8
Introduction to Haptic Rendering Algorithms

M. A. Otaduy and M. C. Lin

The second part of this book focuses on the rendering algorithm of a haptic interface system. It presents several techniques commonly used for computing interactions between virtual objects and for displaying the contact forces to a user. The collection of chapters covers different aspects of the rendering algorithm, such as collision detection and contact force computation, with the specific challenges and solutions associated with different configuration spaces (such as 3D point, 3D object, etc.), material properties (rigid vs. solid), or model descriptions (polygonal surface, parametric surface, etc.).

In this chapter we first give an overview on the key elements in a typical rendering algorithm. We start by formulating a general definition of the haptic rendering problem. Then, we list different algorithmic components and describe two of the traditional approaches in detail: direct rendering vs. rendering through a virtual coupling. We continue with an exposition of basic concepts for modeling dynamics and contact, both with rigid and deformable bodies, and we conclude with an introduction to multirate rendering algorithms for enhancing the quality of haptic rendering.

8.1 Definition of the Rendering Problem

In the real world, we perceive contact forces when we touch objects in the environment. These forces depend on the surface and material properties of the objects, as well as the location, orientation, and velocity with which we touch them. Using an analogy with the real world, haptic rendering can be defined as the process of generating contact forces to create the illusion of touching virtual objects.

8.1.1 Kinesthetic Display of Tool Contact

Haptic perception can be divided into two main categories, based on the nature of the mechanoreceptors that are activated: *cutaneous* perception is related to mechanoreceptors in the skin, while *kinesthetic* perception is related to mechanoreceptors in joints, tendons, and muscles. And the type of contact forces that can be perceived can be classified into two types: forces that appear in direct contact between the skin and the environment, and forces that appear due to contact between an object manipulated by the user (i.e., the *tool*) and other objects in the environment.

In this part of the book, we focus mostly on the kinesthetic perception of contact forces through a tool, i.e., the perception of contact between a manipulated tool and other objects, on our joints, tendons, and muscles. As mentioned in Chapter 1, even when using an intermediate tool, subjects can infer medium- and large-scale properties of the objects in the environment as if touching them directly. Moreover, the use of a tool becomes a convenient computational model in the design of haptic rendering algorithms, and even the computation of direct skin interaction could make use of a tool model for representing e.g., fingers (as described in Chapter 4).

8.1.2 Teleoperation of a Virtual Tool

Figure 8.1 shows an example of haptic rendering of the interaction between two virtual jaws. The user manipulates a haptic device, and can perceive through the rendering algorithm the contact between the jaws as if he or she were actually holding and moving the upper jaw. In this example, the upper jaw can be regarded as a virtual tool, and the lower jaw constitutes the rest of the virtual environment.

Haptic rendering of the interaction between a virtual tool and a virtual environment consists of two tasks:

1. Compute and display the forces resulting from contact between the virtual tool and the virtual environment.

2. Compute the configuration of the virtual tool.

A haptic rendering system is composed of two sub-systems: one real system (i.e., the user and the haptic device), and one virtual system (i.e., the tool and the environment). The tool acts as a virtual counterpart of the haptic device. From this perspective, haptic rendering presents a remarkable similarity to master-slave teleoperation, a topic well studied in the field of robotics. The main difference is that in teleoperation both the master and the slave are real physical systems, while in haptic rendering the tool is a virtual system. Similarly, haptic rendering shares some of the challenges of teleoperation, namely, the computation of *transparent*

Figure 8.1. Manipulation of a virtual jaw. The user manipulates a haptic device, and though the rendering algorithm can perceive forces as if manipulating the upper jaw in the virtual environment.

teleoperation: i.e., the virtual tool should follow the configuration of the device, and the device should produce forces that match those computed on the tool, without filtering or instability artifacts.

8.1.3 A Possible Definition

It becomes clear from the discussion above that a haptic rendering problem should address two major computational issues, i.e., finding the tool configuration and computing contact forces, and it should use information about device configuration. Moreover, it requires knowledge about the environment, and in some cases it modifies the environment as well. With these aspects in mind, one possible general definition of the rendering problem could be as follows:

Given a configuration of the haptic device $\mathcal{H}$, find a configuration of the tool $\mathcal{T}$ that minimizes an objective function $f(\mathcal{H} - \mathcal{T})$, subject to environment constraints. Display to the user a force $\mathbf{F}(\mathcal{H}, \mathcal{T})$ dependent on the configurations of the device and the tool.

This definition assumes a causality precedence where the input variable is the configuration of the haptic device $\mathcal{H}$, and the output variable is the force $\mathbf{F}$. This precedence is known as *impedance rendering*, because the haptic rendering algorithm can be regarded as a programmable mechanical impedance [Hogan 85, Adams and Hannaford 98], as depicted in Figure 8.2.

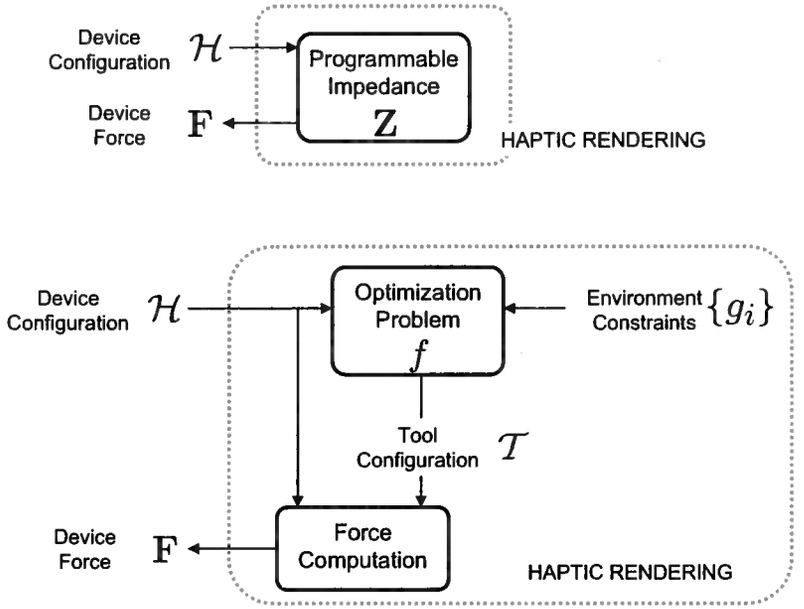


Figure 8.2. Overview of haptic rendering. The top block diagram shows haptic rendering as an impedance control problem, with the device configuration as input, and the device forces as output. A more detailed look in the bottom breaks the haptic rendering problem into two subproblems: (1) computation of the tool configuration, and (2) computation of device forces.

In impedance rendering, the device control system should provide position information and implement a force control loop.

A different possibility is *admittance rendering*, where the haptic rendering system can be regarded as a programmable mechanical admittance that computes the desired device configuration, as the result of input device forces. In that case, the device control should provide device forces and implement a position control loop. As discussed by [Adams and Hannaford 98], the two types of rendering systems present dual properties, and their design can be addressed in a unified manner; therefore, here we restrict the exposition to impedance rendering.

In the definition of the haptic rendering problem presented above, we have not specified the objective function that must be optimized for computing the tool's configuration, nor the function for computing output forces. The differences among various haptic rendering algorithms lie precisely in the design of these functions and are briefly outlined next and covered in the following chapters.

