided into two subsystems: tactile (cutaneous) and kinesthetic.

1.1.1 Tactile sensing

The sense of touch is generally understood as a skin receptor response to stimuli such as pressure, vibration or temperature. These receptors help us with recognizing objects, especially when determining the surface texture of an object, also called haptic texture. Various mechanoreceptors (e.g. Meissner corpuscle, Pacinian corpuscle, Merkel disks, Ruffini endings) are sensitive to different frequencies and detect diverse types of stimuli as shown in Figure 1.1. Devices that make use of cutaneous mechanoreceptors are called tactile or touch devices.

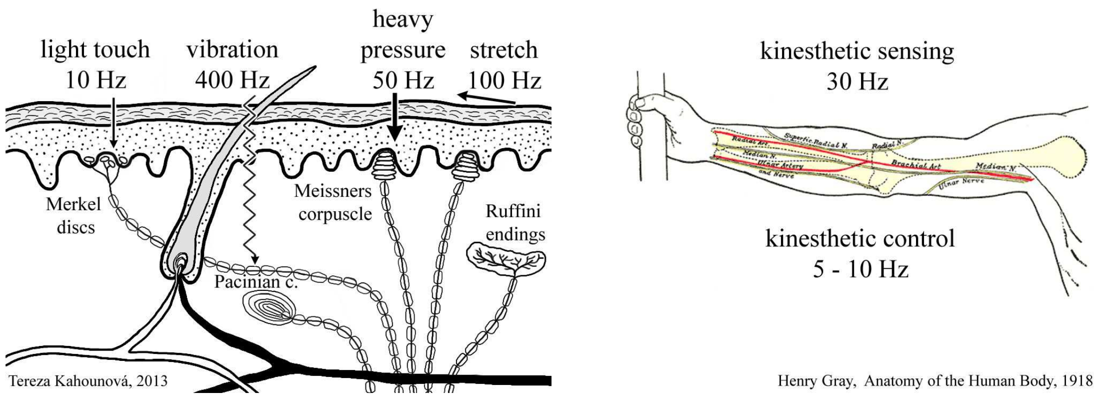


Figure 1.1: Approximate frequencies of cutaneous mechanoreceptors (on the left) and kinesthetic sensors (on the right)

1.1.2 Kinesthetic sensing

To recognize the shape of an object we mainly use a part of the somatosensory system called kinesthetic sense. Kinesthetic sense provides us with information about movement and position of our body parts in the environment. We are able to feel various forces in different directions using our muscles and joints and use this information to determine the size, shape and other characteristics of objects we touch and the forces they exert. Control bandwidth of the kinesthetic system is approximately 5 Hz and the sensing bandwidth is around 20 Hz [1]. *Haptic* modality of human-computer interaction utilizes the sense of touch and kinesthetic sense to perceive virtual objects; it generally incorporates hands, upper torso, head and other parts of the body.

1.2 Applications of computer haptics

There are many situations where we depend on haptic perception in everyday life. The role of computer haptics in virtual reality is to simulate these situations or even let us experience interaction impossible in the real world. Many new areas of haptic applications have emerged and become an intriguing research topic.

Medical simulations

Medical simulations create an opportunity for training surgical procedures. Virtual tissue behavior is haptically rendered through a surgical instrument connected to a haptic device. Combination of stereoscopy and haptics provides realistic sensation and helps students to improve their fine motor skills. Augmented reality helps to enhance sensations of telemanipulation, for example using virtual objects or guiding cues [2].

Virtual prototyping

Virtual prototyping aims to create virtual reality prototypes of complex systems and objects that are costly or require excessive effort to manipulate. Typical applications are new design evaluation, virtual assembly or physical ergonomics tests. Highly realistic immersive applications such as Haptic CAVE (Haptic Computer Automated Virtual Environment [3]) combine human-scale haptics with large projector screens, position tracking and 3D sound system.

Virtual hair modeling project

Virtual hair modeling (project Stubble) was a team software project at Charles University in Prague where I have implemented support of haptic interaction. The application helps the user to model hair with hair guides in Autodesk Maya using a mouse or a haptic device (Figure 1.2).

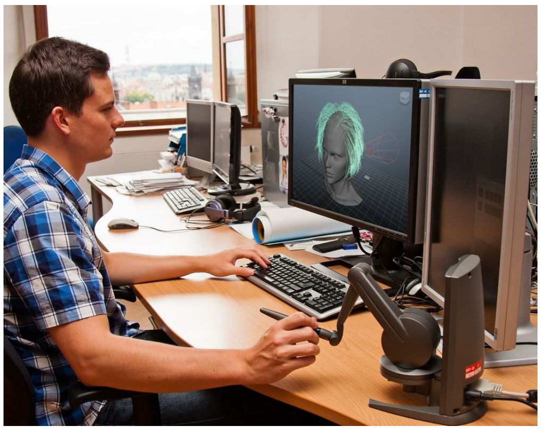


Figure 1.2: Virtual hair modeling in Autodesk Maya

Other applications of computer haptics include scientific visualization, physical rehabilitation, virtual clay modeling and 3D painting, veterinary training, advanced ultrasound training, dental training systems, flight simulators, computer games, user interface enhancement or telerebotics. More information can be found in [2].

1.3 Haptic devices

Haptic devices can be divided into two classes: impedance-type devices and admittance-type devices. The purpose of an *impedance* type haptic device is to generate force feedback of a given direction and magnitude in a specified workspace and send the position information of a control part of the apparatus to the computer. *Admittance* type haptic devices measure applied forces and generate a position change of a control part of the apparatus as haptic feedback. This thesis will focus only on impedance type haptic devices. Various classifications of haptic devices exist. More detailed classification and analysis from the engineering point of view can be found in [1].

1.3.1 Degrees of freedom

Haptic devices consist of several sensors and actuators. The number of sensors is usually the same as the number of actuators. However, for reasons of simplicity lower cost, less actuators than sensors may be present. Dimension of an orientation ability (or configuration space of the mechanical parts of a haptic device) is a property of haptic devices known as the number of degrees of freedom. Typical 3-DOF (three degrees of freedom) devices (e.g. Novint Falcon) have a movable grip connected via three arms allowing translation of the grip within the workspace and providing force feedback. 6-DOF devices provide positional sensing using a pivoting stylus, force feedback and torque feedback in roll, pitch and yaw axes of rotation. In this thesis, the focus will be on asymmetric 6/3-DOF devices (especially on a PHANToM Desktop device) that combine 6-DOF positional sensing and 3-DOF force feedback, i.e. no torque feedback is present.

1.3.2 PHANToM Desktop

PHANToM Desktop (Figure 1.3) is a 6/3-DOF haptic device manufactured by SensAble Technologies, Inc. Workspace of the device is 160 mm width x 120 mm

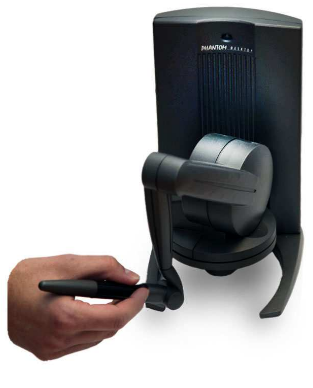


Figure 1.3: PHANToM Desktop 6/3-DOF haptic device

height x 120 mm depth. Positional sensing has resolution of more than 1100 DPI (0.023 mm) and the maximum force that the actuators can exert is approximately 8 N. More detailed specification can be found in [4].

1.4 The goals of the thesis

The main goals of this thesis are:

- present several methods of haptic rendering for 3-DOF and 6-DOF haptic devices
- propose a method of haptic rendering for 6/3-DOF haptic devices supported by a theoretical analysis possibly including pseudo-haptics
- implement the proposed method with support for a PHANToM Desktop haptic device
- prepare and perform a user study of the implemented algorithm and analyze its results

2. Haptic rendering

Haptic device sensors measure movement of mechanical parts and transform it to a signal which is transmitted to a computer. The computer receives the signal and processes it using a driver of the device.

The driver layer converts the data from the sensor space to Cartesian space using a forward kinematic algorithm. The position of an end-effector (stylus of the device) is then sent to a haptic API. There are several types of haptic APIs that provide different level of abstraction, from a low-level approach up to high-level interaction often including a scene graph [5].

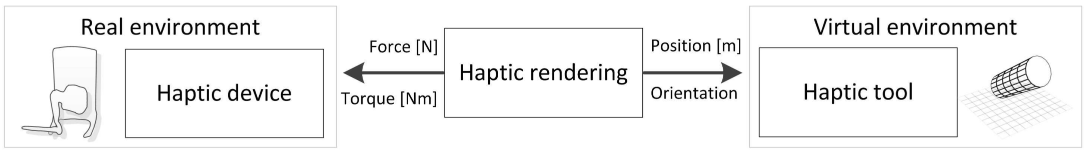


Figure 2.1: Haptic rendering illustration

Virtual objects with specific haptic properties are added to the scene graph together with a representation of the end-effector called a haptic tool. The haptic tool interacts with virtual objects and *interaction forces in Cartesian space are computed* and sent back to the driver.

The driver performs force mapping between the Cartesian space and the actuator space so the user finally feels the force feedback proportional to the haptic tool interaction. This process of generating forces from the haptic tool interaction in a virtual environment is called *haptic rendering*. Haptic rendering is the integral component of haptic simulation which connects the real and virtual environment as illustrated in Figure 2.1.

2.1 1-DOF haptic rendering and stability issues

A computer works as a discrete system, optical encoders often used as sensors in haptic devices perform quantization of position information and the virtual environment is frequently simulated by a discrete time system of dynamic states. All these discretization processes may induce stability problems which can not appear in a continuous space. For example, when bolting two steel beams, approximations of state variables may cause the entire assembly to exhibit sustained or growing oscillations, as stated in Colgate et al. [6]. Colgate et al. also stress that the human tactile sensory apparatus (shown in Figure 1.1) is extremely receptive to small amplitude mechanical vibrations in the 100 Hz - 1 kHz range (while vision is not).

Another problem lies in the limited dynamic range of mechanical impedances (also called Z-width) of the haptic device. The user should be able to move freely without any resistance in one moment and then feel full resistance in the next moment. Such a criterion can not be met with current devices and the problem of instability has to be resolved by defining criterion for the system such as a passivity criterion.

2.1.1 Passivity

One way to ensure stability of the system is to use a passivity design criterion. *Energy of the passive system* must be greater than or equal to negation of the initial energy during time t, i.e. it does not exceed negative initial energy:

$$\mathrm{E} = \int_0^{\mathrm{t}} \mathrm{P}(t)\,\mathrm{d}t = \int_0^{\mathrm{t}} \mathrm{F}(t)\mathrm{v}(t)\,\mathrm{d}t \geq -\mathrm{E_i}(0) \tag{2.1}$$

for power P, force F, velocity v and initial energy E_i in time 0 with the convention that energy absorption (e.g. resistance) is defined as positive energy. For zero initial energy $E_i(0) = 0$ the equation can be simplified to:

$$\int_0^{\mathrm{t}} \mathrm{F}(t)\mathrm{v}(t)\,\mathrm{d}t \geq 0 \tag{2.2}$$

There are many ways how to ensure passivity in a haptic system by mechanical methods, electrical methods, psychophysical methods or control methods [7].

Virtual coupling

Virtual coupling is one of such control methods. It connects the virtual haptic tool and the haptic device by an ideal spring defined by Hooke's law, and a damper (Figure 2.2). The method is analyzed for a 1-DOF model [8]: a virtual wall that generates resisting force proportional to penetration depth of the haptic tool. Colgate et al. proposed virtual coupling as an approach to guarantee passivity of a sampled-data system by limiting maximum impedance. In other words, the problem of mechanical stability is decoupled from the passive virtual environment [9].

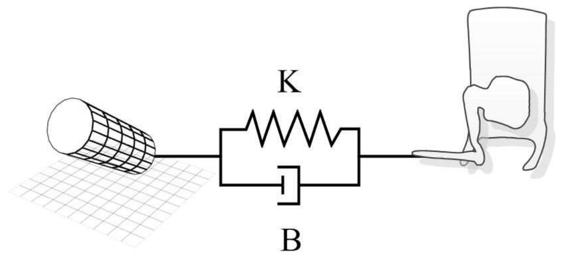


Figure 2.2: Virtual coupling spring-damper illustration

Results of the analysis showed that virtual stiffness K has to be inversely proportional to the sampling period T of a simulation:

$$\mathrm{b} > \frac{\mathrm{KT}}{2} + |\mathrm{B}| \quad \Rightarrow \quad \mathrm{K} < 2\,\frac{|\mathrm{B}| - \mathrm{b}}{T} \tag{2.3}$$

where B is a virtual damping coefficient and b is inherent mechanical damping of a haptic device.

A common standard is to run haptic simulation at a frequency of **1000 Hz** to achieve stable and realistic force feedback of a stiff virtual environment. Comparing this to a frequency of real-time rendering in computer graphics (25 - 60 FPS) makes the haptic rendering a great challenge because all computations have to be done in 1 ms.

2.2 3-DOF haptic rendering

In 3-DOF haptic rendering, a haptic tool that interacts with virtual environment is represented as a point or a sphere. The tool can interact with complex 3D scenes usually represented by polygonal models or voxel models. The output of the rendering is a force vector with a given magnitude and direction.

2.2.1 Direct rendering of vector fields

The virtual wall mentioned in the previous section was in fact a type of one-dimensional vector field. A vector field method specifies a repulsive force vector for every point in a scene. The force field vector is directed at the nearest resting position of the haptic tool. If the haptic tool is outside the object, the resulting force is zero, otherwise the force vector has a magnitude proportional to the penetration distance and regulates the stiffness of an ideal spring attached between the resting point and the haptic tool as shown in Figure 2.3. The force vector which is precomputed or determined analytically is then *directly rendered* to the haptic device.

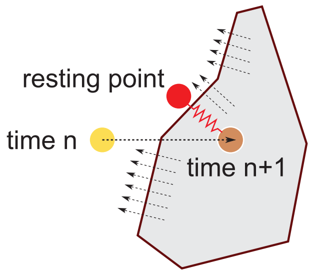


Figure 2.3: Vector force field illustration

This technique, however, has many drawbacks which make it useless for at least plausible simulations. As this method does not save a history of the haptic tool movement, discrete-time position update may result in unnoticed penetration through an object in one haptic loop step as shown in Figure 2.4 on the left. Another pop-through problem may come up when penetration is too deep and the desired nearest resting point is on the other side of the object as shown in Figure 2.4 on the right.

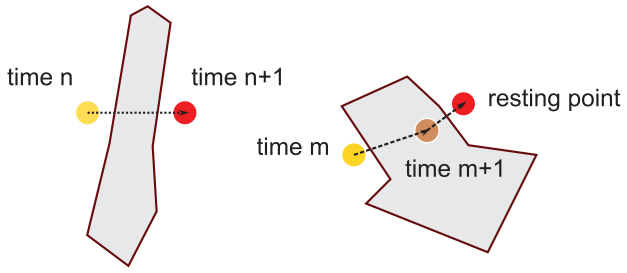


Figure 2.4: Pop-through problems

2.2.2 God-object method

To solve pop-through problems mentioned in direct rendering of vector fields the God-object method was proposed [10]. The god-object represents a virtual point (a proxy point) in the scene that is not able to penetrate into rigid bodies and thus behaves correctly. Zilles et al. use the term *haptic interface point* (HIP) to describe the position of the haptic device end-effector so that it is not confused with the god-object.

Position of the god-object is updated in every haptic loop step. When moving in free space, the position of the god-object is the same as the position of the HIP. If the HIP penetrates into an object, the movement of the god-object towards the HIP is constrained by the surface of this object and the resulting force is calculated by simulating an ideal mass-less spring (shown in Figure 2.5).