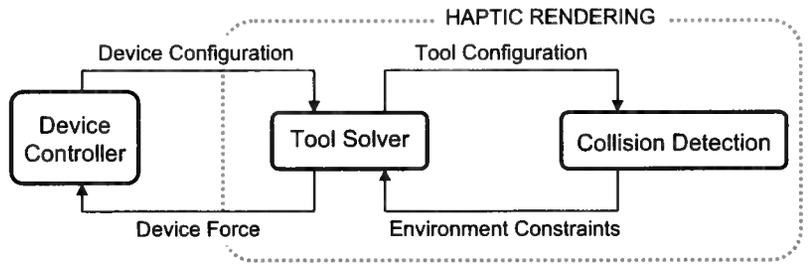


Figure 8.3. Main components of a general impedance-type rendering algorithm.

8.2 Components of a Rendering Algorithm

On a high level, for an impedance-rendering system, the rendering algorithm must be composed of (i) a solver module that determines the configuration of the tool, and (ii) a collision detection module that defines environment constraints on the tool. Figure 8.3 depicts such a high-level description of the algorithm. More specifically, we distinguish the following components of the rendering algorithm, which may vary among specific implementations.

8.2.1 The Tool Model

The number of degrees of freedom (DOFs) of the tool is a variable that to a large extent depends on the application, and should be minimized as much as possible to reduce the complexity of the rendering. Hence, in many modeling applications (see Chapter 26), it suffices to describe the tool as a point with three DOFs (translation in 3D), in medical applications (see Chapter 24), it is often possible to describe it as a ray segment with five DOFs, and in virtual prototyping (see Chapter 22), it is often required to describe the tool as a solid with six DOFs (translation and rotation in 3D). When the tool is represented as a point, the haptic rendering problem is known as *three-degree-of-freedom* (3-DOF) haptic rendering, and when the tool is represented as a rigid body, it is known as *six-degree-of-freedom* (6-DOF) haptic rendering.

In case of modeling the tool and/or the environment with solid objects, the haptic rendering algorithm also depends on the material properties of these objects. Rigid solids (see Section 8.4.1 and Chapter 16) are limited to six DOFs, while deformable solids (see Section 8.4.2 and Chapters 19 and 20) may present a large number of DOFs. At present, the dynamic simulation of a rigid tool is efficiently handled in commodity processors, and the challenge may lie on the complexity of the environment and the

contact configuration between tool and environment. Efficient dynamic simulation of complex deformable objects at haptic rates is, however, an issue that deserves further research.

8.2.2 The Optimization Problem

The simplest possible option for the objective function is the distance between tool and haptic device, i.e., $f = \|\mathcal{H} - \mathcal{T}\|$. Given this objective function, one can incorporate information about the environment in multiple ways. In the most simplest way, known as *direct rendering* and described in more detail in the next section, the environment is not accounted for in the optimization leading to the solution $\mathcal{T} = \mathcal{H}$. Another possibility is to introduce hard inequality constraints $g_i(\mathcal{T}) \geq 0$ that model non-penetration of the tool in the environment. The highly popular *god-object* 3-DOF haptic rendering algorithm [Zilles and Salisbury 95] formulates such an optimization problem. Non-penetration may also be modeled following the penalty method, with soft constraints that are added to the objective function: $f = \|\mathcal{H} - \mathcal{T}\| + \sum_i k_i g_i^2(\mathcal{T})$.

As in general optimization problems, there are multiple ways of solving the tool's configuration. The optimization may be solved until convergence on every rendering frame, but it is also possible to perform a finite number of solver iterations (e.g., simply one), leading to a quasi-static solution for every frame.

Moreover, one could add inertial behavior to the tool, leading to a problem of dynamic simulation. Then, the problem of solving the configuration of the tool can be formulated as a dynamic simulation of a virtual physically based replica of a real object. This approach has been, in fact, followed by several authors (e.g., [McNeely et al. 99, Otaduy and Lin 06]). Modeling the virtual tool as a dynamic object has shown advantages for the analysis of stability of the complete rendering algorithm, as described in Chapter 7. If the dynamic simulation can be regarded as a solution to an optimization problem, where forces represent the gradient of the objective function, then the selected type of numerical integration method can be viewed as a different method for solving the optimization problem. For example, a simple explicit Euler corresponds to gradient descent, while implicit Euler corresponds to a Newton solver.

8.2.3 Collision Detection

As noted in Figure 8.3, in the context of haptic rendering, collision detection is the process that, given a configuration of the tool, detects potentially violated environment constraints. Collision detection can easily become the computational bottleneck of a haptic rendering system with geometrically complex objects, and its cost often depends on the configuration space

of the contacts. Therefore, we devote much attention to the detection of collisions with a point tool for 3-DOF rendering (see Chapter 10) and the detection of collisions with a solid tool for 6-DOF rendering (see Chapters 9, 11, 12, and 13).

There are also many variations of collision detection, depending on the geometric representation of the objects. In this book, we focus mostly on objects using surface representations, with most of the chapters discussing approaches for polygonal representations, and one chapter on techniques for handling parametric surfaces (see Chapter 17). Similarly, we separately handle collision detection for objects with high-resolution texture information in Chapter 18.

8.2.4 Collision Response

In algorithms where the tool's configuration is computed through a dynamic simulation, *collision response* takes the environment constraints given by the collision detection module as input and computes forces acting on the tool. Therefore, collision response is tightly related to the formulation of environment constraints $g_i(\mathcal{T})$ discussed above.

The two commonly used approaches for collision response are penalty methods and Lagrange multipliers, which we introduce later in Section 8.4.3. Other chapters in this book describe collision response in more detail. Chapter 11 presents a penalty-based approach for collision response on a rigid body in the context of 6-DOF rendering, while Chapters 16 and 20 describe constraint-based approaches on rigid and deformable bodies, respectively. In case of dynamic environments, collision response must be computed on the environment objects as well.

8.3 Direct Rendering vs. Virtual Coupling

We now focus on two specific approaches to the rendering algorithm, to illustrate with examples the different components of the algorithm.

8.3.1 Direct Rendering Algorithm

The overall architecture of direct rendering methods is shown in Figure 8.4. Direct rendering relies on an impedance-type control strategy. First, the configuration of the haptic device is received from the controller, and it is assigned directly to the virtual tool. Collision detection is then performed between the virtual tool and the environment. Collision response is typically computed as a function of object separation or penetration depth (see Section 9.4) using penalty-based methods. Finally, the resulting contact force (and possibly torque) is directly fed back to the device controller.

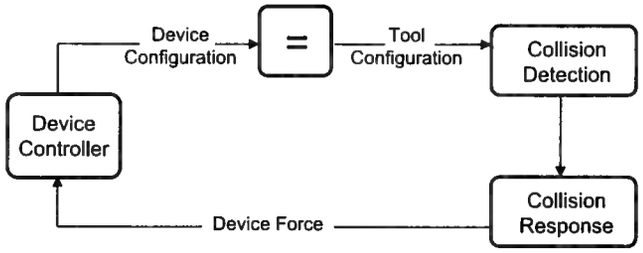


Figure 8.4. Main components of a direct rendering algorithm.

More formally, and following the discussion from Section 8.2.2, direct rendering corresponds to an optimization problem that trivially assigns $\mathcal{T} = \mathcal{H}$ (up to some scale or rigid transformation). This solution answers the second problem in haptic rendering, i.e., the computation of the configuration of the tool. Then, the first problem, the computation of forces, is formulated as a function of the location of the tool in the virtual environment, $\mathbf{F}(g, \mathcal{T})$.

The popularity of direct rendering stems obviously from the simplicity of the calculation of the tool's configuration, as there is no need to formulate a complex optimization problem (for example, rigid body dynamics in 6-DOF rendering). However, the use of penalty methods for force computation has its drawbacks, as penetration values may be quite large and visually perceptible, and system instability can arise if the force update rate drops below the range of stable values (see the discussion in Chapter 7).