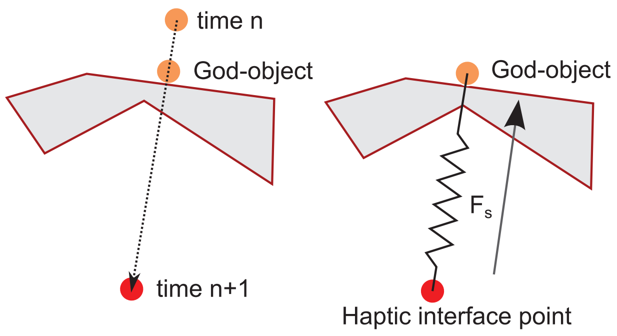


Figure 2.5: God-object method illustration

Geometric definition of proxy-based haptic rendering

The concept of the god-object can be also described as a geometric problem [11]:

$$\begin{aligned} \mathbf{q}(k) &= \mathcal{G}(K, \mathbf{p}(k), \mathbf{q}(k-1)) \\ \mathbf{f}(k) &= K(\mathbf{q}(k) - \mathbf{p}(k)) \end{aligned} \tag{2.4}$$

where k is a discrete-time index, $\mathbf{p} \in \mathbb{R}^n$ is the position of the haptic tool, $\mathbf{q} \in \mathbb{R}^n$ is the position of the god-object and K is the stiffness of the virtual coupling. The function $\mathcal{G} : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ is a geometric algorithm that accepts the current tool position $\mathbf{p}(k)$ and the god-object position $\mathbf{q}(k-1)$. The force function $\mathbf{f}(k) \in \mathbb{R}^n$ is determined by the position relation between the haptic tool and the god-object.

This geometric formulation may help programmers to understand and develop haptic rendering algorithms without notion of physical forces as stated in [11].

Constraint planes

One of the advantages of the God-object method is that the method works with triangular meshes used in graphic rendering. When the HIP moves, a line is traced from the position of the god-object to the HIP. Well known methods of ray tracing collision detection are used to determine if the line passes trough a triangular facet. Such a facet is called active when the HIP resides in a negative distance and the god-object in a positive distance in the direction of the facet surface normal. New position of the god-object is constrained by up to three active facets described as constraint planes:

$$P_n(x, y, z) : A_n x + B_n x + C_n z - D_n = 0 \tag{2.5}$$

Virtual spring as an energy function

Zilles et al. used an energy function minimization process to find the best location for the new position of the god-object. The energy function is described as potential energy of an ideal mass-less spring (Hooke's law: $\mathrm{F}(x) = -kx$) between the HIP and the god-object:

$$Q = -\int F(x)\,\mathrm{d}x = -\int -kx\,\mathrm{d}x = \frac{1}{2}kx^2 \tag{2.6}$$

Applying the HIP position (x_p, y_p, z_p) and the god-boject position (x, y, z) with spring stiffness $k = 1$ to equation 2.6 gives:

$$Q = \frac{1}{2}(x - x_p)^2 + \frac{1}{2}(y - y_p)^2 + \frac{1}{2}(z - z_p)^2 \tag{2.7}$$

Minimizing energy function using Lagrange multipliers

The task of finding the new god-object position given the HIP can be formulated as: *minimize Q with subject to P_n* (eq. 2.5, 2.7) which is a typical optimization problem that can be solved with Lagrange multipliers [12] using the following equation:

$$\nabla L(\vec{x}) = \nabla(Q + \sum_{i=1}^{n} \lambda_i P_i)(\vec{x}) = 0 \tag{2.8}$$

where λ_i is a Lagrange multiplier. Substituting Q and P_i gives:

$$L = \frac{1}{2}(x - x_p)^2 + \frac{1}{2}(y - y_p)^2 + \frac{1}{2}(z - z_p)^2 + \sum_{i=1}^{n} \lambda_i (A_i x + B_i x + C_i z - D_i) \tag{2.9}$$

which in the case of a 3-DOF problem and 3 constraint planes gives 6 variables $(x, y, z, \lambda_1, \lambda_2, \lambda_3)$. Partial derivatives can be easily derived to give a set of equations and the task of finding the new god-object position is simplified to finding a solution to these equations. Concave surface intersections require to iterate the whole process by using the new god-object position as the HIP position to find other possible surface constraints.

2.2.3 Virtual proxy method

Polygonal meshes often contain small surface gaps because of low-quality digitization or non-precise modeling. When the god-object enters a mesh through a small gap, the user gets stuck inside the mesh until they find the gap again. To resolve the problem we either fill in small gaps in the process of loading the mesh or we set a radius of the god-object in collision detection with constraint planes. The Virtual proxy method [13] proposes to treat the representation of the haptic tool in the virtual environment as a sphere (as shown in Figure 2.6).

The Virtual proxy method extends the God-object method also in rendering of surface properties like static and viscous dynamic friction, haptic textures adopting an idea of bump mapping in computer graphics and force shading of polygonal surfaces using interpolation of surface normals similar to Phong shading.

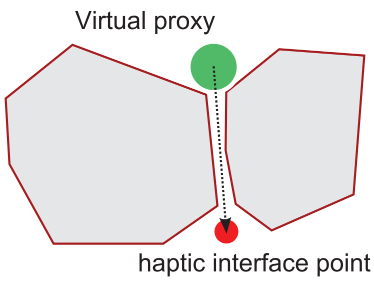


Figure 2.6: Virtual Proxy method illustration

2.3 6-DOF haptic rendering

Single-point interaction in the virtual environment may be sufficient for some tasks. However, there are a lot of virtual scenarios which require interaction and maneuvering with more a complex object. Imagine a virtual assembly task to test whether a component can be installed on a particular place when pushing it around different obstacles. The component may be long, thin, convex or non-convex and generate various types of haptic interaction response including force feedback and also torque feedback (a rotational force shown in Figure 2.7).

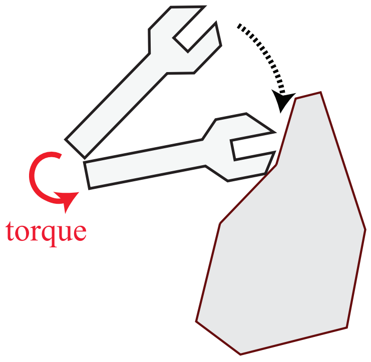


Figure 2.7: Torque illustration for a 6-DOF haptic tool

2.3.1 Direct rendering

Direct rendering methods do not simulate a proxy tool that is constrained by object boundaries as in the God-object method. Instead, the haptic tool position is equal to the position of the haptic device end-effector. Reaction to a collision response may not be sufficient and the tool may penetrate into an object which is visually distracting as illustrated in Figure 2.8. The collision response is usually proportional to the depth of the penetration as in 3-DOF direct haptic rendering of vector fields. Same problems as in 3-DOF methods may occur in 6-DOF rendering, e.g., a pop-through effect with thin objects.

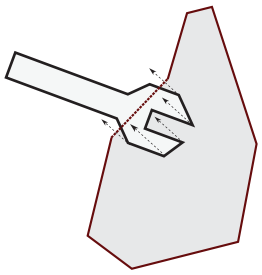


Figure 2.8: Haptic tool penetration in 6-DOF direct rendering

6-DOF direct haptic rendering pipeline

A typical direct haptic rendering algorithm pipeline consists of the following stages (as illustrated in Figure 2.9):

1. Retrieve information about position p and orientation q of the haptic device end-effector
2. Apply workspace transformations and set new position and orientation of the haptic tool
3. Perform **collision detection** between the tool and virtual environment
4. Generate an appropriate **collision response** including force F and torque τ feedback
5. Stabilize the force feedback and torque feedback (optional) and send it to the haptic device

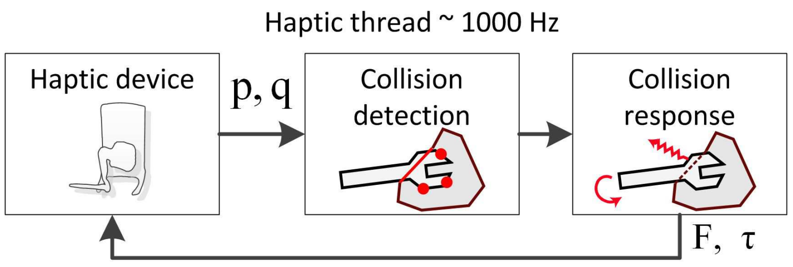


Figure 2.9: 6-DOF direct haptic rendering pipeline illustraton

Performance issues

The first performance problem arises at the 3rd stage: collision detection. Detecting collisions for a point is far less computationally intensive than detecting collisions for a complex object. The collision detection itself may take more than 1 ms to compute. Various hierarchical data structures optimized for sequential and coherent queries have been researched and implemented [14].

Collision response is also a very difficult task involving generation of force and torque feedback for multiple contacts. Discrete-time solvers often generate unrealistic feedback with discontinuities and different methods of force smoothing have to be applied.

Haptic rendering of polygonal models

Gregory et al. [15] proposed a method for 6-DOF haptic rendering of polygonal models using a virtual proxy. A similar approach of constraining the proxy as in the god-object method is used.

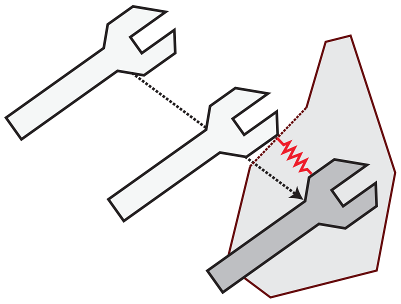


Figure 2.10: 6-DOF Virtual Proxy

However, the algorithm of finding the best proxy location can not be simply adopted. The location of the proxy is constrained by the surface location so that the actual position of an end-effector locally minimizes the penetration depth (Figure 2.10). This is done by predicting the location of collision occurrence using interpolation of the current and previous states of the proxy and end-effector. Polygonal models are decomposed into convex pieces for which pairs of closest features are tracked to estimate the penetration depth.

The method works well (i.e. maintains frequency of 1000 Hz) for tens of convex pieces which might be insufficient for some applications. Another problem arose for collision response which is computed using the penalty method with force proportional to the magnitude of the estimated penetration depth which may vary, thus generating distractive forces. Gregory et al. adopted the force shading technique from [13] by interpolating force normals; forces are also smoothed between successive frames.

2.3.2 Simulation-based haptic rendering

The previous algorithm did not apply any control method to stabilize the force and torque feedback. Direct rendering of force and torque computed using penalty methods often causes mechanical instability, e.g. magnitude of the force is greater than the maximum force exertable by the haptic device. Lower collision response stiffness is set to maintain stability at the expense of a less realistic haptic feedback. Moreover, deeper penetration of the tool may even cause an undesirable pop-through effect.

Virtual coupling presented in section 2.1 solves stability issues by filtering artificial energy which is characterized by high frequency changes in force magnitude and direction. The filtering, though, may be visually apparent and distracting. Virtual coupling may also introduce an apparent damping force in free space as a result of unbalanced dynamic simulation. Simulating dynamics of the virtual coupling also involves application of numerical methods that increase computational load. On the other hand, the benefit of virtual coupling is in enhancement of the penalty-based force feedback by setting a high stiffness of a contact force field which minimizes penetration of the haptic tool.

Multi-rate methods

Otaduy et al. [16] presented a method capable of stable and responsive 6-DOF haptic rendering of complex (tens of thousands of triangles) polygonal models using virtual coupling. A dynamic proxy is simulated using rigid-body dynamics based on an implicit integration of Newton-Euler equations where the rest of the environment is considered to be static. The implicit integration ensures the stability of penalty-based force feedback.

Collision detection is performed using "sensation preserving simplification algorithm" that uses static LOD (level of detail) technique for collision queries between two polyhedral objects. A multiresolution hierarchy where tree nodes may be used as approximate bounding volumes is generated to get faster processing of collision detection. The query may return multiple collision contacts so K-means clustering is applied to mitigate discontinuities. Nevertheless, analysis showed that performance of the algorithm is still insufficient for stable haptic rendering which lead to decoupling of the collision detection from the haptic thread.

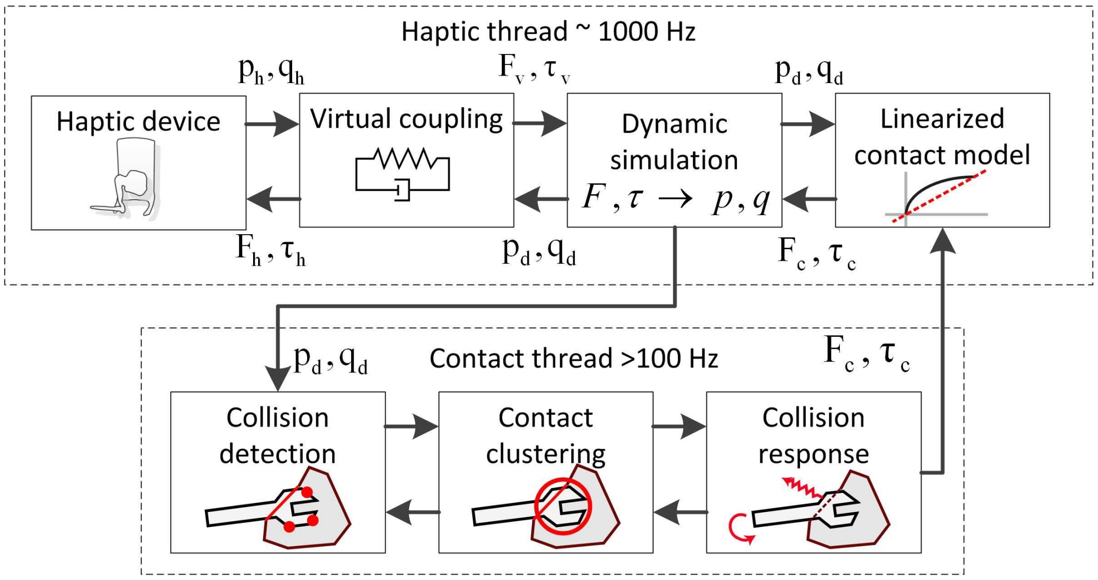


Figure 2.11: Haptic rendering pipeline of [16] illustration

Otaduy et al. decided to linearize contact forces that are computed in a special *contact thread* as shown in Figure 2.11. Algorithms that involve decoupling of haptic rendering pipeline parts into separate threads that run at different frequencies are called multi-rate algorithms. For the referred algorithm and a complex testing scenario the contact thread frequency varied from 100 Hz to 700 Hz.

Collision response is computed using viscoelastic penalty-based force and torque feedback. Stability of the response is very well managed by implicit integration and contact clustering. Nevertheless, discontinuities in torque can arise, for example, when large flat parallel surfaces are in contact as stated in [16].

Constraint-based algorithms: 6-DOF God-object

Penalty methods can not guarantee that the haptic tool does not penetrate into an object. Local force models also do not generate exact realistic force and torque feedback. Ortega et al. [17] extended the 3-DOF God-object method to 6-DOF using a constraint-based quasi-static approach. The method completely prevents interpenetrations and generates realistic and stable force feedback with the correct direction. However, active constraints can be potentially updated at low frequencies which might lead to filtering of high-frequency details.

Voxel sampling methods

All the algorithms presented above struggle with computational load of exact contact determination and collision response of polygonal models. One way to cope with this problem is to use different representation of virtual objects in the scene.

McNeely et al. [18] proposed a scalable method that could achieve a relatively stable 6-DOF haptic rendering on a 350 MHz processor. It was one of the first 6-DOF haptic rendering algorithms applicable to real commercial use. The main idea of the method is to voxelize the static virtual environment and represent the haptic tool by a set of surface points. When a point crosses an object surface voxel boundary, a repulsive force is exerted and the collision response is computed. The thesis will follow with a detailed explanation of this method in the next chapter.

3. 6/3-DOF Haptic rendering

The previous chapter described selected haptic rendering algorithms for symmetric devices, i.e. devices with an equal number of sensors and actuators. The complexity of 6-DOF haptic devices with support of force and torque feedback makes such devices costly and a wide variety of under-actuated devices exist due to the low cost of integrating additional sensors.

Barbagli et al. [19] formally defined the problem of sensor/actuator asymmetry and provided a framework for under-actuated device analysis. Two 2-DOF examples showed that it is possible to correctly perceive missing a degree of freedom to some extent which users marked with a level of realism ranging from almost perfect to completely unrealistic. Despite concluding that asymmetric devices have limited usability for certain situations demanding a hight realistic force feedback, a proposition of future work in haptic rendering that would alleviate limitations of asymmetric devices was mentioned mainly due to broad use of such devices.

Verner et al. [20] examined 3-DOF force only feedback in a 6-DOF manipulation experiment consisting of simple drawing and tracing tasks. Experiment results showed that *force and torque can be well approximated by force only feedback* in simple tasks. A torque only feedback also did not completely fail in experiment evaluation, but the overall impression was negative. Verner et al. also pointed out that *asymmetric haptic devices are not passive* which makes it even harder to achieve stability of the system that relies on the passivity criterion.

A lot of work has been done in investigating asymmetries in telemanipulation. This thesis, however, focuses on developing a moderately stable **haptic rendering algorithm for non-trivial polygonal models interacting with a 6-DOF haptic tool and providing 3-DOF force only feedback**. The key concept is to extend existing 6-DOF haptic rendering algorithms by applying methods that restrain instability and provide other means of torque simulation. These may include several types of additional visual information, mapping from the torque space to the force space and other methods of haptic illusion or cognitive interpretation of a visuo-haptic simulation both representing a concept of *pseudo-haptics* [21].

This chapter will describe the main components of our 6/3-DOF haptic rendering algorithm and decisions made during the analysis and development.

3.1 Trivial implementations

Choosing an appropriate haptic rendering algorithm without any previous experience of implementing 6-DOF haptic rendering was not an easy task. To the best of my knowledge there is also no freely available open source implementation of a 6-DOF haptic rendering algorithm. Some authors publish haptic rendering demo executables to make it possible to compare algorithms. Unfortunately, demos are available only for specific platforms and device families which make the comparison actually impossible. Because of this, it was desirable to implement a trivial prototype that would provide necessary information and a testing environment for missing torque feedback.

3.1.1 Mesh-mesh collision

The first intuitive approach was to detect collisions of two meshes represented by a set of triangles. The simplest brute force collision detection algorithm iterates through all triangles of the tool mesh and performs triangle vs. triangle collision tests with all triangles of other meshes in the scene. The algorithm returns a set of colliding triangles between two meshes.

This information is not sufficient for an explicit computation of collision response. A new problem is to find exact contact points and a force response vector. The trivial implementation did not save a position history of the tool mesh, nor velocity or any other kinematic quantity for simplicity reasons. A different way of determining contact points and force response directions and magnitudes had to be employed.

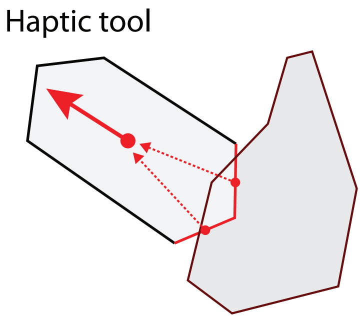


Figure 3.1: Mesh-Mesh collision response approximations

An assumption that a colliding triangle is moderately compact allowed to use an approximate position of a contact point computed as the center of the colliding triangle. Force direction is also approximated using a local force model that computes the force response direction of every colliding triangle as a vector from the approximated contact point to the center of mass of the haptic tool mesh. The resulting direction is the sum of all force direction vectors from all colliding triangles (as shown in Figure 3.1). Magnitude of the force is set constant which eliminates expensive computation of penetration depth but introduces discontinuities so the resulting force needed to be artificially smoothed.

There are many situations where these approximations generate unrealistic force feedback. The local force model also assumes that the haptic tool mesh is convex. However, this trivial prototype revealed that a lot of rotational movements that should normally cause both the force and torque feedback can be, to some extent, well approximated just with the force feedback.

3.1.2 Proxy point set collision

Another prototype approach was inspired by the God-object method described in section 2.2 but in a rather simpler way than the 6-DOF God-object method (mentioned on page 16). One of the main advantages of developing 3-DOF haptic rendering over 6-DOF haptic rendering is the fact that computing a point interaction is much simpler. It is, therefore, of no surprise that many reasearches have chosen to trade accuracy for simplicity and increased performance by approximating the surface of the haptic tool by a point set.

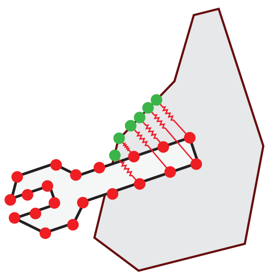


Figure 3.2: Point set collision

There are many ways how to perform point sampling of the mesh and these will not be discussed here. One particular method using space subdivision will be described later in this chapter.

The prototype algorithm has two initialization stages. The first step is to generate a surface point set of the haptic tool mesh stored in mesh local coordinates. The second step is to initialize a proxy point set corresponding to the surface point set stored in global coordinates. The haptic rendering loop has the following stages:

1. For each point in the surface point set: determine the current global position of the surface point using a rotation matrix and position vector obtained from the haptic device

2. For each proxy point: trace a line from the i-th proxy point to the current global position of the i-th surface point

3. Detect line-mesh collisions (i.e. collisions of proxy points with the virtual environment) and store a collision point and collided polygon normal vector for each collision

4. If no collision occured, set all proxy points' positions to current global positions of surface points and jump to the start of the loop

5. If a collision occured, set the proxy point position to the collision point (acutally a little above the collision point in the direction of the normal vector to prevent pop-through of the proxy due to rounding errors) and calculate a force contribution proportional to the penetration depth of the current global position of the surface point as shown in Figure 3.2

6. The resulting force is the sum of all force contributions

Although rendering stability was much improved compared to the approximate local forcel model of the previous prototype, the resulting behaviour felt as if the haptic tool surface was sticky. The reason for this is that the proxy point remains at the colliding point unless the user moves the haptic tool away from the object, which makes it impossible to slide the haptic tool along the surface of the virtual object.

3.1.3 Rendering of surface properties

A straightforward solution to the sticking problem is to move the proxy point along the surface in a direction obtained by projecting the penetration vector on the colliding polygon plane. Various surface properties can be simulated now as shown in Figure 3.3. A slippery surface effect would move the proxy point to the endpoint of the penetration vector projection. A friction effect would move the proxy point partially along the projected vector (e.g. to the midpoint).

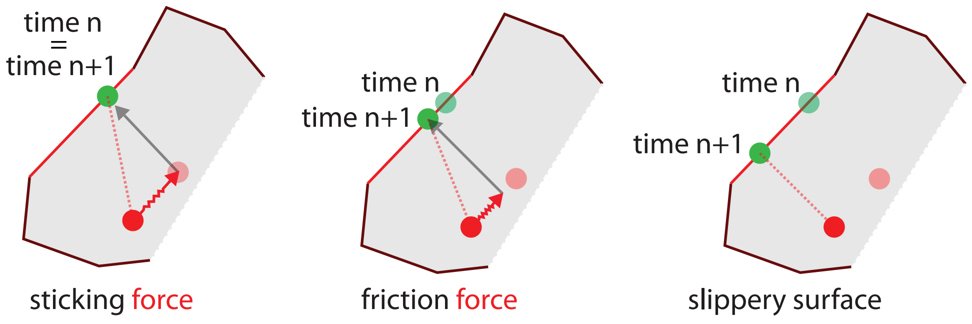


Figure 3.3: Surface

However, the slipping effect has one major drawback. When proxy points have the ability to slide along the surface, compactness of the haptic tool representation is lost. This means that for a specific case, the repulsive force is easily overcome and proxy points tend to surround the colliding object (as shown in Figure 3.4).

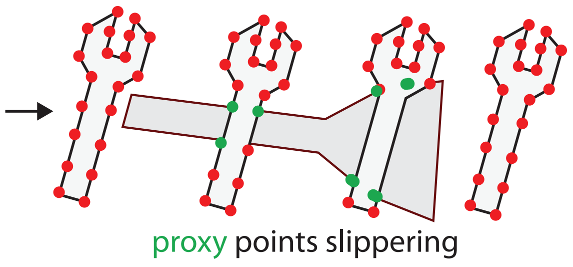


Figure 3.4: Surface drawback

This problem just emphasizes the complexity of 6-DOF haptic rendering. A configuration solver for each proxy point with compactness constraints would have to be applied to solve the problem. On the other hand, this prototype provided a lot of valuable information about user behaviour when interacting with force only feedback.

3.2 Philosophy of the final approach

The previous chapter gave an overview of selected haptic rendering algorithms mainly for interaction with polygonal models. The God-object method for the 3-DOF case is considered to be one of the best methods that is sufficient for a wide variety of haptic applications. Unfortunately, this can not be said about a 6-DOF case which is still too computationally demanding and no 6-DOF haptic rendering algorithm is considered to be widely usable. All of the algorithms mentioned employ some kind of approximation or linearization which make them unusable for certain applications.

The algorithm for 6/3-DOF haptic rendering used in this thesis is based on the *voxel sampling* method proposed by McNeely et al. [18] for 6-DOF rendering. The method is not perfectly accurate in a geometrical way and it may occasionally suffer from pop-through effects because of the penalty based collision response but the implementation is not excessively complex and the algorithm can achieve sufficient haptic thread frequency.

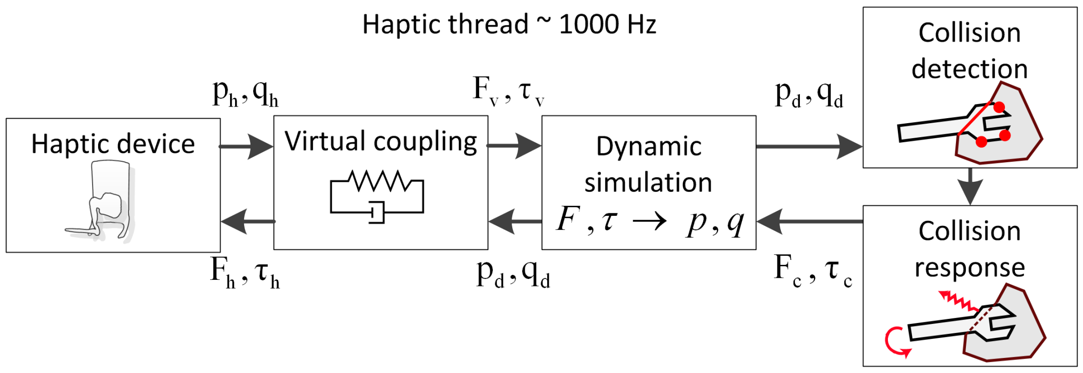


Figure 3.5: Haptic rendering pipeline of the *voxel sampling* method proposed by McNeely et al. [18]

The voxel sampling method is composed of the following parts used in the haptic rendering pipeline as illustrated in Figure 3.5. All parts will be discussed in detail in later sections:

1. Collision detection
2. Collision response
3. Rigid body dynamics
4. Virtual coupling

The chapter will then continue with a description of pseudo-haptics and techniques used to preserve stability for a non-passive 6/3-DOF haptic device and methods to improve artifical torque sensation using multimodal information.

At the end of this chapter, a tool simulating the behavior of a haptic device will be introduced. This tool is called Virtual Phantom and it eases the process of debugging haptic rendering.

Implementation details are described in Appendix A.

3.3 Collision detection

As mentioned before, this method represents the haptic tool and the virtual environment in non-polygonal structures. Data preprocessing is therefore the first step in the initialization process.

The virtual environment, which is considered to be static, is represented by *voxels*, i.e. a regular 3D grid of values. Voxel size determines accuracy of the representation. The accuracy rises with a smaller value of the voxel size (as shown in Figure 3.6) but the performance of the algorithm decreases rapidly. It should also be noted that haptic devices have limited sensing resolution which inherently introduces a minimum limit on the voxel size. For example, Phantom Desktop used during the creation of this thesis has sensing resolution of about 0.023 mm. However, current voxel sampling algorithms can not achieve sufficient haptic thread frequency for such accuracy in non trivial scenes.

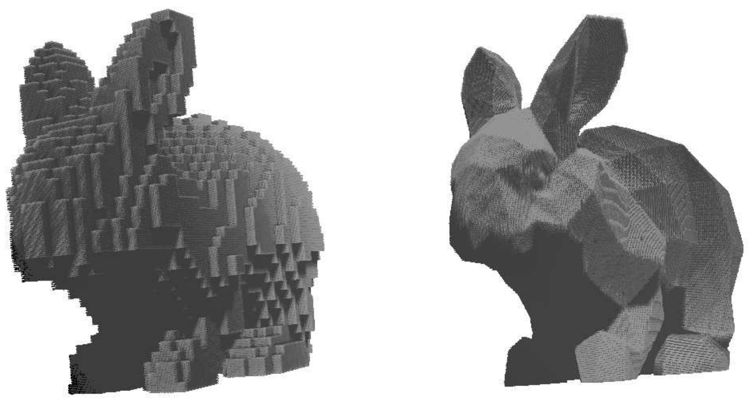


Figure 3.6: Voxel representation of a polygonal model with different voxel sizes

Another factor is the quality of the polygonal model. It can be easily observed that the polygonal model used for generating the voxel representation in Figure 3.6 is a *low poly* model (i.e. has a small number of polygons). A smaller voxel size representation (i.e. rabbit on the right) reveals visible boundaries between adjacent polygons which can not be observed on the left rabbit.

The haptic tool is represented by a point set (illustrated in Figure 3.7) in the same way as described in the previous section about trivial implementations. Properties similar to the voxel representation apply to the point set. Higher density of the point set makes the haptic tool interaction more realistic but computationally intensive.

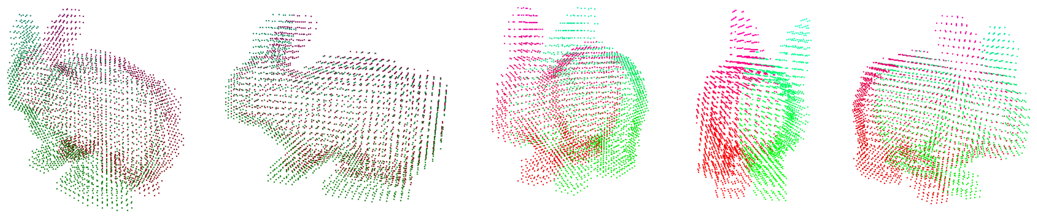


Figure 3.7: Point set representation of a polygonal model

3.3.1 Voxel tree

Storing the voxel representation in a 3D array would be inefficient in terms of memory usage. Let's assume that we want to represent objects in a workspace of 1 cubic meter with an accuracy of 1 millimeter and that a voxel value is stored in 1 byte. The 3D array would consume approximately 1 gigabyte. Storing a 4-byte float value in a voxel would consume almost 4 gigabytes.

Considering that a typical mesh representation stored in the 3D array is sparse, it is convenient to create a hieararchical data structure that stores voxels in a more efficient way. McNeely et al. proposed an octree-like subdivision of space and pointed out that tree depth has to be limited in order to keep the frequency of haptic thread as stable as possible (i.e. invariant to position and orientation of the haptic tool). A subdivision of a cubical volume node into 8 x 8 x 8 subnodes was empirically determined to be the most memory efficient for this purpose.

The term *VoxMap* is used by McNeely et al. for merged voxel trees of all objects in the scene. The VoxMap approach makes it impossible to differentiate between objects. This, however, is not a significant problem since the algorithm does not utilize object-specific information anyway.

Generating a VoxMap

The voxel tree is composed of axis-aligned cubical volumes. A mesh representing an object in the virtual environemnt is assumed to be a triangle mesh. The first step is to create a voxel representation of the mesh surface.

The algorithm traverses the voxel tree and performs intersection tests between cubical volumes and triangles of the mesh until it reaches a leaf node represented by a voxel. A method of finding a *separating axis* [22] (from Hyperplane separation theorem) between AABB (axis-aligned bounding box) and triangle is used. The triangle and the box is projected onto a potential separating axis and tested for overlap as illustrated in Figure 3.8. The algorithm ends when a first separating axis is found (i.e. triangle and the box do not overlap) or all 13 axes (3 AABB face normals, 1 triangle face normal, 9 edge cross products) have overlaps, which means that the objects are intersecting.

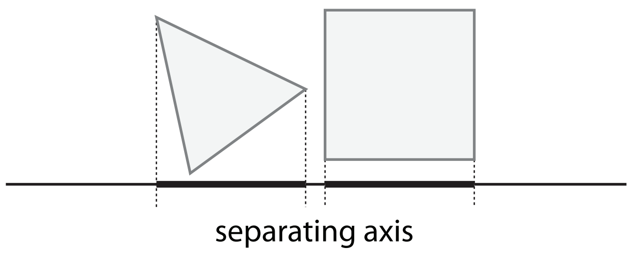


Figure 3.8: Separating axis illustration

The voxel tree is a *sparse voxel representation* of the polygonal mesh. A tree node points deeper only at those child nodes that contain data at the leaf level. The voxel tree structure for the rabbit polygonal mesh and a sagittal slice of the tree middle level is shown in Figure 3.9 and 3.10.