Throughout the years, direct rendering architectures have often been used as a first practical approach to haptic rendering. Thus, the first 3-DOF haptic rendering algorithms computed forces based on potential fields defined inside the environment. However, as pointed out early by [Zilles and Salisbury 95], this approach may lead to force discontinuities and pop-through problems. Direct rendering algorithms are perhaps more popular for 6-DOF rendering, and a number of authors have used them, in combination with convex decomposition and hierarchical collision detection [Gregory et al. 00b, Ehmann and Lin 01] (see Section 9.3 for more details); with parametric surface representations [Nelson et al. 99] (see Chapter 17); with collision detection hierarchies based on normal cones [Johnson and Cohen 01, Johnson and Willemsen 03, Johnson and Willemsen 04, Johnson et al. 05] (see also Section 17.7); or together with fast penetration depth computation algorithms and contact clustering [Kim et al. 02c, Kim et al. 03] (see Section 9.4). In most of these approaches, the emphasis was on fast collision detection or proximity queries, and the work can also be combined with the virtual coupling algorithms described next.

8.3.2 Rendering through Virtual Coupling

Despite the apparent simplicity of direct rendering, the computation of contact and display forces may become a complex task from the stability point-of-view. As discussed in more detail in Chapter 7, stability of haptic rendering can be answered by studying the range of programmable impedances. With direct rendering and penalty-based contact forces, rendering impedance is hardly controllable, leading often to unstable haptic display.

As also described in Chapter 7, stability enforcement can largely be simplified by separating the device and tool configurations, and inserting in-between a viscoelastic link referred to as *virtual coupling* [Colgate et al. 95]. The connection of passive subsystems through virtual coupling leads to an overall stable system. Figure 8.5 depicts the configurations of the device and the tool in a 6-DOF virtual contact scenario using virtual coupling. Contact force and torque are transmitted to the user as a function of the translational and rotational misalignment between tool and device configurations.

Figure 8.6 depicts the general structure of a rendering algorithm based on virtual coupling (for an impedance-type display). The input to the rendering algorithm is the device configuration, but the tool configuration is solved in general through an optimization problem, which also accounts for environment constraints. The difference between device and tool configu-

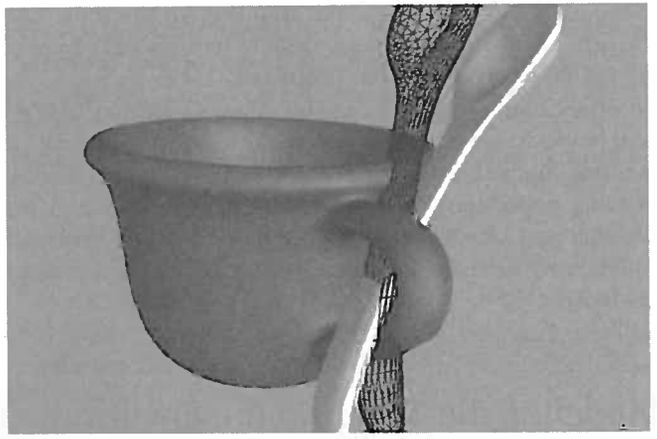


Figure 8.5. Manipulation through virtual coupling. As the spoon is constrained inside the handle of the cup, the contact force and torque are transmitted through a virtual coupling. A wireframe image of the spoon represents the actual configuration of the haptic device [Otaduy and Lin 06]. (© 2006 IEEE)

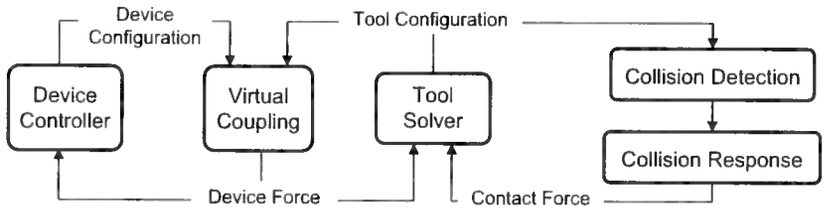


Figure 8.6. Main components of a rendering algorithm using virtual coupling.

ration is used both for the optimization problem and for computing output device forces.

The most common form of virtual coupling is a viscoelastic spring-damper link. Such a virtual coupling was used by [Zilles and Salisbury 95, Ruspini et al. 97] in the god-object and virtual proxy algorithms for 3-DOF rendering. The concept was later extended to 6-DOF rendering [McNeely et al. 99], by considering translational and rotational springs. For simplicity, here we also group under the name of virtual coupling other approaches that separate tool and device configurations, such as the four-channel architecture based on teleoperation control designed by [Sirouspour et al. 00], or constraint-aware projections of virtual coupling for 6-DOF rendering [Ortega et al. 06].

The use of a virtual coupling allows a separate design of the impedance displayed to the user (subject to stability criteria), from the impedance (i.e., stiffness) of environment constraints acting on the tool. Environment constraints can be of high stiffness, which reduces (or even completely eliminates) visible interpenetration problems.

On the other hand, virtual coupling algorithms may suffer from noticeable and undesirable filtering effects, in case the update rate of the haptic rendering algorithm becomes too low, which highly limits the value of the rendering impedance. Multirate algorithms [Adachi et al. 95, Mark et al. 96, Otaduy and Lin 06] (discussed in more detail in Section 8.5 in this chapter) can largely increase the transparency of the rendering by allowing stiffer impedances.

8.4 Modeling the Tool and the Environment

In this section we pay special attention to the optimization problem for computing the tool configuration, by briefly introducing some examples: a 6-DOF tool solved with rigid body dynamic simulation, deformable objects, and formulation of contact constraints.

8.4.1 Rigid Body Dynamics

We first consider a tool modeled as a rigid body in 3D, which yields 6 DOFs: 3D translation and rotation. One possibility for solving the configuration of the tool is to consider a dynamic model where the tool is influenced by the environment through contact constraint forces, and by the user through virtual coupling force and torque.

We define the generalized coordinates of the tool as $\mathbf{q}$, composed of the position of the center of mass $\mathbf{x}$ and a quaternion describing the orientation θ. We define the velocity vector $\mathbf{v}$ by the velocity of the center of mass $\mathbf{v_x}$ and the angular velocity ω expressed in the world frame. We denote by $\mathbf{F}$ the generalized forces acting on the tool (i.e., force $\mathbf{F_x}$ and torque $\mathbf{T}$, including gravity, centripetal and Coriolis torque, the action of the virtual coupling, and contact forces).

Given the mass m and the inertia tensor $\mathbf{M}$ of the tool, the dynamics are defined by the Newton-Euler equations:

$$\begin{aligned} m\dot{\mathbf{v}}_\mathbf{x} &= \mathbf{F}_\mathbf{x}, \\ \mathbf{M}\dot{\omega} &= \mathbf{T} + (\mathbf{M}\omega) \times \omega. \end{aligned} \tag{8.1}$$

and the relationship between the generalized coordinates and the velocity vector is

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{v}_\mathbf{x}, \\ \dot{\theta} &= \mathbf{G}\omega. \end{aligned} \tag{8.2}$$

The matrix $\mathbf{G}$ relates the derivative of the quaternion to the angular velocity, and its definition is out of the scope of this book, but may be found in e.g., [Shabana 89]. The same reference will serve for an introduction to rigid body dynamics and the derivation of the Newton-Euler equations.

For compactness, it is useful to write the equations of motion in general form:

$$\begin{aligned} \mathbf{M}\dot{\mathbf{v}} &= \mathbf{F}, \\ \dot{\mathbf{q}} &= \mathbf{G}\mathbf{v}. \end{aligned} \tag{8.3}$$

Time discretization with implicit integration. Here, we consider time discretization schemes that yield a linear update of velocities and positions of the form

$$\begin{aligned} \tilde{\mathbf{M}}\mathbf{v}(i+1) &= \Delta t\tilde{\mathbf{F}}, \\ \mathbf{q}(i+1) &= \Delta t\tilde{\mathbf{G}}\mathbf{v}(i+1) + \mathbf{q}(i). \end{aligned} \tag{8.4}$$

Note that the updated coordinates $\mathbf{q}(i+1)$ need to be projected afterwards onto the space of valid rotations (i.e., unit quaternions).

One example of linear update is obtained by using Backward Euler discretization with first-order approximation of derivatives. As pointed out by [Colgate et al. 95], implicit integration of the differential equations describing the virtual environment can ease the design of a stable haptic display. This observation has lead to the design of 6-DOF haptic rendering algorithms with implicit integration of the rigid body dynamics of the tool [Otaduy and Lin 05, Otaduy and Gross 07]. Implicit integration also enhances display transparency by enabling stable simulation of the tool with small mass values.

Linearized Backward Euler discretization takes a general ODE of the form $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, t)$ and applies a discretization $\mathbf{y}(i+1) = \mathbf{y}(i) + \Delta t \mathbf{f}(\mathbf{y}(i+1), t(i+1))$, with a linear approximation of the derivatives $\mathbf{f}(\mathbf{y}(i+1), t(i+1)) \approx \mathbf{f}(\mathbf{y}(i), t) + \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(\mathbf{y}(i+1) - \mathbf{y}(i)) + \frac{\partial \mathbf{f}}{\partial t}\Delta t$. For time-independent derivatives, which is our case, this yields an update rule: $\mathbf{y}(i+1) = \mathbf{y}(i) + \Delta t(\mathbf{I} - \Delta t \frac{\partial \mathbf{f}}{\partial \mathbf{y}})^{-1}\mathbf{f}(i)$.

Applying the linearized Backward Euler discretization to (8.3), and discarding the derivative of the inertia tensor, the terms of the update rule given in Equation (8.4) correspond to

$$\tilde{\mathbf{M}} = \mathbf{M} - \Delta t \frac{\partial \mathbf{F}}{\partial \mathbf{v}} - \Delta t^2 \frac{\partial \mathbf{F}}{\partial \mathbf{q}} bfG,$$
$$\tilde{\mathbf{F}} = \mathbf{F}(i) + \left(\frac{1}{\Delta t}\mathbf{M} - \frac{\partial \mathbf{F}}{\partial \mathbf{v}}\right)\mathbf{v}(i). \tag{8.5}$$

As can be inferred from the equations, implementation of implicit integration requires the formulation of Jacobians of force equations $\frac{\partial \mathbf{F}}{\partial \mathbf{q}}$ and $\frac{\partial \mathbf{F}}{\partial \mathbf{v}}$. These Jacobians include the term for inertial forces $(\mathbf{M}\omega) \times \omega$, contact forces (see Section 8.4.3), or virtual coupling (see next). [Otaduy and Lin 06, Otaduy 04] formulate in detail these Jacobians.

Six-DOF virtual coupling. We pay special attention here to modeling viscoelastic virtual coupling for 6-DOF haptic rendering. The tool $\mathcal{T}$ will undergo a force and torque that move it toward the configuration $\mathcal{H}$ of the haptic device, expressed in coordinates of the virtual environment. We assume that the tool undergoes no force when the device's configuration in the reference system of the tool corresponds to a position $\mathbf{x}_c$ and orientation θ_c. We refer to this configuration as *coupling configuration*. Coupling is often engaged at the center of mass of the tool (i.e., $\mathbf{x}_c = 0$), but this is not necessarily true. Coupling at the center of mass has the advantage that coupling force and torque are fully decoupled from each other.

Given configurations $(\mathbf{x}_\mathcal{T}, \theta_\mathcal{T})$ and $(\mathbf{x}_\mathcal{H}, \theta_\mathcal{H})$ for the tool and the device, linear coupling stiffness k_x and damping b_x, the coupling force $\mathbf{F}$ on the tool can be defined as

$$\mathbf{F} = k_x(\mathbf{x}_\mathcal{H} - \mathbf{x}_\mathcal{T} - \mathbf{R}_\mathcal{T}\mathbf{x}_c) + b_x(\mathbf{v}_\mathcal{H} - \mathbf{v}_\mathcal{T} - \omega_\mathcal{T} \times (\mathbf{R}_\mathcal{T}\mathbf{x}_c)). \tag{8.6}$$

The definition of the coupling torque requires the use of an equivalent axis of rotation $\mathbf{u}$ [McNeely et al. 99]. This axis of rotation can be defined using the scalar and vector parts ($\Delta\theta_s$ and $\Delta\theta_u$ respectively) of the quaternion $\Delta\theta = \theta_\mathcal{H} \cdot \theta_c^{-1} \cdot \theta_\mathcal{T}^{-1}$ describing the relative coupling orientation between tool and device. Then,

$$\mathbf{u} = 2 \cdot \mathrm{acos}(\Delta\theta_s) \cdot \left(\frac{1}{\sin(\mathrm{acos}(\Delta\theta_s))} \cdot \Delta\theta_u\right). \tag{8.7}$$

The coupling torque can then be defined using rotational stiffness k_θ and damping b_θ as

$$\mathbf{T} = (\mathbf{R}\mathbf{x}_c) \times \mathbf{F} + k_\theta \mathbf{u} + b_\theta(\omega_\mathcal{H} - \omega_\mathcal{T}). \tag{8.8}$$

Multibody simulation. Dynamic simulation of the rigid tool itself is not a computationally expensive problem, but the problem becomes considerably more complex if the tool interacts with multiple rigid bodies. In fact, the fast and robust computation of multibody contact is still a subject of research, paying special attention to stacking or friction [Stewart and Trinkle 00, Mirtich 00, Milenkovic and Schmidl 01, Guendelman et al. 03, Kaufman et al. 05].

Moreover, with multibody contact and implicit integration it is not possible to write a decoupled update rule, given in Equation (8.4) for each body. The coupling between bodies may appear (a) in the Jacobians of contact forces when using penalty methods, or (b) through additional constraint equations when using Lagrange multipliers (see Section 8.4.3).

8.4.2 Dynamics of Deformable Objects

There are multiple options for modeling deformable objects, and researchers in haptic rendering have often opted for approximate approaches that trade accuracy for computational cost. Here we do not aim at covering in depth the modeling of deformable objects, but rather highlight through an example the inclusion in the complete rendering algorithm. Chapters 19 and 20 discuss in more detail practical examples of deformable object modeling for haptic rendering, focusing respectively on fast approximate models and the handling of contact.

The variational formulation of continuum elasticity equations leads to elastic forces defined as the negative gradient of elastic energy $\int_\Omega \sigma \cdot \epsilon \, \mathrm{d}\Omega$, where σ and ϵ represent stress and strain tensors. The various elasticity models differ in the definition of elastic strain or the definition of the

relationship between stress and strain. Given the definition of elastic energy, one can reach a discrete set of equations following the finite element method [Zienkiewicz and Taylor 89]. Typically, the dynamic motion equations of a deformable body may be written as

$$\mathbf{M} \cdot \dot{\mathbf{v}} = \mathbf{F} - \mathbf{K}(\mathbf{q}) \cdot (\mathbf{q} - \mathbf{q}_0) - \mathbf{D} \cdot \mathbf{v}, \tag{8.9}$$

$$\dot{\mathbf{q}} = \mathbf{v}. \tag{8.10}$$

where $\mathbf{M}$, $\mathbf{D}$, and $\mathbf{K}$ represent, respectively, mass, damping, and stiffness matrices. The stiffness matrix captures elastic forces and is, in general, dependent on the current configuration $\mathbf{q}$.