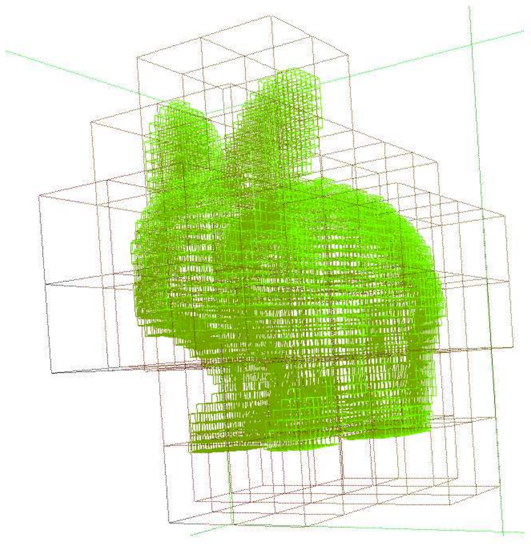
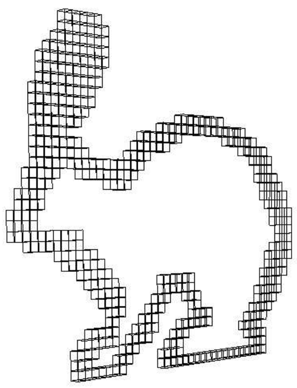



Figure 3.9: Voxel tree structure (left) and a sagittal slice of the tree level (right)

Voxel data types

The interface of the tree is completely transparent and provides *at* and *set* methods as if the structure was a 3D array. A bit array storage would be completely sufficient to determine the free space and interior of the scene. However, for the purpose of haptic rendering and especially for the collision response part described in the next section, there are more auxiliary voxel types:

1. Undefined - set in the process of Voxel tree initialization
2. Free space - an empty voxel, i.e. does not correspond to any object
3. Surface - the surface of the mesh, generated by separating axes interpenetration test
4. Interior - interior of the object
5. Proximity - an object proximity layer surrounding the surface

When a tree node contains all children of the same voxel data type, aggregation of the voxel data can be applied. An aggregate node behaves in the same way as a normal node but it does not point to children. Instead, it stores the voxel data type only once.

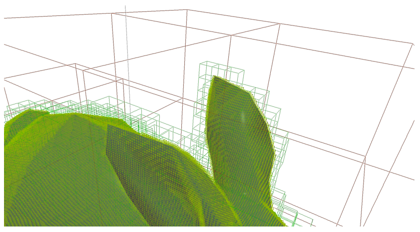


Figure 3.10: A rendered polygonal mesh with a Voxel tree visualization

Interior voxelization

The voxel tree still contains only mesh surface voxels. The next step is to voxelize the *interior* of the mesh and set other voxels that our outside the mesh to the *free space* type. A common approach is to use a flood fill algorithm. The first problem may be determining a starting point of the flood fill. For example, there is absolutely no guarantee that a voxel at the center of mass is in the interior of the object. The second problem might be that if a mesh was not watertight because of low-quality digitization, the process would fail.

A simple approach that works very well for convex objects is *bounding box wall propagation*. Six bounding walls propagate free space one by one towards the center until a surface voxel is reached. Problem arises for non convex objects with "bays". A bay is a spot missed by all propagating bounding walls. Iterative wall propagation solves the problem. The number of iterations depends on complexity of the mesh which can sometimes be unbearable (e.g. for a hole in the object of a spiral tube shape). However, accuracy of voxelization for such shapes is not essential for the purpose of this thesis and the number of iterations is therefore limited. An illustration of interior voxelization is shown in Figure 3.11.

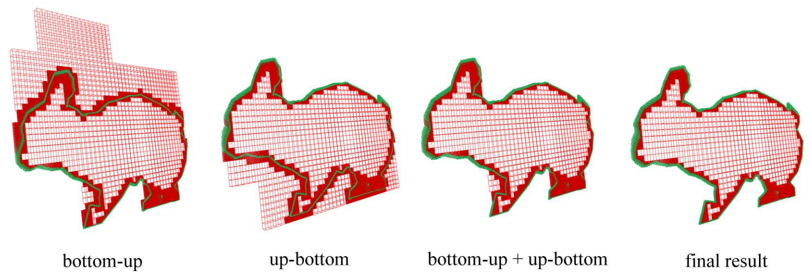


Figure 3.11: Voxelization by iterative wall propagation

Haptic tool vs VoxMap

Collision detection between the haptic tool and the virtual environment is done by checking collisions for every point of the point set representing the haptic tool against the VoxMap. This involves a simple collision test between a box with center c and extent e vs. point p. Point p is translated by $-c$ and tested against an interval $(-e, +e)$ in each dimension as illustrated in Figure 3.12.

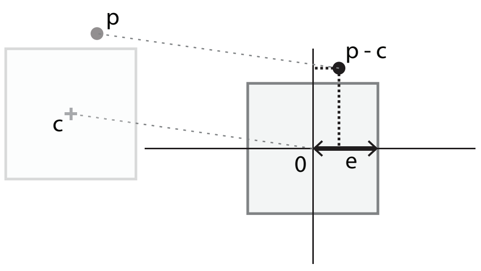


Figure 3.12: A simple Point vs. Box collision detection

3.4 Collision response

The goal of the collision response is to **generate a force** F **and torque** τ using the output of collision detection which is a list of {colliding point, colliding voxel} pairs. The haptic rendering algorithm uses a partially extended local force model called *tangent-plane force model* presented by McNeely et al. The force model does not compute an exact contact normal. Instead, it uses precomputed vectors stored in a *PointShell* representation.

3.4.1 PointShell representation

Firstly, generating a point set representing the haptic tool is not a trivial task. Places with higher curvature should ideally be covered with more points to preserve details of the haptic tool mesh where necessary. Also, there shouldn't be holes in the representation as this could cause partial interpenetration of polygon meshes still used for visual rendering. The easiest approach is to use a regular grid point sampling of the surface which is in fact the surface voxelization where all generated voxel centers define the surface point set.

The point set representation is extended into a *PointShell* representation by adding inward-directed vectors at positions of points from the point set. The PointShell is therefore composed of surface points and corresponding surface normals of the mesh representing the haptic tool. Unfortunately, usage of surface normals from the mesh representation does not create a good PointShell as shown in Figure 3.13 on the left (surface normals are directed outwards for the sake of visualization). It can be observed that normals at the tip of the rabbit's ears, especially on the top edge, are not directed inwards, which causes problems for the tangent-plane force model.

A simple, yet effective way to generate normals that are directed inwards even on edges is to use the voxel representation itself and compute normals from neighboring voxel occupancy. The result can be seen in Figure 3.13 on the right. The method can easily be enhanced by extending the neighborhood from which the normal vector is computed.

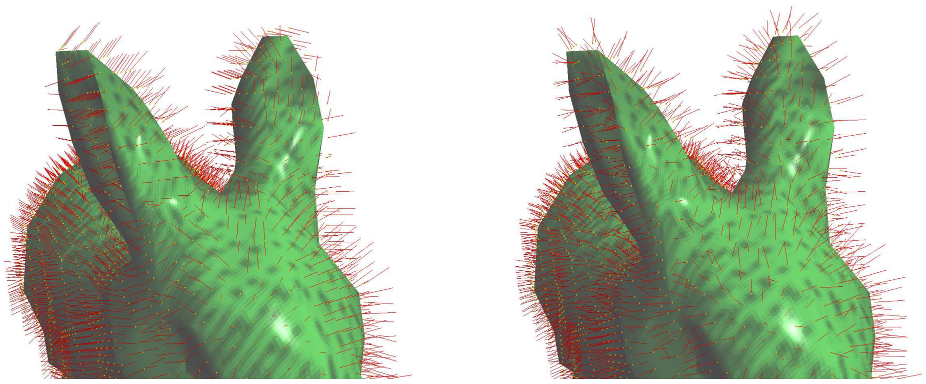


Figure 3.13: PointShell normals generated from polygon normals (left) and from voxel representation (right)

3.4.2 Tangent-plane force model

A voxel *tangent plane* T is a plane defined by a *voxel center* c and an inward-directed *surface normal vector* n that is associated with a *point* p from the PointShell representation:

$$T(x): \; n \cdot (x - c) = 0 \tag{3.1}$$

Penetration depth d is computed as the distance between the tangent plane and the point p from the PointShell. A *force contribution* F_p is an ideal spring force defined as:

$$F_p = -k_s d = -k_s(n \cdot (p - c)) \tag{3.2}$$

where $\cdot$ is the dot product operator and k_s is force field stiffness. An illustration of the tangent-plane force model is shown in Figure 3.14.

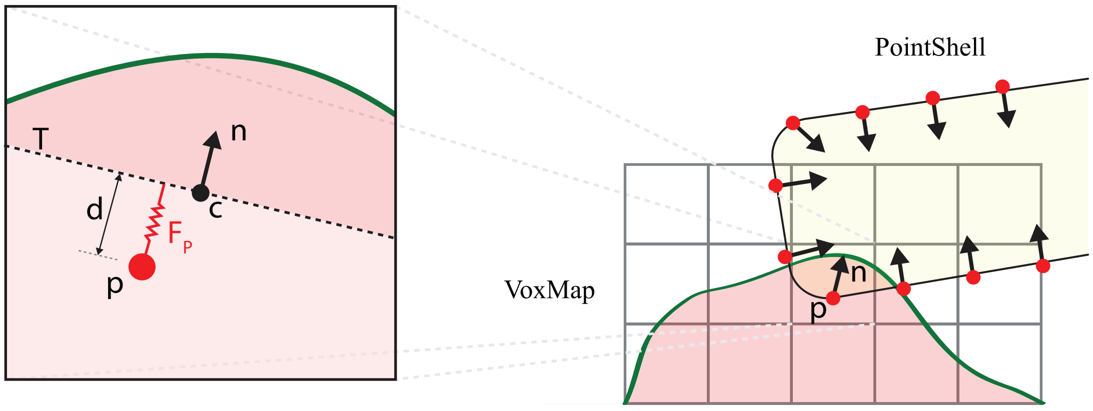


Figure 3.14: Tangent-plane force model

Computing force contributions in surface type voxels may cause distracting interpenetrations of the haptic tool mesh with the virtual environment. The proximity voxel layer mentioned in the previous section is therefore used in the force model. If a point from the PointShell crosses the proximity voxel boundary deeper to a surface voxel layer, the force contribution is computed as:

$$F_p = -k_s \frac{s_v}{2} \tag{3.3}$$

where s_v is a voxel size. Also the maximal penetration depth d in the proximity layer is clamped to an interval $\langle 0; \frac{s_v}{2} \rangle$ to eliminate discontinuities. If the user overcomes the force reaction and the point p penetrates into the interior, the force contribution F_p is zero.

The total force F generated from the PointShell penetration is the sum of all force contributions F_p and the total torque τ is the sum of all torque contributions τ_p:

$$\begin{aligned} F &= \sum_{p \in P} F_p \\ \tau &= \sum_{p \in P} \tau_p = \sum_{p \in P} (p - p_{com}) \times F_p \end{aligned} \tag{3.4}$$

where P is the set of points of the PointShell, p_{com} is the center of mass of the haptic tool and $\times$ is the cross product operator.

3.4.3 Force filtering

A great advantage of the tangent-plane force model is its computational simplicity and efficiency. However, the approximation of the contact normal vector by the PointShell surface normal n may cause force response discontinuities, e.g. when sliding along the surface and crossing voxel boundaries with the tangent plane tilted as illustrated in Figure 3.15. This problem is mitigated by the virtual coupling which filters high frequency changes in force magnitudes.

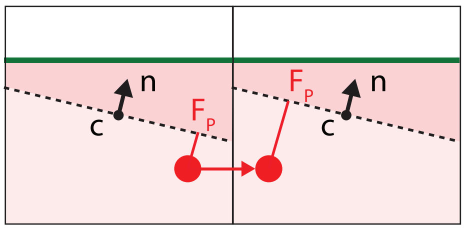


Figure 3.15: Voxel boundary crossing force discontinuity

Clustering and averaging force contributors

When too many PointShell points penetrate into proximity voxels, the total force F might exceed the maximum force that can be exerted by the haptic device. Some algorithms solve this problem by clustering force contributions (e.g. using K-means clustering). Another method is to average force contributions for a certain number of contributors. McNeely et al. proposed to average the total force if the number of contributors is more or equal than 10. The algorithm used in the thesis responded best when averaging for more than 50 contributors:

$$\begin{aligned} F_a &= F \qquad for N < 50 \\ F_a &= \frac{F}{N/50} \;\; for\ N \geq 50 \end{aligned} \tag{3.5}$$

where F_a is the averaged total force and N is the number of force contributors.

Direct force rendering

This collision response part of the haptic rendering algorithm can be directly tested with a haptic device as it is a separate unit producing force F which is an input of the haptic device. However, direct rendering of the total force F with discontinuities causes the haptic device to produce unpleasent vibrations and overall impression is negative. A common way to stabilize the force response is to use virtual coupling as described in the previous chapter. Virtual coupling is implemented using a simulation of rigid body dynamics.

3.5 Rigid body dynamics

A *virtual dynamic object* is an object representing the haptic tool in the virtual environment that is *coupled with the haptic device motion.* This object interacts with the virtual environment and its dynamic behaviour is simulated using rigid body dynamics. The simulation is required to compute the position and orientation of the dynamic object in time with regard to applied force and torque. Derivation of equations 3.6 and 3.9 can be found in [23]. The following notation is used:

- F - total force acting on a rigid body in time t
- τ - total torque acting on a rigid body in time t
- $x(t)$ - position of a rigid body in time t defined by a vector
- $q(t)$ - orientation of a rigid body in time t defined by a quaternion
- m - constant mass of a rigid body
- I - constant moment of inertia of a rigid body
- $v(t)$ - linear velocity of a rigid body in time t
- $\omega(t)$ - angular velocity of a rigid body in time t
- $p(t) = mv(t)$ - linear momentum of a rigid body in time t
- $L(t) = I\omega(t)$ - angular momentum of a rigid body in time t

Newton-Euler equations

The motion of a rigid body can be described using Newton-Euler equations of motion:

$$\begin{aligned} \mathrm{F} &= \frac{d\mathrm{p}}{dt} = \frac{d}{dt}(m\mathrm{v}) = m\frac{d\mathrm{v}}{dt} = m\mathrm{a} \\ \tau &= \frac{d\mathrm{L}}{dt} = \frac{d}{dt}(\mathrm{I}\omega) = \mathrm{I}\frac{d\omega}{dt} = \mathrm{I}\alpha \end{aligned} \tag{3.6}$$

where a is acceleration of the rigid body and α is angular acceleration. A general equation for describing rotational dynamics ($\tau = \omega \times (\mathrm{I}\omega) + \mathrm{I}\frac{d\omega}{dt}$) could be simplified because of a constant moment of inertia I in time. This is due to a limitation of virtual coupling formalism of torsional spring described in the next section. The dynamic object is therefore represented in the simulation as a sphere with radius r of a uniform density defined by an inertia mass tensor:

$$I = \begin{pmatrix} \frac{2}{5}mr^2 & 0 & 0 \\ 0 & \frac{2}{5}mr^2 & 0 \\ 0 & 0 & \frac{2}{5}mr^2 \end{pmatrix} \tag{3.7}$$

3.5.1 Explicit integration of Newton-Euler equations

Let $\mathbf{X}(t)$ represent a *state of a rigid body* in time t. The simulation is computed at a minimal frequency of 1000 Hz and begins at $t = 0$ with a given state $\mathbf{X}(0)$. Simulation time t is advanced by a time step Δt. One simulation cycle is defined as:

$$S : (F, \tau, \mathbf{X}(t)) \rightarrow \mathbf{X}(t + \Delta t) \tag{3.8}$$

One method to compute $\mathbf{X}(t + \Delta t)$ is to use explicit numerical integration of Newton-Euler equations 3.6. The state vector of a rigid body contains position $x(t)$, orientation $q(t)$ and another two variables describing the dynamic state. Although using velocity variables in the state of the rigid body may be intuitive for the linear case (as shown in equation 3.11), momentum variables are used for simplicity reasons and better numerical stability [23]. The state vector of a rigid body and its corresponding time derivative is defined as:

$$\mathbf{X}(t) = \begin{pmatrix} x(t) \\ q(t) \\ p(t) \\ L(t) \end{pmatrix} \quad \frac{d}{dt}\mathbf{X}(t) = \begin{pmatrix} v(t) \\ \frac{1}{2}\omega(t)q(t) \\ F(t) \\ \tau(t) \end{pmatrix} \tag{3.9}$$

The time derivative of the state vector is used to approximate the state of the rigid body in the next step. By applying Euler's method we compute the current state (denoted with subscript 1) and auxiliary variables from the previous time step (denoted with subscript 0) as follows:

Euler integration of translational variables

$$p_1 = p_0 + F\Delta t \tag{3.10}$$

$$v_1 = v_0 + a\Delta t = v_0 + \frac{F}{m}\Delta t = \frac{p_0}{m} + \frac{F\Delta t}{m} = \frac{p_0 + F\Delta t}{m} = \frac{p_1}{m} \tag{3.11}$$

$$x_1 = x_0 + v_0\Delta t \tag{3.12}$$

Euler integration of rotational variables

$$L_1 = L_0 + \tau\Delta t \tag{3.13}$$

$$\omega_1 = I_1^{-1}L_1 \tag{3.14}$$

$$q_1 = q_0 + \frac{1}{2}\omega_0 q_0 \Delta t \tag{3.15}$$

3.5.2 Stability of integration and semi-implicit integration

Explicit integration equations 3.10 - 3.15 can be summarized as:

$$\mathbf{X}(t+\Delta t) = \mathbf{X}(t) + \frac{d}{dt}\mathbf{X}(t)\Delta t \tag{3.16}$$

The problem of an explicit integration solution is a high approximation error caused by linearization of the derivative term. The explicit method generates aritifial energy which leads to instability. The problem may be reduced by lowering the time step (i.e. increasing frequency). However, this is often not possible due to collision detection and collision response computational performance. Colgate et al. [8] stated that it is essentially impossible to guarantee stability of the system when using an explicit method together with a penalty-based collision response.