At present times, haptic rendering calls for fast methods for modeling elasticity, and a reasonable approach is to use the linear Cauchy strain tensor, as well as the linear Hookean relationship between stress and strain. Linear strain leads, however, to considerable artifacts under large deformations, which can be eliminated by using corotational methods that measure deformations in the unrotated setting of each mesh element [Müller et al. 02, Müller and Gross 04].

The use of corotational strain allows for stable and robust implicit integration methods, while producing a linear update rule for each time step. With linearized Backward Euler (described before for rigid bodies), the update rule becomes

$$\tilde{\mathbf{M}}\mathbf{v}(i+1) = \Delta t \tilde{\mathbf{F}}, \tag{8.11}$$

$$\mathbf{q}(i+1) = \Delta t \mathbf{v}(i+1) + \mathbf{q}(i). \tag{8.12}$$

The discrete mass matrix $\mathbf{M}$ and force vector $\mathbf{F}$ become

$$\tilde{\mathbf{M}} = \mathbf{M} + \Delta t \left(\mathbf{D} - \frac{\partial \mathbf{F}}{\partial \mathbf{v}} \right) + \Delta t^2 \left(\mathbf{K}(\mathbf{q}) - \frac{\partial \mathbf{F}}{\partial \mathbf{q}} \right),$$

$$\tilde{\mathbf{F}} = \mathbf{F}(i) + \left(\frac{1}{\Delta t}\mathbf{M} + \mathbf{D} - \frac{\partial \mathbf{F}}{\partial \mathbf{v}} \right) \mathbf{v}(i). \tag{8.13}$$

Chapter 20 offers a more detailed discussion on the efficient simulation of linear elastic models with corotational methods, as well as the implementation of virtual coupling with a deformable tool.

8.4.3 Contact Constraints

Contact constraints model the environment as algebraic equations in the configuration space of the tool, $g_i(\mathcal{T}) \geq 0$. A configuration of the tool $\mathcal{T}_0$ such that $g_i(\mathcal{T}_0) = 0$ indicates that the tool is exactly in contact with the environment. Collision response exerts forces on the tool such that environment constraints are not violated. We will look at two specific ways

of modeling environment contact constraints, penalty-based methods and Lagrange multipliers, and we will focus as an example on their application to a rigid tool.

Penalty method. In general terms, the penalty method models contact constraints as springs whose elastic energy increases with object interpenetration. Penalty forces are computed as the negative gradient of the elastic energy, which produces collision response that moves objects toward a non-penetrating configuration.

For simplicity, we will consider here linearized point-on-plane contacts. Given a colliding point $\mathbf{p}$ of the tool, a general contact constraint has the form $g_i(\mathbf{p}) \geq 0$, and after linearization $\mathbf{n}^T(\mathbf{p} - \mathbf{p}_0) \geq 0$, where $\mathbf{n} = \nabla g_i$ is the normal of the constraint (i.e., the normal of the contact plane), and $\mathbf{p}_0$ is the contact point on the environment. With such a linearized constraint, penetration depth can easily be defined as $\delta = -\mathbf{n}^T(\mathbf{p} - \mathbf{p}_0)$.

Penalty energies can be defined in multiple ways, but the simplest is to consider a Hookean spring, which yields an energy $E = \frac{1}{2}k\delta^2$, where k is the contact stiffness. Then, the contact penalty force becomes $\mathbf{F} = -\nabla E = -k\delta\nabla\delta$. It is also common to apply penalty forces when objects become closer than a certain tolerance d, which can be easily handled by redefining the penetration depth as $\delta = d - \mathbf{n}^T(\mathbf{p} - \mathbf{p}_0)$. The addition of a tolerance has two major advantages: the possibility of using penalty methods in applications that do not allow object interpenetration, and a reduction of the cost of collision detection. With the addition of a tolerance, object interpenetration occurs less frequently, and, as noted in Section 9.4, computation of penetration depth is notably more costly than computation of separation distance.

With a rigid tool, the contact point $\mathbf{p}$ can be expressed in terms of the rigid body state as $\mathbf{p} = \mathbf{x} + \mathbf{R}\mathbf{r}$, where $\mathbf{x}$ and $\mathbf{R}$ are, respectively, the position and orientation of the tool, and $\mathbf{r}$ is the position of the contact point in the tool's reference frame. Then, for the case of a rigid tool, and adding a damping term $\mathbf{b}$, the penalty force and torque are

$$\mathbf{F} = -k\mathbf{N}(\mathbf{x} + \mathbf{R}\mathbf{r} - \mathbf{p}_0) - b\mathbf{N}(\mathbf{v} + \omega \times (\mathbf{R}\mathbf{r})),$$

$$\mathbf{T} = (\mathbf{R}\mathbf{r}) \times \mathbf{F}, \tag{8.14}$$

where $\mathbf{N} = \mathbf{n}\mathbf{n}^T$ is a matrix that projects a vector onto the normal of the constraint plane.

Penalty-based methods offer several attractive properties: the force model is local to each contact and computationally simple, object interpenetration is inherently allowed, and the cost of the numerical integration is almost insensitive to the complexity of the contact configuration. This last property makes penalty-based methods well suited for interactive applications with fixed time steps, such as haptic rendering. In fact,

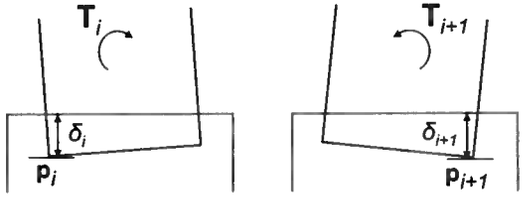


Figure 8.7. Torque discontinuity: (a) Penetration depth and torque at time t_i, with contact point $\mathbf{p}_i$; (b) Penetration depth and torque at time t_{i+1}, after the contact moves to contact point $\mathbf{p}_{i+1}$.

penalty-based methods have been applied in many 6-DOF haptic rendering approaches [McNeely et al. 99, Kim et al. 03, Johnson and Willemsen 03, McNeely et al. 06, Otaduy and Lin 06, Barbič and James 07].

However, penalty-based methods also have some disadvantages. For example, there is no direct control over physical parameters, such as the coefficient of restitution, and friction forces are difficult to model, as they require tracking contact points and using local methods [Karnopp 85, Hayward and Armstrong 00]. But, most importantly, geometric discontinuities in the location of contact points and/or normals lead to torque discontinuities, as depicted schematically in Figure 8.7. Different authors have proposed various definitions for contact points and normals, with various advantages and drawbacks. [McNeely et al. 99, McNeely et al. 06, Barbič and James 07] sample the objects with a discrete set of points, and define contact points as the penetrating subset. [Johnson and Willemsen 03, Otaduy and Lin 06], on the other hand, employ continuous surface definitions, and define contact points as local extrema of the distance function between colliding surfaces. Using a fixed discrete set of points allows for increased force continuity, while using continuous surface definitions allows for the detection of all interpenetrations. With the strict definition of penalty energy given above, penalty force normals are defined as the gradient of penetration depth, which is discontinuous on the medial axis of the objects. [McNeely et al. 99, McNeely et al. 06, Barbič and James 07] avoid this problem by defining as contact normal the surface normal at each penetrating point. This alternative definition is continuous in time, but does not guarantee that contact forces reduce interpenetration.