Implicit integration

An implicit integration method can be defined as:

$$\mathbf{X}(t+\Delta t) = \mathbf{X}(t) + \frac{d}{dt}\mathbf{X}(t+\Delta t)\Delta t \tag{3.17}$$

The derivative term here conserves the energy, thus implicit integration provides more stable behavior. However, the derivative term can not be directly evaluated as in the explicit case and the integration leads to solving a set of implicit algebraic equations which can be time consuming.

Example of an implementation using a linear approximation of 3.17 is provided by Otaduy et al. [16] who implemented the implicit integration involving linearization of virtual coupling force and torque.

Semi-implicit integration

Semi-implicit integration is a method that is not as accurate as the implicit method, but provides very good results and stable behavior with minimal effort. From an implementation point of view, the only difference between the explicit and semi-implicit method is just the order of performing computations. The velocity v_1 is computed the same way as in the explicit case. However, the position of the rigid body x_1 is not computed from the previous velocity v_0, but from the currently computed velocity v_1:

$$\begin{aligned} v_1 &= \frac{p_1}{m} \\ x_1 &= x_0 + v_1\Delta t \end{aligned} \tag{3.18}$$

The orientation q_1 is computed accordingly:

$$\begin{aligned} \omega_1 &= I_1^{-1}L_1 \\ q_1 &= q_0 + \frac{1}{2}\omega_1 q_0\Delta t \end{aligned} \tag{3.19}$$

Comparison of explicit and semi-implicit integration

An experiment was conducted to illustrate the difference between the explicit and semi-implicit method. A testing scene was prepared and an unconstrained motion in the virtual environment (i.e. without exerting force) of the haptic device was recorded using the Virtual Phantom tool. The recorded motion was then used for playback and tested for two haptic rendering setups. The first setup applied explicit integration of Newtons-Euler equations and the second setup applied semi-implicit integration. Force response magnitudes were recorded for both setups. The data was then split into subsets of size 20 and for each subset the average value *avg* (blue line) and a shifted variance value *var* (green line) was computed using the following equations:

$$\begin{aligned} avg &= \frac{1}{20}\sum_{i=1}^{20}|F_i| \\ var &= avg + \sum_{i=1}^{20}(|F_i| - avg)^2 \end{aligned} \tag{3.20}$$

The variance corresponds to noise in the force magnitude. If the variance is high, the haptic device produces unpleasant vibrations. The graph of *avg* and *var* values for the recording is shown in Figure 3.16.

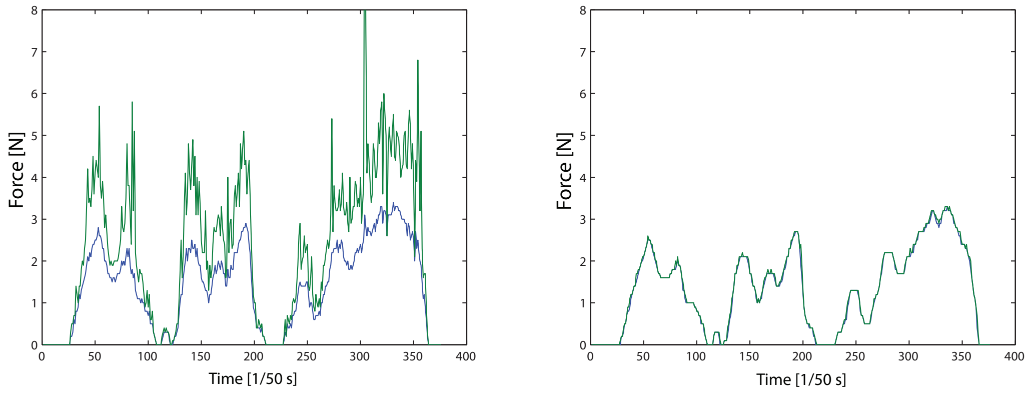


Figure 3.16: Explicit integration force magnitudes (left) and semi-implicit integration force magnitudes (right)

Average values for both rendering setups are similar, however it can be observed that the haptic rendering setup with semi-implicit integration (graph on the right) produces much better results and that the explicit integration generates a lot of artificial noise.

3.6 Virtual coupling

The concept of virtual coupling as a stabilization component was described in the previous chapter. The previous section described how forces acting on the dynamic object change its translational and rotational motion. This section will clarify how exactly virtual coupling is integrated into the haptic rendering algorithm and how it behaves.

Virtual tool as a dynamic object

In a virtual coupling scheme, the haptic tool is represented by a dynamic object. The object interacts with the (in this case static) virtual environment by penetrating into the proximity layer and the repulsive force is applied. The goal of the repulsive force is to prevent interpenetration of the haptic tool with the static environment.

The dynamic object is also connected to the haptic device with a spring-damper system. This connection controls the movement of the dynamic object. The advantage of using the dynamic object as a haptic tool proxy is that the magnitude of the repulsive force can be much larger than the one used for direct force rendering. A consequence of the large repulsive force is that the proxy is constrained by the proximity layer and remains outside the surface layer, while the actual position (i.e. the haptic device end-effector position) of the haptic tool may be inside a static object. This emulates the concept of analytically computed constrained proxy. Unfortunately, the stability of the discete dynamic simulation at a low haptic thread frequency is a limiting factor of the repulsive force stiffness value so it does not fully address the pop-through effect.

Integration scheme of Virtual coupling

To summarize information above, there are two types of forces acting on the dynamic object representing the virtual tool:

- F_c, τ_c - the contact force and torque resulting from interaction with the virtual environemt
- F_v, τ_v - the virtual coupling spring-damper system connecting the haptic device with the dynamic object

Firstly, position p_d and orientation q_d of the dynamic object is computed from applying these forces using the rigid body dynamics. Next, position p_h and orientation q_h of the haptic device end-effector is obtained and the new spring-damper force F_v and torque τ_v can be calculated and sent again to the dynamic simulation. Finally, as a consequence of Newton's third law, the opposite of spring-damper force $F_h = -F_v$ and torque $\tau_h = -\tau_v$ is sent as force feedback to the haptic device. The scheme is shown in Figure 3.17.

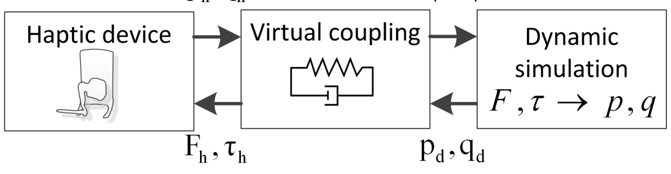


Figure 3.17: Virtual coupling integration scheme

Spring-damper system

As mentioned above, the haptic device and the dynamic object is coupled with a system composed of a linear spring, torsional spring, linear damper and torsional damper.

The **translational part** of the system (i.e. linear spring and linear damper) is parametrized by the following constants:

- k_t - translational stiffness of the virtual coupling
- b_t - translational damping coefficient of the virtual coupling

And by the following variables:

- $d_v = p_d - p_h$ - distance between the position of the haptic device end-effector and the dynamic object
- $v_r = \frac{d}{dt} d_v$ - relative velocity of the dynamic object

The **rotational part** of the system (i.e. torsional spring and torsional damper) is parametrized by the following constants:

- k_r - rotational stiffness of the virtual coupling
- b_r - rotational damping coefficient of the virtual coupling

And by the following variables:

- $\theta = \text{axisAngle}(q_d * q_h^{-1})$ - axis-angle representation of angular displacement where $*$ is the Grassman product operator and *axisAngle* is a function converting quaternion to axis-angle representation
- $\omega_r = \frac{d}{dt}\theta$ - relative angular velocity of the dynamic object

The whole spring-damper system is therefore described by the following equations:

$$\begin{aligned} F_v &= k_t d_v - b_t v_r \\ \tau_v &= k_r \theta - b_r \omega_r \end{aligned} \tag{3.21}$$

Critical damping

The spring system part of virtual coupling produces oscillations that are mitigated by the damping part. The whole system can be either under-damped which means that the system will still oscillate but at a gradually diminishing frequency, over-damped which would result in high viscous resistance, or critically damped which results in a non-oscillating system with the lowest possible viscous resistance, which is the desired behavior.

The translational part of the spring-damper system is critically damped if the damping ratio $\zeta_t = 1$:

$$\zeta_t = \frac{b_t}{2\sqrt{mk_t}} = 1 \quad \Rightarrow \quad b_t = 2\sqrt{mk_t} \tag{3.22}$$

where m is constant mass of the dynamic object. The rotational part is critically damped accordingly.

Sufficient condition for passivity

We can use the derivation of the critical damping to compute maximal translational stiffness constant k_r. If we analyze virtual coupling spring-damper system in the same way as a virtual wall system described in the previous chapter, we may compute a stiffness constant that comply with the passivity criterion for a virtual wall:

$$\mathrm{b} > \frac{\mathrm{KT}}{2} + |\mathrm{B}| \quad \Rightarrow \quad \mathrm{b} > \frac{\mathrm{KT}}{2} + 2\sqrt{mK} \tag{3.23}$$

An example of derivation of stiffnes constant K for following constants

- $T = 0.001\, s$ - a simulation period corresponding to frequency 1000 Hz
- $b = 8\, Ns/m$ - inherent mechanical damping of a haptic device
- $m = 0.025\, kg$ - a mass of the dynamic object

Solving the equation above will give:

$$8 > \frac{0.001\,\mathrm{K}}{2} + 2\sqrt{0.025\,\mathrm{K}} \quad \Rightarrow \quad \mathrm{K} \in <0;\, 594> N/m \tag{3.24}$$

The maximal translation stiffness of the virtual coupling is $594\, N/m$.

3.7 Pseudo-haptics

Pseudo-haptic systems can be defined as "systems providing haptic information generated, augmented or modified, by the influence of another sensory modality" [24]. In other words, pseudo-haptics creates an illusion of haptic interaction using, for example, visual feedback. The following example will illustrate the use of pseudo-haptics in a common situation without the need for any special device.

Mouse cursor viscous effect example

Let's say we want to simulate a viscous effect using a computer mouse and its cursor. The computer mouse itself cannot exert any force and the only thing that can be controlled is the position of the cursor. We split the screen where the cursor is moving into two areas. The first area represents a normal freespace behavior without any alteration of the position of the cursor. The second area will display a viscous effect by decreasing the relative cursor position change proportionally to its speed, i.e. when the mouse moves slowly, the viscous effect is small and when the mouse moves quicky, the viscous effect is large. Even though the mouse does not exert any forces, users moving in the viscous area intuitively react as if they feel the viscous effect, i.e. they push harder against the mouse, thus actually experiencing force. The visual feedback acts here as a pseudo-haptic system emulating viscosity.

Section overview

In this thesis, torque feedback is emulated by applying a pseudo-haptic system which utilizes force feedback and visual feedback. This section will describe one particular solution that is subsequently tested in a user study presented in chapter 4. First of all, behavior of a 6-DOF haptic rendering algorithm tested on a 6/3-DOF haptic device is analyzed and its problems are stated. Secondly, a solution to the stated problems is proposed. Next, three types of torque feedback emulation are introduced. Finally, the integration to the haptic rendering algorithm is described.

3.7.1 6/3-DOF interaction

The problem of missing torque feedback in a 6-DOF haptic rendering algorithm can be illustrated on the problem of missing force feedback in a 3-DOF haptic rendering algorithm.

Zero force response

Imagine that the haptic rendering algorithm operates normally but the generated force response is not sent to the haptic device. Tha haptic tool can move in the virtual environment without any constraints, penetrate into an object, move freely within the object and get to the other side. This behavior was previously defined as a pop-through effect that resulted either from incorrectly computed force response, instabilities or discrete collision detection. The pop-through effect makes haptic rendering nonrealistic and is undesirable.

Missing torque feedback

A similar problem, though more complex, applies for the case of missing torque feedback. The orientation of the haptic tool can be changed by rotating the end-effector of the haptic device. However, no force generated in the virtual environment will ever change the orientation of the end-effector. Consequences of this behaviour can be generalized into two types of interaction.

The first type of interaction (called *good torque* interaction) happens when the haptic tool is rotating in some direction and partially penetrates an object. The penalty-based haptic rendering algorithm computes the repulsive force at the location of penetration. The torque corresponding to the repulsive force would normally cause the haptic tool to rotate in direction opposite to the original rotation. However, as no torque can be applied, the haptic tool only changes its position as a reaction to the repulsive force. This kind of interaction has often no side effects causing instability of the system or pop-through effects. The response feels realistic depending on the distance of the penetration location from the center of mass of the haptic tool. The interaction is illustrated in Figure 3.18.

The second type of generalized interaction (called *bad torque* interaction) can be described as a lever-type or prying interaction. This happens when the haptic tool is rotating about the center of mass and encounters an obstacle at two opposite sides of the tool in a given direction. The tool partially penetrates objects at both locations and repulsive forces are generated. The sum of these forces is

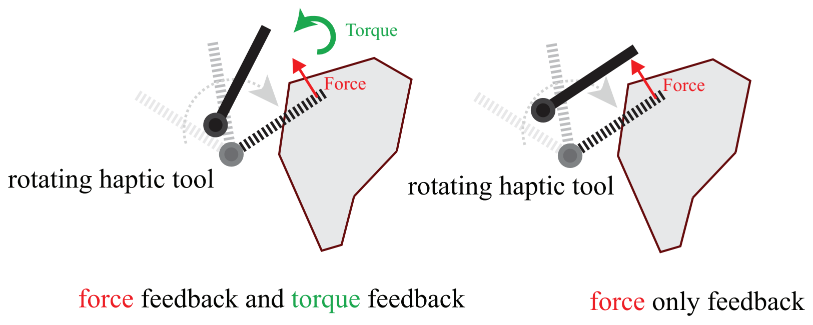


Figure 3.18: Response to the interaction with and without applied torque

zero because of their opposite directions. However, the torque corresponding to these forces is two times larger in magnitude. The torque would normally prevent the tool from penetrating the object, but again as no torque can be applied and this time even no force is generated, the haptic tool penetrates the object as illustrated in Figure 3.19.

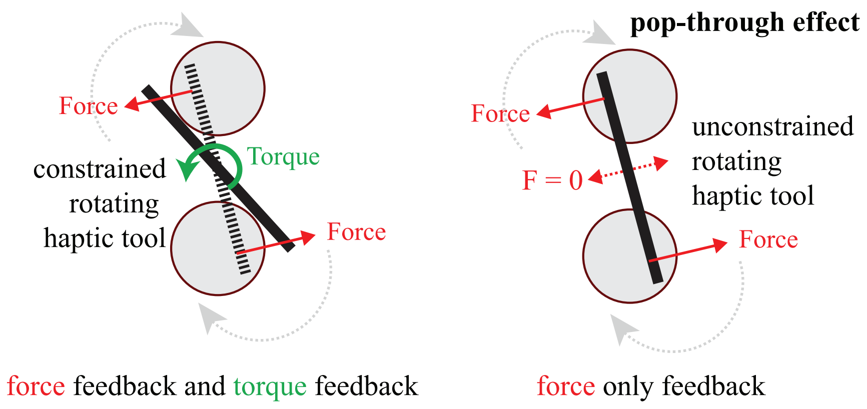

pop-through effect
Force
constrained rotating haptic tool
Torque
Force
Force
F = 0
unconstrained rotating haptic tool
Force
force feedback and torque feedback
force only feedback

Figure 3.19: Pop-through problem of missing torque feedback

When the haptic tool penetrated an object in the 3-DOF case of missing force feedback mentioned above, the repulsive force magnitudes were enormously large causing instability of the simulation system. But as the force was not sent to the haptic device, the user could not feel it. However, for the case of missing torque, the problem of large torue magnitude may result in unpredictable behavior.

3.7.2 Simulation-based solution to pop-through effect

The pop-through effect caused by missing torque feedback makes penalty-based 6-DOF haptic rendering algorithms unsuitable for 6/3-DOF haptic devices. A desirable effect would be to ignore the rotational movement of the haptic device end-effector and prevent the haptic tool from penetrating the object. Such an effect will make use of pseudo-haptics in that the visual feedback will inform the user that the haptic tool is constrained and that it cannot rotate anymore.

Lowering the stiffness of the torsional spring

The first idea to create the constraining effect was to utilize the virtual coupling. In the virtual coupling scheme, the dynamic object representing the haptic tool is coupled with the haptic device end-effector by a torsional spring as described in the previous section. If the rotational stiffness of the torsional spring k_r is set to a reasonably small value, the *torque generated from repulsive forces* will be stronger than the torque generated by the torsional coupling spring and the haptic tool will be constrained as shown in Figure 3.20 on the left.

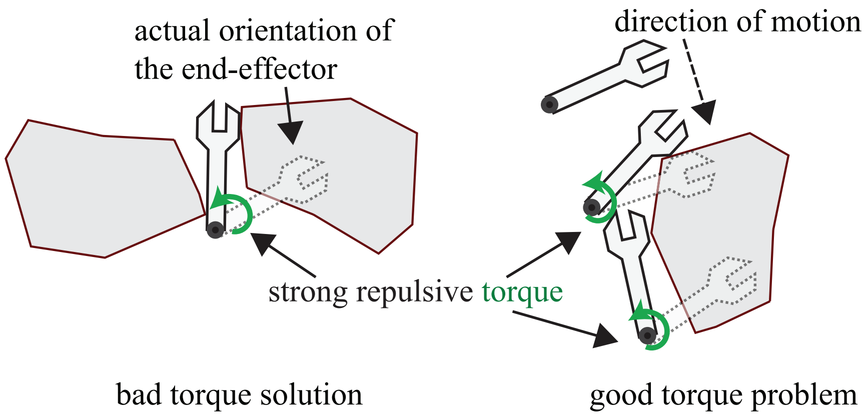


Figure 3.20: A low torsional spring stiffness

The low stiffness of the coupling solves the problem of the pop-through effect caused by the lever-type (*a bad torque*) interaction, but the problem of this approach is that the low stiffness of the coupling breaks the first type (*good torque*) interaction. When the torsional spring stiffness is low, the haptic tool tends to rotate around the object instead of generating a repulsive force as illustrated in Figure 3.20 on the right. An ideal solution would let the *good torque* work and limit only the *bad torque*.

Torque/force correlation

The problem is how to differentiate between the *good torque* and the *bad torque*. A characteristic property of the *bad torque* is the small magnitude of the total force acting on the haptic tool and the large magnitude of the torque. This can be used for a heuristic that would compare the applied torque and force and limit coupling torque stiffness only if the *force magnitude is not proportional to the torque magnitude*.

If we look at the problem from the *good torque* point of view, the *good torque* magnitude is actually proportional to the force magnitude which is exactly the opposite of the *bad torque* characteristics.

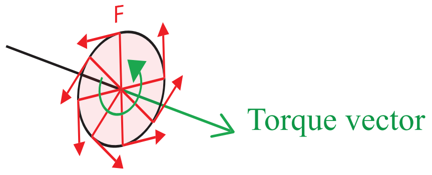


Figure 3.21: Torque vector illustration

The heuristic can be improved by a simple observation that all forces generating the *bad torque* are perpendicular to the torque vector as illustrated in

Figure 3.21. Thus, the heuristic compares only the magnitude of the sum of all forces that are perpendicular to the torque vector. Also, torque magnitude filtering by moving average is applied to mitigate discontinuities in the torque response caused by inaccurate dynamic simulation.

Vibration feedback

The visual emulation of the torque feedback by constraining the haptic tool in a *bad torque* interaction may be improved by applying another form of pseudo-haptic feedback. The visual feedback does not provide information about the magnitude of the torque that can not be haptically rendered. Therefore, the pseudo-haptic part of the 6/3-DOF haptic rendering algorithm is extended by a vibration effect that corresponds to the mentioned magnitude of the torque. This vibration effect largerly improves the overall cognitive intepretation of the missing torque feedback.

3.7.3 Integration of pseudo-haptics

To summarize this, there are actually three pseudo-haptic systems emulating the torque.

- The first system is the *good torque* interaction which utilizes the force feedback and is part of the collision response.
- The second system is visual feedback constraining the haptic tool from interpenetration and pop-through effect using the information from collision response and virtual coupling.
- The third system is a vibration force that uses the information from the second system. The vibration force is not added to the virtual coupling force as this would lead to instabilities; instead, it is directly rendered to the haptic device.

The final pipeline of the 6/3-DOF haptic rendering algorithm is shown in Figure 3.22.

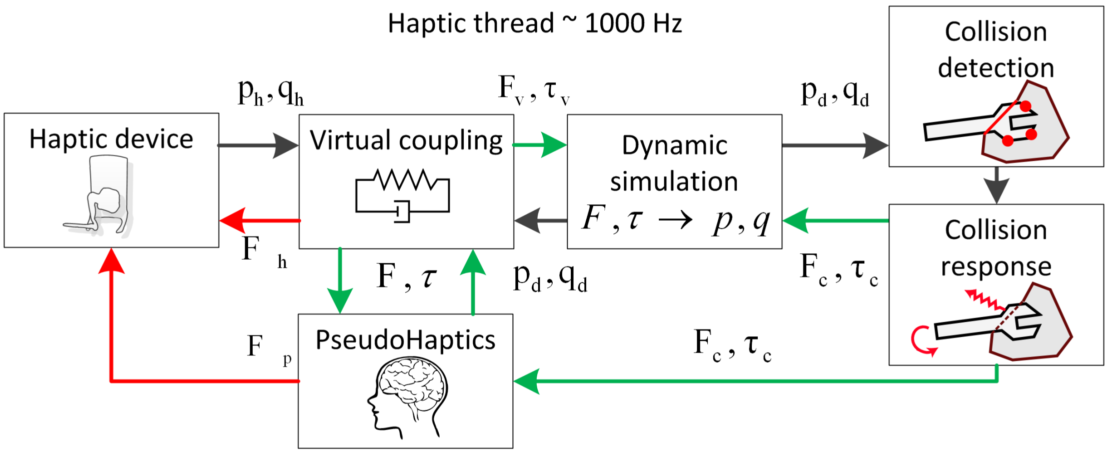


Figure 3.22: Pipeline of the 6/3-DOF haptic rendering algorithm

3.8 Virtual Phantom

Virtual Phantom si a tool specifically developed as part of this thesis to debug haptic rendering algorithms. The idea of a virtual device originates from 3-DOF CHAI 3D *Virtual Haptic Device* (also called dhdVirtual). The virtual device makes it possible to test haptic applications developed using CHAI 3D without having a real haptic device. *Virtual Haptic Device* is a separate application that communicates with CHAI 3D and acts like a real haptic device that does not exert forces. The application is started automatically when no real devices are available.

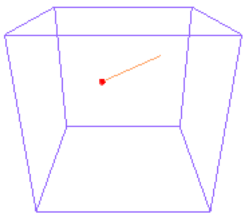


Figure 3.23: 3-DOF CHAI 3D Virtual Haptic Device workspace

It provides a graphical representation of workspace, the 3-DOF haptic tool (a small sphere) and generated force as a vector as shown in Figure 3.23. The applicaton allows the user to move the virtual haptic tool using computer mouse and keyboard. Virtual Phantom extends the concept to 6 degrees of freedom and adds several functionalities.

Translational and rotational motion

The 6-DOF Virtual Phantom haptic tool is represented by a drill object that is a part of CHAI 3D SDK shown in Figure 3.24. A visual aid represented by a line connecting the tool and the center of the workspace is also presented to help locate the haptic tool.

Figure 3.24: 6-DOF Virtual Phantom haptic tool

The 6-DOF haptic tool can be translated and rotated using computer mouse and the position and rotation data is then sent to CHAI 3D library at a frequency of 1000 Hz.

The position and orientation of the tool is changed by relative distances travelled when dragging over the workspace, i.e. the user holds down the mouse button and moves the cursor. Holding the left mouse button causes the translation in YZ-plane, right mouse button causes the translation in XY-plane. The rotation about Z-axis and Y-axis of the tool is changed by holding down the middle mouse button.

Another way of changing the position and orientation of the virtual haptic tool is by attaching a real haptic device. The position and orientation of the Virtual Phantom then corresponds to the position and orientation of the attached real haptic device.

Force and Torque visualization

The force and torque is visualized in two ways. Firstly, a red line visualizing the force sent to the device is placed at the tip of the tool. The length of the line is proportional to the force magnitude. Analogously, a green line visualizing the torque sent to the device is also placed at the tip of the tool as shown in Figure 3.25.

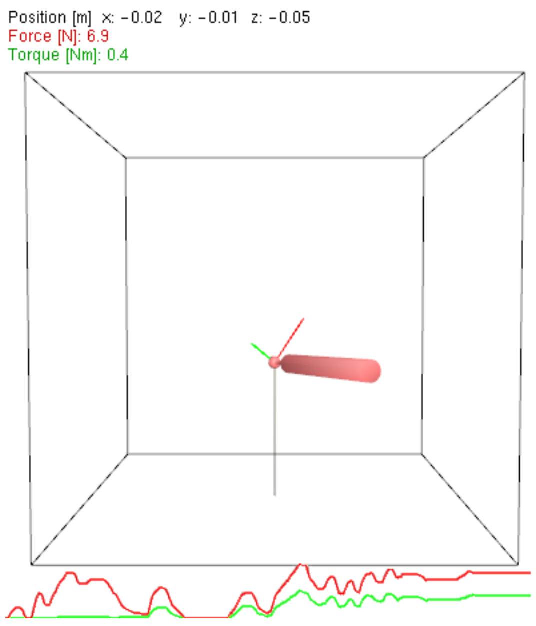


Figure 3.25: 6-DOF Virtual Phantom workspace

The second type of force and torque visualization is a floating graph representing magnitudes of force and torque. The graph contains average value of the magnitude and the shifted variance as described in the comparison of explicit and semi-implicit integration (i.e. equation 3.20).

Haptic recording and playback

Motion of the haptic tool can be recorded along with the forces sent to the device. The data is saved to a file containing the time, position and force. The recording can be later played back which makes it possible to compare different setups of the haptic rendering algorithm as shown in the comparison of explicit and semi-implicit integration (Figure 3.16).

The recording can be also used for evaluation and verification of diffent haptic rendering algorithms providing the same input data. Need for ways of haptic rendering evaluation is generally recognized and several other approaches have been proposed, such as: physical scanning of real-world objects [25] or an empirical approach for the evaluation of haptic rendering algorithms presented by [26].

Pseudo-haptic force feedback simulation

A trivial pseudo-haptic force feedback simulation was implemented to improve debugging possibilities. When a force is sent to the Virtual Phantom device, the haptic tool is moved proportianally to the magnitude and in the direction of the force, thus trivially simulating dynamics of the resistance of a hand holding the haptic device end-effector. This behaviour helps with a lot of situations when the interaction of the virtual haptic tool is actually needed to debug the haptic rendering setup.

4. User study

The main goal of the user study is to evaluate performance and realism of the developed 6/3-DOF haptic rendering algorithm. The user study consists of 4 applications simulating different scenarios of haptic interaction, with an emphasis on missing torque feedback. Different combinations of pseudo-haptic feedback were tested. All testing applications provided a possibility to turn on and off the vibration pseudo-haptic feedback. It was also possible to show or hide a wireframe model of the actual position and orientation of the haptic device end-effector. More details about user controls can be found in Appendix B.

A total of 6 users participated in the study. Most of the users did not have any previous experience with computer haptics.

Testing environment

Applications were tested on the PHANToM Desktop haptic device with 6-DOF positional sensing and 3-DOF force feedback connected via parallel port (IEEE 1284). The computer running user study testing applications was equipped with a Quad-Core Intel Core-i7 2.4 GHz CPU and 8 GB RAM. This configuration was sufficient to produce realistic force feedback at a minimal haptic thread frequency of 1000 Hz.

Questions asked during the user study

The following questions were asked during the user study for all testing applications:

1. Does the force feedback correspond with objects in the virtual environment which is displayed on the monitor?
2. How would you describe the environment interaction from the haptic point of view?
3. Does the force feedback feel realistic?
4. Does the vibration pseudo-haptic feedback help you interpret the missing torque?
5. Do you prefer to see the actual position and orientation of the haptic device end-effector using a wireframe model or not?
6. In which type of applications would you use the 6/3-DOF haptic rendering if you could choose whether to buy an expensive full 6-DOF haptic device or a considerably cheaper 6/3-DOF haptic device?

4.1 Testing applications

4.1.1 Rabbit and hammer

The first testing application is called Rabbit and hammer. The goal of this application was to present participants with the behavior of force feedback generated by the 6/3-DOF haptic rendering algorithm and exerted by the haptic device. The presentation of the scene is shown in Figure 4.1.

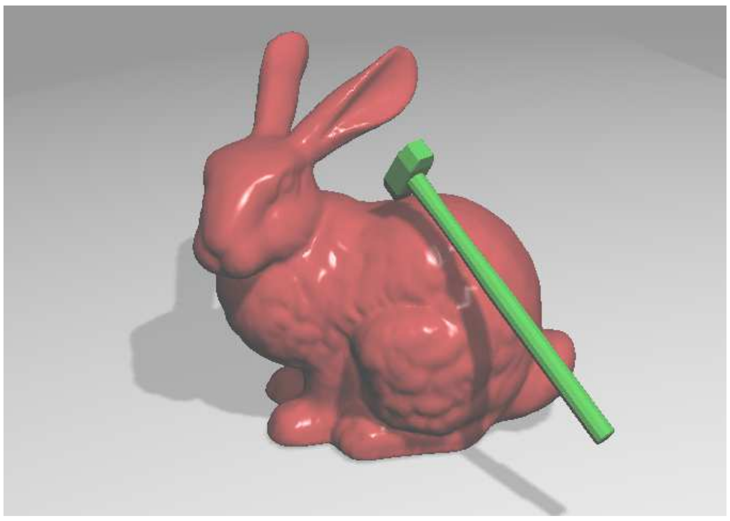


Figure 4.1: Rabbit and hammer

In the first phase, the participant was asked to explore the environment and get used to the interaction phenomenon. Next, the participant was asked to utilize the proportions of the haptic tool and interact with parts of the tool distant from the center of mass.

4.1.2 Pipe and torus

The second testing application is called Pipe and torus. The goal of this application was to present a different type of the haptic tool and increase requirements on end-effector manipulation skills. The presentation of the scene is shown in Figure 4.2.

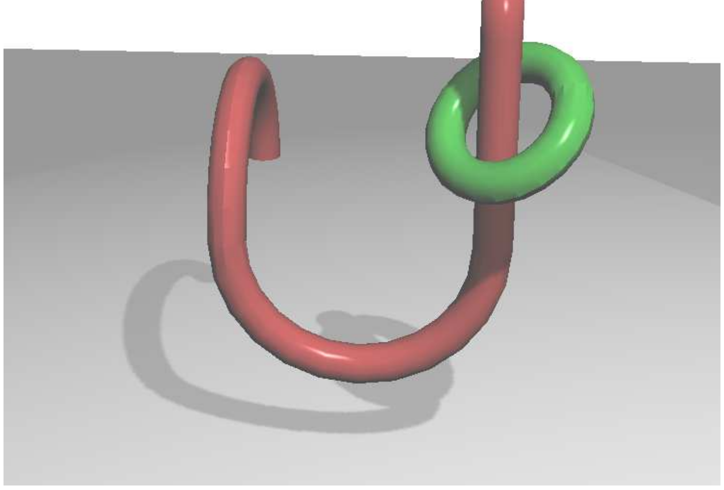


Figure 4.2: Pipe and torus

Firstly, the participant was asked to move the torus along the pipe from one side to the other. This scenario demanded a greater manipulation skills. Secondly, the participant was asked to try prying the pipe with the torus and describe the sensation.