With penalty-based methods, non-penetration constraints are enforced by means of very high contact stiffness, which could yield instability problems if numerical integration is executed using fast explicit methods. The use of implicit integration of the tool, as described in Section 8.4.1, enhances stability in the presence of high contact stiffness [Wu 00, Larsen 01, Otaduy

and Lin 06, Barbič and James 07]. However, the dynamic equations of the different dynamic bodies (see Equation (8.4) for rigid bodies or Equation (8.12) for deformable bodies) then become coupled, and a linear system must be solved for each contact group. We refer to [Otaduy and Lin 06] for further details on the Jacobians of penalty force and torque for 6-DOF haptic rendering.

Lagrange multipliers. The method of Lagrange multipliers allows for an exact enforcement of contact constraints $\mathbf{g}(\mathcal{T}) \geq 0$ by modeling workless constraint forces $\mathbf{F}_c = \mathbf{J}^T\lambda$ normal to the constraints. Here we consider multiple constraints grouped in a vector $\mathbf{g}$, and their generalized normals are gathered in a matrix $\mathbf{J}^T = \nabla\mathbf{g}$. Constraint forces are added to regular forces of the dynamic equations of a colliding object (e.g., the tool). Then, constraints and dynamics are formulated in a joint differential algebraic system of equations. The "amount" of constraint force λ is the unknown of the system, and it is solved such that constraints are enforced.

Typically, contact constraints are nonlinear, but the solution of constrained dynamics systems can be accelerated by linearizing the constraints. Given the state $\mathbf{q}(i)$ of the tool at a certain time, constraint linearization yields $\mathbf{g}(i+1) \approx \mathbf{g}(i) + \Delta t\mathbf{J} \cdot \mathbf{v}(i+1)$. This linearization, together with the discretized state update equation, yields the following system to be solved per simulation frame:

$$\begin{aligned} \tilde{\mathbf{M}} \cdot \mathbf{v}(i+1) &= \Delta t\tilde{\mathbf{F}} + \mathbf{J}^T\lambda, \\ \mathbf{J} \cdot \mathbf{v}(i+1) &\geq -\frac{1}{\Delta t}\mathbf{g}(i). \end{aligned} \tag{8.15}$$

The addition of constraints for non-sticking forces $\lambda \geq 0$, $\lambda^T\mathbf{g}(\mathbf{q}) = 0$ yields a *linear complementarity problem* (*LCP*) [Cottle et al. 92], which combines linear equalities and inequalities. The problem in Equation (8.15) is a mixed LCP and can be transformed into a strict LCP through algebraic manipulation:

$$\mathbf{J}\tilde{\mathbf{M}}^{-1}\mathbf{J}^T\lambda \geq -\frac{1}{\Delta t}\mathbf{g}(i) - \Delta t\mathbf{J}\tilde{\mathbf{M}}^{-1}\tilde{\mathbf{F}}. \tag{8.16}$$

The LCP can be solved through various techniques [Cottle et al. 92], and once the Lagrange multipliers λ are known, it is possible to update the state of the tool.

There are other variants of the problem, for example by allowing sticking forces through equality constraints, or differentiating the constraints and expressing them on velocities or accelerations. Several of these variants of contact constraints with Lagrange multipliers have been employed in practical solutions to haptic rendering, some of them covered in detail in this book. Section 15.2.1 discusses the god-object method of [Zilles and

Salisbury 95], the first application of Lagrange multipliers for contact in 3-DOF haptic rendering. Chapter 16 describes an extension of the god-object method to 6-DOF rendering, and Chapter 20 formulates in detail frictional contact for haptic rendering of deformable objects.

Constraint-based methods with Lagrange multipliers handle all concurrent contacts in a single computational problem and attempt to find contact forces that produce physically and geometrically valid motions. As opposed to penalty-based methods, they solve one global problem, which allows, for example, for relatively easy inclusion of accurate friction models. However, constraint-based methods are computationally expensive, even for the linearized system in Equation (8.15), and the solution of constrained dynamics and the definition of constraints (i.e., collision detection) are highly intertwined. The full problem of constrained dynamics is highly nonlinear, but there are various time-stepping approaches that separate a collision-free dynamics update, collision detection, and collision response, for solving locally linear problems [Bridson et al. 02, Cirak and West 05]. Fast enforcement of constrained motion is, however, still a topic of research in haptic rendering, in particular for rendering deformable objects.

8.5 Multirate Algorithm

As discussed in Section 8.3.2, rendering algorithms based on virtual coupling [Colgate et al. 95] can serve to easily design stable rendering. However, the complexity of tool and environment simulation may require low update rates, which turn into low admissible coupling stiffness, and hence low-quality rendering.

Independently of the simulation and collision detection methods employed, and the mechanical characteristics of the tool or the environment, a common solution for enhancing the quality and transparency of haptic rendering is to devise a multirate algorithm (see [Barbagli et al. 03] for stability analysis of multirate algorithms). A slow process computes accurate interaction between the tool and the environment and updates an approximate but simple intermediate representation [Adachi et al. 95]. Then, a fast process synthesizes the forces to be sent to the device, using the intermediate representation. There have been two main approaches for designing intermediate representations, which we discuss next.

8.5.1 Geometric vs. Algebraic Intermediate Representations

One approach is to design a local and/or coarse geometric representation of the tool and/or the environment. A slow thread performs a computation of the interaction between the full representations of the tool and the environment, and updates the local coarse representation. In parallel, a fast thread computes the interaction between tool and environment using the simplified representations. The fast computation involves identifying simplified contact constraints, through collision detection, and computing the rendering forces. Note that this approach can be used in the context of both virtual coupling algorithms or direct rendering.

The earliest example of multirate rendering by [Adachi et al. 95] computes collision detection between a point tool and the environment in the slow thread, approximates the environment as a plane, and then uses the plane representation in the fast thread. A similar approach was followed by [Mark et al. 96], with addition of plane filtering between local model updates. Others used meshes of different resolutions coupled through Norton equivalents [Astley and Hayward 98], or local linearized submeshes for approximating high-frequency behavior [Çavuşoğlu and Tendick 00]. Recently, [Johnson et al. 05] have suggested the use of local collision detection algorithms for updating the contact constraints in the fast loop.

The second approach is to design a simplified representation of the collision response model between the tool and the environment. The slow thread performs full computation of collision detection and response between tool and environment, and updates a simplified version of the collision response model. This model is then used in the fast thread for computing collision response for rendering forces to the user. The main difference with the geometric approach is that the fast thread does not recompute collision detection for defining contact constraints.

Early approaches to multirate simulation of deformable models considered force extrapolation for defining the local algebraic model [Picinbono et al. 00]. This book also describes in further detail two recent approaches that identify contact constraints in the slow thread, and then use those constraints for force computation in the fast thread, for rigid bodies [Ortega et al. 06] (in Chapter 16), or for deformable bodies [Duriez et al. 04] (in Chapter 20). Apart from those, we should mention the use of contact constraints for computing a least-squares solution to Poisson's restitution hypothesis for rigid bodies [Constantinescu et al. 05]. Last, the rest of this section describes two examples that compute in the slow thread a linear model of the contact response between the tool and the environment, and then simply evaluate this linear model in the fast thread, for penalty-based methods [Otaduy and Lin 05, Otaduy and Lin 06] or for constraint-based methods with Lagrange multipliers [Otaduy and Gross 07].