4.1.3 Pipe and tube

The third testing application is called Pipe and tube. The goal of this application was to evaluate a task that normally involves high torque feedback. The presentation of the scene is shown in Figure 4.3. The wireframe model of the actual position of the haptic device end-effector is displayed and marked in the figure.

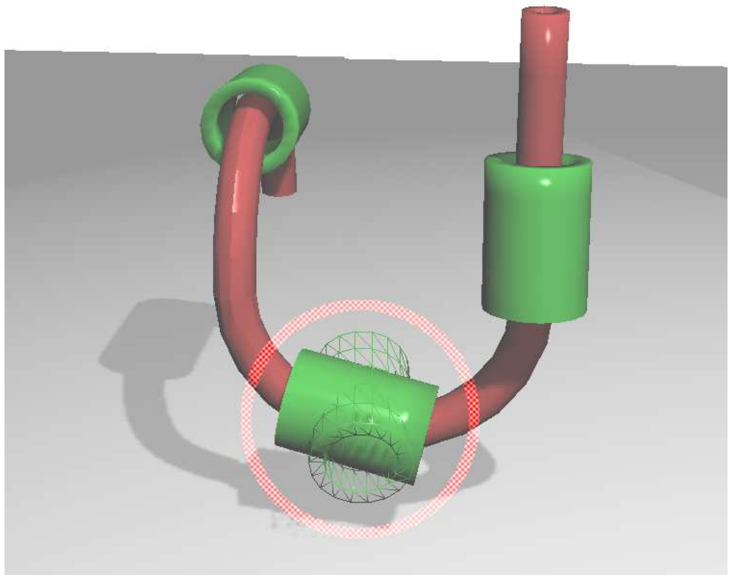


Figure 4.3: Pipe and tube

The participant was asked to move the tube along the pipe from one side to the other as in the previous test. However, the tube is highly constrained by the pipe. Thus, the system generates high torque feedback when the haptic tool is not finely manipulated.

4.1.4 Piping and tube tool

The last testing application is called Piping and tube tool. The goal of this application was to evaluate spatial orientation when the the haptic tool was not fully visible. The presentation of the scene is shown in Figure 4.4.

Firstly, the participant was asked to move the tube tool through the piping without colliding. Next, the participant was asked to move the tube tool with their eyes closed and describe whether the force feedback alone guides them.

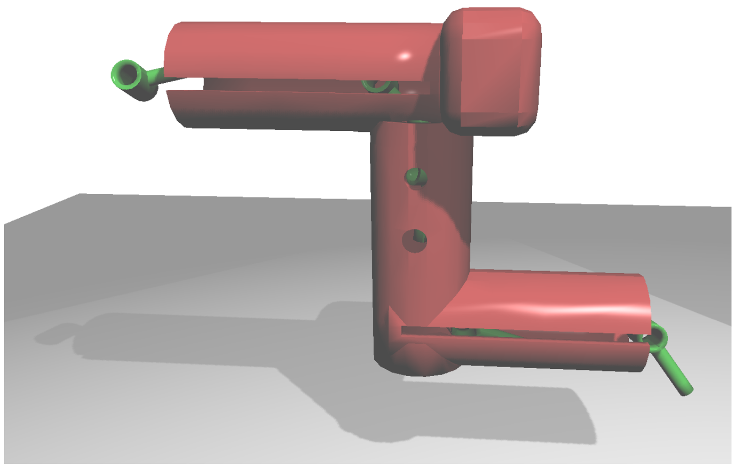


Figure 4.4: Piping and tube tool

4.2 Results

The results of the user study are summarized in the following list of answers to questions 1-6 above:

1. *Does the force feedback correspond with objects in the virtual environment which is displayed on the monitor?*

   All participants have confirmed that in every testing application the force feedback corresponds to the virtual environment which is displayed on the monitor. This shows that the collision detection part of the haptic rendering algorithm developed in the thesis works correctly and that the collision response generates appropriate force feedback.

2. *How would you describe the environment interaction from the haptic point of view?*

   One participant described the material of the rabbit in the first testing application as metallic. However, after a longer exploration, the participant agreed with other participants that the material of the rabbit feels like slipping rubber. This behavior is a consequence of zero surface friction and the lower stiffness of the virtual coupling which, on the other hand, preserves stability.

3. *Does the force feedback feel realistic?*

   All participants agreed that the force feedback feels realistic. No participant complained about the force feedback even when interacting in parts of the tool distant from the center of mass. This proves that the problem of missing torque feedback was well addressed by the pseudo-haptic system used in the force feedback.

4. *Does the vibration pseudo-haptic feedback help you interpret the missing torque?*

The first reaction to the vibration effect was rather unpleasant. However, every participant intuitively stopped rotating the haptic tool and rotated the tool back into the orientation that does not produce vibrations. Also, the magnitude of vibration force helped participants with finding a non-vibrating orientation.

After a longer exploration, participants got used to the concept of vibrating force and agreed that the vibration effect largely improves the information about the missing torque feedback and that they have subconsciously connected the non-realistic vibrating force with the torque.

5. *Do you prefer to see the actual position and orientation of the haptic device end-effector using a wireframe model or not?*

   In most situations, the wireframe model was distracting and interfered with the visual pseudo-haptic feedback. However, in the third testing application (Pipe and tube), participants often lost sense of orientation because of pipe constraints as shown in Figure 4.3. Therofore in this situation, the wireframe model helped with completion of the task.

6. *In which type of applications would you use the 6/3-DOF haptic rendering if you could choose whether to buy an expensive 6/6-DOF haptic device or a considerably cheaper 6/3-DOF haptic device?*

   All participants agreed that a 6/6-DOF haptic device is needed for highly realistic haptic feedback such as high accuracy virtual assembly involving wrenching. However, the 6/3-DOF haptic rendering algorithm developed in this thesis makes a competitive alternative for simpler tasks such as touching and exploring objects with tools. An example of such usage may be haptic exploration of scanned archaeological objects in museums or exhibitions. Most participants stated that the 6/3-DOF haptic rendering algorithm highly improves overall impression and realism of haptic feedback over single-point 3-DOF interaction.

5. Conclusion

5.1 Summary

The thesis has introduced haptic modality of the human-computer interaction and provided basic information about sources of haptic stimuli.

The definition of haptic rendering for impedance type devices was stated. Various haptic rendering algorithms were analyzed beginning with the 1-DOF haptic rendering where the problem of stability was introduced and a passivity criterion of virtual coupling scheme was presented as a solution to instabilities.

Direct rendering methods for 3-DOF haptic rendering were described and their problems illustrated. The God-object method was then analyzed and described in detail together with geometric definition of proxy-based haptic rendering. A solution to rendering of low-quality digitization polygonal meshes was also presented.

Performance issues of 6-DOF haptic rendering were stated and several methods with different drawbacks were analyzed. A simulation based multi-rate method that decoupled part of the haptic rendering pipeline from the haptic rendering thread was described and illustrated.

A problem of haptic rendering for underactuated devices was introduced and pseudo-haptics solution to the problem proposed. Trivial implementations of 6/3-DOF haptic rendering and their outcomes were described and a final approach for 6/3-DOF haptic rendering based on the voxel sampling method was chosen.

The developed 6/3-DOF haptic rendering algorithm was then analyzed in detail for every part of the algorithm including: collision detection of the PointShell representation and the voxel tree data structure, collision response from a tangent plane force model, rigid body dynamics and the virtual coupling.

Psuedo-haptics systems were then described and a heuristic for determining lever-type interaction was analyzed and implemented. Different types of pseudo-haptic feedback were combined and tested in the user study.

The user study showed that proposed pseudo-haptic systems greatly improves the overall impression for underactuated devices and that the 6/3-DOF haptic rendering algorithm developed as a part of the thesis can be used in several applications that do not demand highly realistic torque feedback.

5.2 Future work

The developed haptic rendering algorithm can be improved in many ways. First, a stability of the dynamic object simulation can be improved by using a multi-rate approach. Problem of the instable behavior can be observed when holding the end-effector far from the rotating tip. However, Colgate et al. [8] stated that even if a non-passive virtual environment may be stable, interaction with a human via a haptic interface may cause instability.

Secondly, haptic rendering of a surface material would also improve the performance of pseudo-haptic systems. The lever-type interaction could cause high friction which would constrain the haptic tool from moving away freely as it is possible now.

Next, implementation of pseudo-haptic systems to constraint-based haptic rendering algorithm would improve the stability and maybe bring new possibilities of pseudo-haptic feedback.

Virtual Phantom device can be improved in many ways: the simple simulation of pseudo-haptic feedback can be extended to a simulation of a moving hand, haptic recording and playback performance can be improved, mouse control can be also enhanced.

Bibliography

[1] Thorsten A. Kern, editor. *Engineering Haptic Devices: A Beginner's Guide for Engineers.* Springer, 2009 edition, 7 2009.

[2] Ming C. Lin and Miguel Otaduy, editors. *Haptic Rendering: Foundations, Algorithms and Applications.* A K Peters/CRC Press, 7 2008.

[3] Bern University of Applied Sciences. Haptic computer automated virtual environment. url: *http://www.cpvrlab.ti.bfh.ch/research/projects/cave*, 2012.

[4] SensAble Technologies Inc. Phantom desktop. url: *http://www.sensable.com/haptic-phantom-desktop.htm*, 2013.

[5] Kadleček Petr. A Practical Survey of Haptic APIs, Bachelor's thesis, Charles University in Prague, Czech Republic, 2010.

[6] J Edward Colgate and J Michael Brown. Factors affecting the z-width of a haptic display. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3205–3210. IEEE, 1994.

[7] DW Weir and JE Colgate. Stability of haptic displays.

[8] J Edward Colgate, Michael C Stanley, and J Michael 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*, volume 3, pages 140–145. IEEE, 1995.

[9] Richard J Adams and Blake 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*, volume 2, pages 1254–1259. IEEE, 1998.

[10] C. B. Zilles and J. K. Salisbury. A constraint-based god-object method for haptic display. IEEE Computer Society, 1995.

[11] Ryo Kikuuwe and Hideo Fujimoto. Incorporating geometric algorithms in impedance-and admittance-type haptic rendering. In *EuroHaptics Conference, 2007 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics 2007. Second Joint*, pages 249–254. IEEE, 2007.

[12] Pultr Aleš. Skripta z matematické analýzy pro I. ročník, 1994.

[13] Diego C. Ruspini, Krasimir Kolarov, and Oussama Khatib. The haptic display of complex graphical environments. SIGGRAPH '97, pages 345–352, 1997.

[14] M Sagardia, T Hulin, C Preusche, and G Hirzinger. Improvements of the voxmap-pointshell algorithm-fast generation of haptic data-structures. In *53rd IWK-Internationales Wissenschaftliches Kolloquium, Ilmenau, Germany*, 2008.

[15] Arthur Gregory, Ajith Mascarenhas, Stephen Ehmann, Ming Lin, and Dinesh Manocha. Six degree-of-freedom haptic display of polygonal models. In *Proceedings of the conference on Visualization'00*, pages 139–146. IEEE Computer Society Press, 2000.

[16] Miguel A Otaduy and Ming C Lin. 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*, pages 247–256. IEEE, 2005.

[17] Michael Ortega, Stephane Redon, and Sabine Coquillart. A six degree-of-freedom god-object method for haptic display of rigid bodies. In *Virtual Reality Conference, 2006*, pages 191–198. IEEE, 2006.

[18] William A McNeely, Kevin D Puterbaugh, and James J Troy. Six degree-of-freedom haptic rendering using voxel sampling. In *Computer graphics proceedings, annual conference series*, pages 401–408. Association for Computing Machinery SIGGRAPH, 1999.

[19] Federico Barbagli and Kenneth Salisbury. The effect of sensor/actuator asymmetries in haptic interfaces. In *Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2003. HAPTICS 2003. Proceedings. 11th Symposium on*, pages 140–147. IEEE, 2003.

[20] Lawton N Verner and Allison M Okamura. Force & torque feedback vs force only feedback. In *EuroHaptics conference, 2009 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics 2009. Third Joint*, pages 406–410. IEEE, 2009.

[21] Andreas Pusch and Anatole Lécuyer. Pseudo-haptics: from the theoretical foundations to practical system design guidelines. In *Proceedings of the 13th international conference on multimodal interfaces*, pages 57–64. ACM, 2011.

[22] Christer Ericson. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology)*. CRC Press, 12 2004.

[23] David Baraff. Physically based modeling: Rigid body simulation. *SIGGRAPH Course Notes, ACM SIGGRAPH*, 2(1):2–1, 2001.

[24] Alexis Paljic, J-M Burkhardtt, and Sabine Coquillart. Evaluation of pseudo-haptic feedback for simulating torque: a comparison between isometric and elastic input devices. In *Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2004. HAPTICS'04. Proceedings. 12th International Symposium on*, pages 216–223. IEEE, 2004.

[25] Emanuele Ruffaldi, Dan Morris, Timothy Edmunds, Federico Barbagli, and Dinesh K Pai. Standardized evaluation of haptic rendering systems. In *Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2006 14th Symposium on*, pages 225–232. IEEE, 2006.

[26] Chris Raymaekers, Joan De Boeck, and Karin Coninx. An empirical approach for the evaluation of haptic algorithms. In *Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005. World Haptics 2005. First Joint*, pages 567–568. IEEE, 2005.

[27] Conti Francois et al. *CHAI 3D set of libraries*, 2009. `http://www.chai3d.org/`.

Attachments

A. Programmer's guide

This appendix provides information about implementation details of 6/3-DOF haptic rendering algorithm and the Virtual Phantom tool. Firstly, a structure and integration of 6/3-DOF haptic rendering algorithm in a CHAI 3D library will be described. Next section will explain the implementation of VoxMap collision detection algorithm. A class that connects all parts of the haptic rendering pipeline will be described in the third section. The fourth section will briefly describe why and how shadow rendering was implemented. Finally, a Virtual Phantom solution and project files will be described.

A.1 Integration into CHAI 3D

The 6/3-DOF haptic rendering algorithm is implemented as an extension to a CHAI 3D SDK [27]. CHAI 3D is a scene graph API written in the C++ programming language with aim to create a modular, open source and cross platform haptic API with a wide support of different haptic devices.

Modules and groups

The CHAI 3D library is split into several modules and class groups that provide specific tasks. The 6/3-DOF haptic rendering algorithm extends and utilizes mainly these groups: Devices, Graphics, Math, Haptic Tools, Haptic Effects, Force Rendering, Collision Detection, Timers.

Scene graph

The main unit of all objects in the scene graph is a *cGenericObject* class which inherits from a general abstract type *cGenericType*. The generic object creates a tree structure of objects using a standard template vector class of children objects in a *m_children* member.

A.1.1 Generic tool

The scene graph representation of a haptic device is called a tool. An abstract class defining all tools in the scene graph is *cGenericTool*. The generic tool contains a *m_device* member of type *cGenericHapticDevice* which serves as a wrapper for the communication with a haptic device driver.

There are three main methods called successively in a haptic loop running in the haptic thread:

1. *updatePose()* - updates the position and rotation of the tool using the data from the haptic device

2. *computeInteractionForces()* - compute interaction forces between the tool and the virtual environment

3. *applyForces()* - apply computed forces to the haptic device

The only specific tool that CHAI 3D officially provides at this time is a 3-DOF tool identified as a *cGeneric3dofPointer*. This thesis extends the CHAI 3D library by adding a 6/3-DOF tool identified as a *cGeneric63dofPointer* and described later in this chapter.

A class representing a force model is needed in order to compute forces in the tool. The *cGeneric3dofPointer* class uses a *cProxyPointForceAlgo* class which is an implementation of the God-object method described in section 2.2. This thesis extends the CHAI 3D library by adding *cGenericPointShellForceAlgo* class which represents all PointShell algorithms. The 6/3-DOF haptic rendering integration structure is shown in Figure A.1. More detailed information about CHAI 3D can be found in [5].