8.5.2 Example 1: Multirate Rendering with Penalty Methods

Figure 8.8 shows the structure of the rendering algorithm suggested by [Otaduy and Lin 05, Otaduy and Lin 06]. The *visual thread* computes

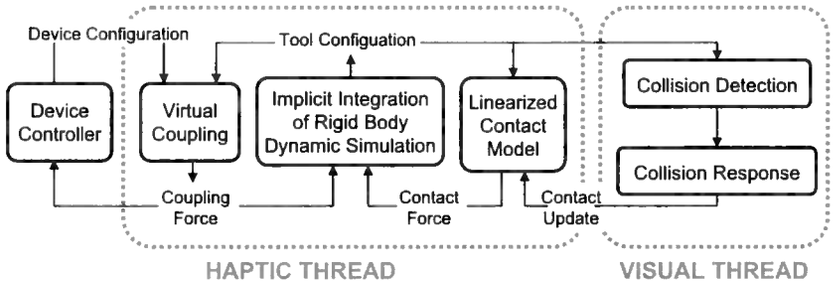


Figure 8.8. Multirate rendering architecture with a linearized contact model. A haptic thread runs at force update rates simulating the dynamics of the tool and computing force feedback, while a visual thread runs asynchronously and updates the linearized contact model [Otaduy and Lin 05, Otaduy and Lin 06].

collision detection between the tool and the environment, as well as collision response using the penalty-based method (see Section 8.4.3). Moreover, the equations of collision response are linearized, and the linear model is fed to the *haptic thread*. The haptic thread runs at fast haptic update rates, solving for the configuration of the tool, subject to forces computed using the linearized contact model. Figure 8.9 shows one application scenario of the multirate rendering algorithm.

[Otaduy and Lin 05, Otaduy and Lin 06] applied the linearization of penalty-based forces to 6-DOF haptic rendering with a rigid tool. Recall Equation (8.14), which describes penalty forces for a rigid tool. Assuming that the visual thread computes collision detection for a configuration $(\mathbf{q}_0, \mathbf{v}_0)$ of the tool, a penalty-based contact model can be linearized in

Figure 8.9. Virtual interaction using a linearized contact model. Dexterous interaction of an upper jaw (47,339 triangles) being moved over a lower jaw (40,180 triangles), using the method by [Otaduy and Lin 05, Otaduy and Lin 06]. (© 2005 IEEE)

general as

$$\mathbf{F}_c(\mathbf{q}, \mathbf{v}) \approx \mathbf{F}_c(\mathbf{q}_0, \mathbf{v}_0) + \frac{\partial \mathbf{F}_c}{\partial \mathbf{q}}(\mathbf{q} - \mathbf{q}_0) + \frac{\partial \mathbf{F}_c}{\partial \mathbf{v}}(\mathbf{v} - \mathbf{v}_0). \tag{8.17}$$

For more details on the linear approximation for a rigid tool, we refer to [Otaduy and Lin 05, Otaduy and Lin 06].

An interesting observation of the linearized penalty-based method is that it imposes no additional cost if the rendering algorithm computes dynamics of the tool with implicit integration. As shown in Equation (8.5), the definition of discrete-time inertia requires the same Jacobians $\frac{\partial \mathbf{F}}{\partial \mathbf{q}}$ and $\frac{\partial \mathbf{F}}{\partial \mathbf{v}}$ as the linearized contact model. We would like to point out that these Jacobians are also used in quasi-static methods for solving the configuration of the tool [Wan and McNeely 03, Barbič and James 07].

8.5.3 Example 2: Multirate Rendering with Constraints

Figure 8.10 shows the overall structure of the multirate rendering algorithm presented by [Otaduy and Gross 07] for 6-DOF haptic rendering between a rigid tool and a deformable environment. This algorithm creates two instances of the rigid tool manipulated by the user. The visual thread, typically running at a low update rate (as low as tens of Hz), performs a full simulation of the *visual tool* coupled to the haptic device and interacting with a deformable environment. The haptic thread, running at a fast update rate of typically 1 kHz, performs the simulation of the *haptic tool* and computes force values to be rendered by the haptic device. Collision detection and full constraint-based collision response are only computed in the visual thread. At the same time, the parameters of a linear contact model are updated, and fed to the haptic thread. This linear model can be evaluated with a fixed, low number of operations, and ensures extremely fast update of contact forces in the haptic thread.

For penalty-based collision response, [Otaduy and Lin 05] proposed a linearized contact model in the state space of the tool. However, for constraint-based collision response, [Otaduy and Gross 07] proposed a model of *contact Jacobian*, linearly relating contact forces $\mathbf{F}_c$ and the rest of the forces $\tilde{\mathbf{F}}$ acting on the tool. The linearized model takes the form

$$\mathbf{F}_c(\tilde{\mathbf{F}}) \approx \mathbf{F}_c(\tilde{\mathbf{F}}_0) + \frac{\partial \mathbf{F}_c}{\partial \tilde{\mathbf{F}}}(\tilde{\mathbf{F}} - \tilde{\mathbf{F}}_0). \tag{8.18}$$

All that needs to be done in the visual thread is to compute the contact Jacobian $\frac{\partial \mathbf{F}_c}{\partial \tilde{\mathbf{F}}}$.

The LCP formulation in Equation (8.16) for collision response can be compactly rewritten as $\mathbf{A}_\lambda \lambda \geq \mathbf{b}_\lambda$. The resolution of the LCP yields a

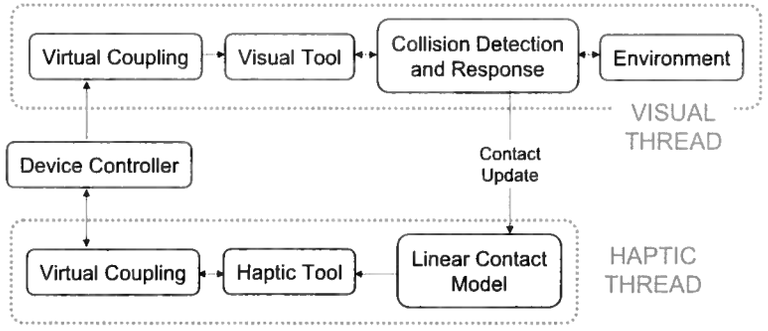


Figure 8.10. Multirate rendering using a discrete-time contact Jacobian [Otaduy and Gross 07].

set of inactive contacts, for which the contact force is zero, and a set of active contacts for which the constraints hold exactly $\mathbf{A}_{\lambda,a}\lambda_a = \mathbf{b}_a \Rightarrow \lambda_a = \mathbf{A}_{\lambda,a}^{-1}\mathbf{b}_a$. The contact force can then be written in terms only of active contacts as $\mathbf{F}_c = \mathbf{J}_a^T\mathbf{A}_{\lambda,a}^{-1}\mathbf{b}_a$. Then, the contact Jacobian can be easily formulated as

$$\frac{\partial \mathbf{F}_c}{\partial \tilde{\mathbf{F}}} = \frac{\partial \mathbf{F}_c}{\partial \mathbf{b}_a} \cdot \frac{\partial \mathbf{b}_a}{\partial \tilde{\mathbf{F}}} = -\Delta t \mathbf{J}_a^T \mathbf{A}_{\lambda,a}^{-1} \mathbf{J}_a \tilde{\mathbf{M}}^{-1}. \tag{8.19}$$

This formulation involves several approximations, such as ignoring the change of active constraints between time steps or changes of inertia. Note also that Equation (8.19) should be slightly modified to account for a moving or deforming environment, as the state of the tool and the environment are not explicitly separated. Multirate algorithms enable programming very high rendering stiffness, under the assumption that the contact space changes slowly. This is in fact the case in many situations, especially during sliding contact between tool and environment.