A.1.2 Application flow

A typical haptic application starts with loading the 3D data (e.g. mesh models, volume data) and initialization. The process then uses two separate threads: a haptic thread and a graphic thread. The graphic thread renders the virtual environment at approximate frequency of 60 Hz. The haptic thread updates at approximate frequency of 1000 Hz and runs the three main methods of the haptic tool class mentioned above. Figure A.2 illustrates a typical flow of the application using 6/3-DOF haptic rendering in CHAI 3D.

A.2 VoxMap

A class *cVoxMap* is a wrapper for the voxel tree defined by a root node member *m_rootNode*. It provides methods for adding a new mesh to the VoxMap, voxelizing the interior of the VoxMap, creating a proximity layer and rendering a voxel tree visualization.

A.2.1 cVoxMapTreeNode

A *cVoxMapTreeNode* class represents a voxel tree node. The class has two template arguments specifying its behavior.

The first template argument is *sVoxMapTreeNodeConfig* which is a structure defining:

- voxelSize - size of the voxel in metres
- nodeDim - a number of subdivisions of the space in one dimension
- maxLevel - maximal depth of the tree

The second template argument is *sVoxMapTreeNodeTraits* which specifies a behavior of voxel data manipulation. The programmer can use its own voxel data structure in the voxel tree provided that he defines a specialization of the *sVoxMapTreeNodeTraits* class.

cVoxMapTreeNode visitors

A *cVoxMapTreeNode* class accepts several visitor classes:

- cVoxelVisitor - visits voxel data in the current node
- cTraversalVisitor - traverses the tree depth-first
- cScanLineVisitor - visits voxels in a scan-line manner

Specified visitors are used for testing collision in the node, rendering the node (*cRenderVoxelVisitor*) or voxelizing an interior.

A.3 6/3-DOF Pointer

A *cGeneric63dofPointer* class defines the 6/3-DOF haptic tool. It contains methods for computing virtual coupling and rigid body dynamics. It also stores corresponding constants and variables needed for computing the simulation based force. The pseudo-haptic torque/force correlation system is applied by *applyPseudoHapticsCouplingTorque* method.

A.3.1 Point-shell force algorithms

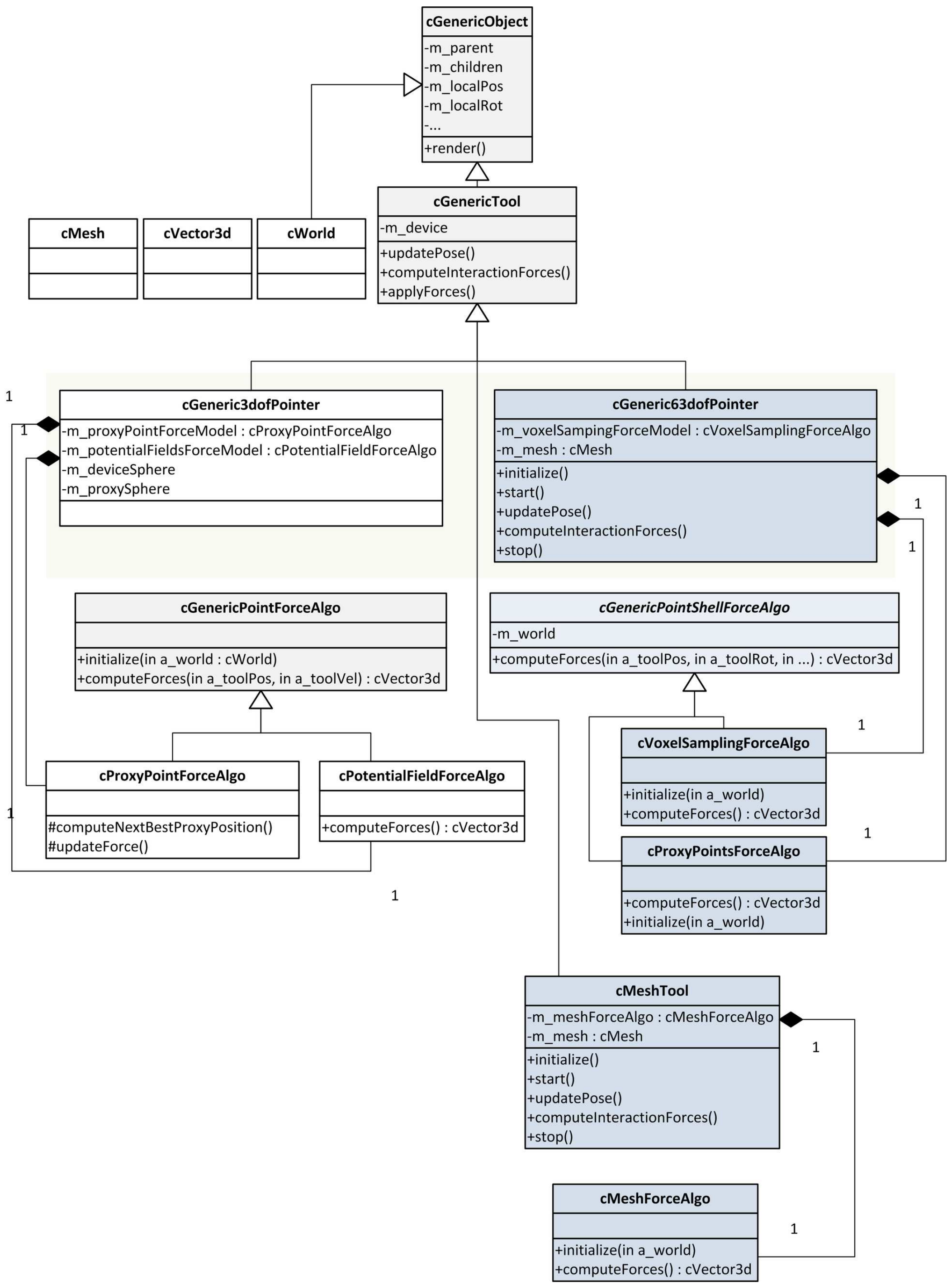
PointShell force algorithms are derived from a *cGenericPointShellForceAlgo* class parametrized by the template argument *tPointShellVoxMapTreeNodeConfig*. The *cGenericPointShellForceAlgo* class contains PointShell data, properties and method to compute the PointShell from the mesh using the VoxMap with the *tPointShellVoxMapTreeNodeConfig* configuration.

cVoxelSamplingForceAlgo

A *cVoxelSamplingForceAlgo* class implements voxel sampling tangent plane force model described in previous chapter. The class contains the VoxMap tree node definition and the VoxMap itself. It provides method to compute the tangent plane force and method to add the mesh to the VoxMap.

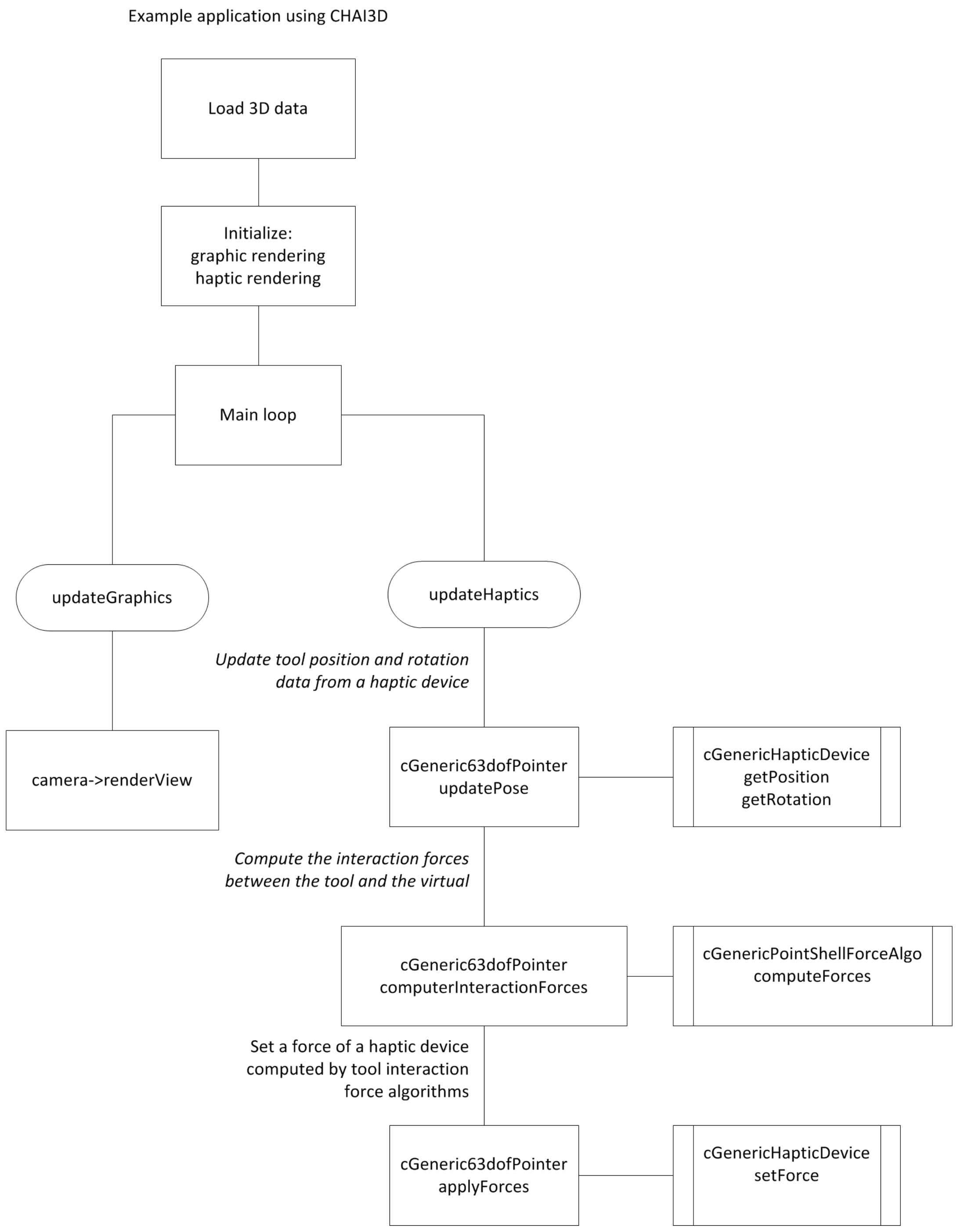
cProxyPointsForceAlgo

A *cProxyPointsForceAlgo* is an implementation of the trivial force rendering method described in previous chapter as *Proxy point set collision*. It contains method for computing the force response, updating the proxy position and setting the surface friction.



cGenericObject
-m_parent
-m_children
-m_localPos
-m_localRot
-...
+render()
cGenericTool
-m_device
+updatePose()
+computeInteractionForces()
+applyForces()
cMesh
cVector3d
cWorld
cGeneric3dofPointer
-m_proxyPointForceModel : cProxyPointForceAlgo
-m_potentialFieldsForceModel : cPotentialFieldForceAlgo
-m_deviceSphere
-m_proxySphere
cGeneric63dofPointer
-m_voxelSampingForceModel : cVoxelSamplingForceAlgo
-m_mesh : cMesh
+initialize()
+start()
+updatePose()
+computeInteractionForces()
+stop()
cGenericPointForceAlgo
+initialize(in a_world : cWorld)
+computeForces(in a_toolPos, in a_toolVel) : cVector3d
cGenericPointShellForceAlgo
-m_world
+computeForces(in a_toolPos, in a_toolRot, in ...) : cVector3d
cProxyPointForceAlgo
#computeNextBestProxyPosition()
#updateForce()
cPotentialFieldForceAlgo
+computeForces() : cVector3d
cVoxelSamplingForceAlgo
+initialize(in a_world)
+computeForces() : cVector3d
cProxyPointsForceAlgo
+computeForces() : cVector3d
+initialize(in a_world)
cMeshTool
-m_meshForceAlgo : cMeshForceAlgo
-m_mesh : cMesh
+initialize()
+start()
+updatePose()
+computeInteractionForces()
+stop()
cMeshForceAlgo
+initialize(in a_world)
+computeForces() : cVector3d
1
1
1
1
1
1
1
1
1
1
1

Figure A.1: 6/3-DOF haptic rendering integration structure in CHAI 3D



Example application using CHAI3D
Load 3D data
Initialize:
graphic rendering
haptic rendering
Main loop
updateGraphics
updateHaptics
camera->renderView
Update tool position and rotation data from a haptic device
cGeneric63dofPointer updatePose
cGenericHapticDevice getPosition getRotation
Compute the interaction forces between the tool and the virtual
cGeneric63dofPointer computerInteractionForces
cGenericPointShellForceAlgo computeForces
Set a force of a haptic device computed by tool interaction force algorithms
cGeneric63dofPointer applyForces
cGenericHapticDevice setForce

Figure A.2: A typical flow of the haptic application

A.4 Shadow map light

A *cShadowMapLight* class implements support of casting shadows which greatly improves spatial orientation as shown in Figure A.3. Shadows are rendered using a basic shadow mapping technique utilizing FrameBuffer object, Pixel and Vertex Shaders which are stored in a *cShadowMapLight* source file.

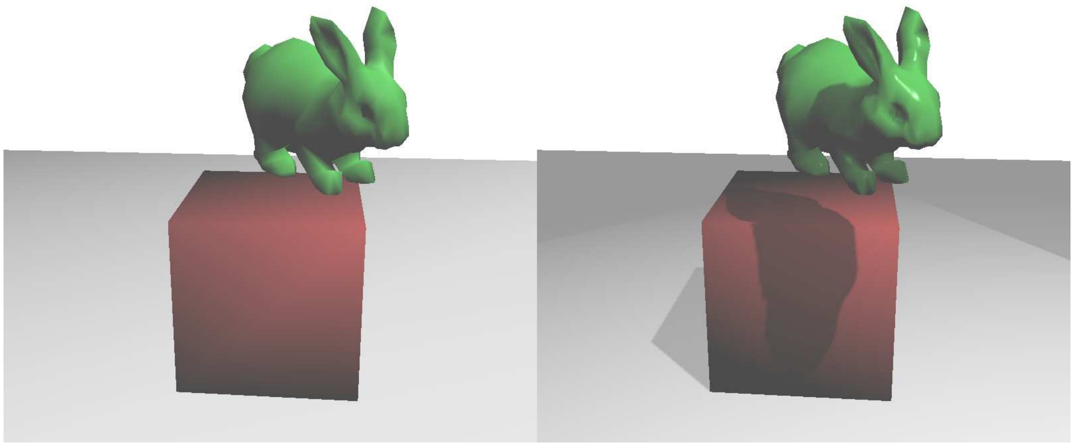


Figure A.3: Shadow

A.5 Virtual Phantom

Virtual Phantom application solution consists of two projects.

- The first project is a library called *hdPhantomVirtual*, which serves as a wrapper and communication library with CHAI 3D. The communication is made through *named file mapping object* (a memory-mapped system file).
- The second project is called *Virtual Phantom* which is a GUI application using QT library to show the workspace of the Virtual Tool and menu bar.

B. User Documentation

B.1 System requirements

Hardware requirements

- PHANToM Desktop haptic device
- Quad-Core Intel Core-i7 2.4 GHz CPU with 4 GB RAM or better

Software requirements

- PHANToM Desktop 64-bit drivers
- Microsoft Windows Vista 64-bit operating system or Microsoft Windows 7 64-bit operating system
- Microsoft Visual C++ 2010 Redistributable Package or Microsoft Visual Studio 2010

B.2 User study keyboard shortcuts

All user study testing applications have following keyboard shortcuts:

- S - turn on/off shadows
- D - show/hide wireframe model of the haptic device end-effector
- T - enable/disable vibrating pseudo-haptic effect

B.3 Running Virtual Phantom

Following steps are required in order to run the Virtual Phantom application correctly:

- Copy VirtualPhantom.exe to the bin/ directory of the CHAI 3D SDK
- Backup hdPhantom.dll file in the bin/ directory of the CHAI 3D SDK
- Copy hdPhantomVirtual.dll to the bin/ directory of the CHAI 3D SDK
- Rename hdPhantomVirtual.dll to hdPhantom.dll

C. Contents of the accompanying CD

The structure of accompanying CD is as follows:

- /bin - contains CHAI 3D SDK and compiled binaries of user study testing applications for Microsft Windows 7 64-bit operating system along with application data files
- /doc - contains the master thesis in PDF format, reference manual and programmer's guide
- /src - contains source code together with a Microsoft Visual Studio 2010 solution and project files