9

Overview on Collision and Proximity Queries

M. C. Lin and D. Manocha

In a geometric context, a collision or proximity query reports information about the relative configuration or placement of two objects. Some of the common examples of such queries include checking whether two objects overlap in space or their boundaries intersect, or computing the minimum Euclidean separation distance between their boundaries, etc.

Many publications have been written on different aspects of these queries in computer graphics, computational geometry, robotics, computer-aided design, virtual environments, and haptics. These queries arise in diverse applications including robot motion planning, virtual prototyping, dynamic simulation, computer gaming, interactive walkthroughs, molecular modeling, etc.

For haptic rendering, in order to create a sense of touch between the user's hand and a virtual object, contact or restoring forces are generated to prevent penetration into the virtual model. This step requires collision detection, penetration depth computation, and determining the contact forces. Often, separation distances between pairs of objects are also computed to estimate time of collision as well.

This chapter[1] gives an overview of different queries and various classes of algorithms for performing queries for different types of geometric models. These techniques include algorithms for collision detection, distance queries, and penetration depth query among convex polytopes, non-convex polygonal models, and curved objects, as well dynamic queries and handling of large environments consisting of multiple objects.

[1] A preliminary version appeared in [Lin and Manocha 03].

15 Three-Degree-of-Freedom Rendering

C. Basdogan, S. D. Laycock, A. M. Day,
V. Patoglu, and R. B. Gillespie

This chapter will introduce the fundamental metaphor for haptic interaction: single-point contact or *three-degree-of-freedom haptic rendering.* Due to its fundamental aspect, we first review some of the introductory concepts about human-computer interaction through a haptic display.

Then, we describe the basic techniques for rendering contact between a single point and a 3D object in a virtual environment. This interaction metaphor corresponds to feeling and exploring the same 3D object through the tip of a stylus in the real world. Throughout the chapter, we refer to the single contact point as a *haptic interface point* (HIP), as shown in Figure 15.1. Three-DOF haptic rendering should be distinguished from 6-DOF rendering, which involves object-object interaction and is covered in later chapters in this book.

15.1 Human-Machine Coupling

Some may consider haptic rendering as significantly more complex (and interesting) than visual rendering, since it is a *bilateral process*: display (rendering) cannot be divorced from manipulation. Thus any haptic rendering algorithm is intimately concerned with tracking the inputs of the user, as well as displaying the haptic response. Also, note that haptic rendering is computationally demanding, due to the high sampling rates required. While the visual system perceives seamless motion when flipping through 30 images per second, the haptic system requires signals that are refreshed at least once every millisecond. These requirements are driven by human haptic ability to detect vibrations that peaks at about 300 Hz but ranges all the way up to 1000 Hz. Note that vibrations up to 1000 Hz might be required to simulate fast motion over fine texture, but also might be required for sharp, impulsive rendering of a changing contact condition.

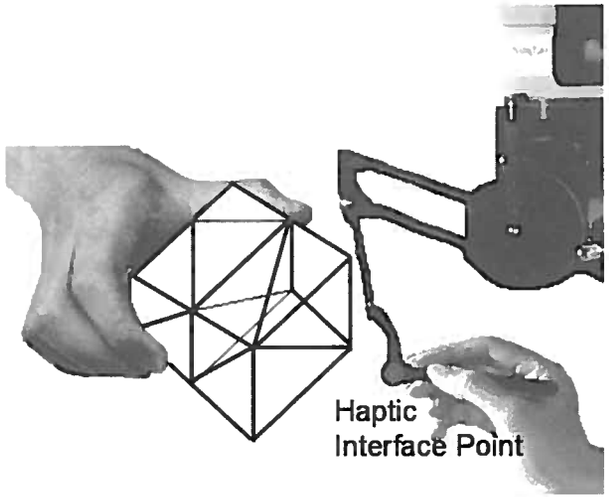


Figure 15.1. Point-based haptic interactions with 3D objects in virtual environments.

Haptic rendering requires a haptic interface, a computationally mediated virtual environment, and a control law, according to which the two are linked. Figure 15.2 presents a schematic view of a haptic interface and the manner in which it is most commonly linked to a virtual environment. On the left portion of the figure, mechanical interaction takes place between a human and the haptic interface device, or more specifically, between a fingertip and the device end-effector. In the computational domain depicted on the right, an image of the device end-effector E is connected to a proxy P through what is called the *virtual coupler*. The proxy P in turn interacts with objects such as A and B in the virtual environment. Proxy P might take on the shape of the fingertip or a tool in the user's grasp.

The virtual coupler is depicted as a spring and damper in parallel, which is a model of its most common computational implementation, though generalizations to 3D involve additional linear and rotary spring-damper pairs not shown. The purpose of the virtual coupler is twofold. First, it links a forward-dynamics model of the virtual environment with a haptic interface designed for impedance-display.[1] Relative motion (displacement and velocity) of the two ends of the virtual coupler determines, through the

[1] *Impedance display* describes a haptic interface that senses motion and sources forces and moments. An admittance display sources motion and senses forces and moments. Most haptic interface devices are controlled using impedance display, which may be implemented using low-inertia motors and encoders connected to the mechanism through low-friction, zero-backlash, direct-drive or near-unity mechanical advantage transmissions. Impedance display does not require force or torque sensors.

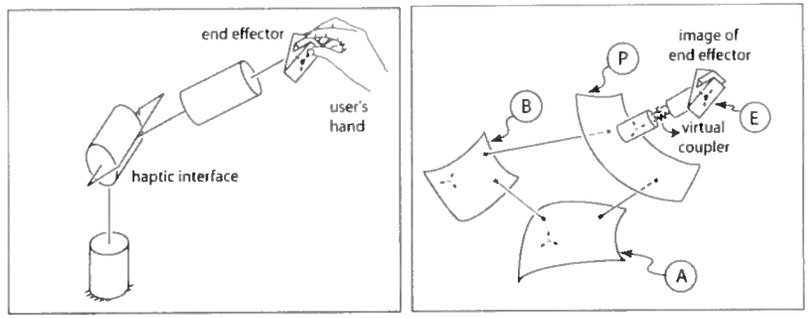


Figure 15.2. This two-part figure presents a schematic representation of haptic rendering. The left figure corresponds to the physical world, where a human interacts with the haptic device. The figure on the right depicts the computationally implemented virtual environment.

applicable spring and damper constants, the forces and moments to be applied to the forward dynamics model and the equal and opposite forces and moments to be displayed by the haptic interface. Note that the motion of P is determined by the forward dynamics solution, while the motion of E is specified by sensors on the haptic interface. The second role of the virtual coupler is to filter the dynamics of the virtual environment so as to guarantee stability when display takes place through a particular haptic device. The parameters of the virtual coupler can be set to guarantee stability when parameters of the haptic device hardware are known and certain input-output properties of the virtual environment are met. Thus the virtual coupler is most appropriately considered part of the haptic interface, rather than part of the virtual environment [Adams and Hannaford 99, Adams and Hannaford 02]. If an admittance-display architecture is used, an alternate interpretation of the virtual coupler exists, though it plays the same two basic roles. For further discussion of the virtual coupler in performance/stability tradeoffs in either the impedance or admittance-display cases, see the work done by [Adams and Hannaford 99, Adams and Hannaford 02, Miller et al. 00].

One final note can be made with reference to Figure 15.2: rigid bodies in the virtual environment, including P, have both configuration and shape—they interact with one another according to their dynamic and geometric models. Configuration (including orientation and position) is indicated in Figure 15.2 using reference frames (three mutually orthogonal unit vectors) and reference points fixed in each rigid body. Shape is indicated by a surface patch. Note that the image of the device end-effector E has configuration, but no shape. Its interaction with P takes place through the virtual coupler and requires only the configuration of E and P.

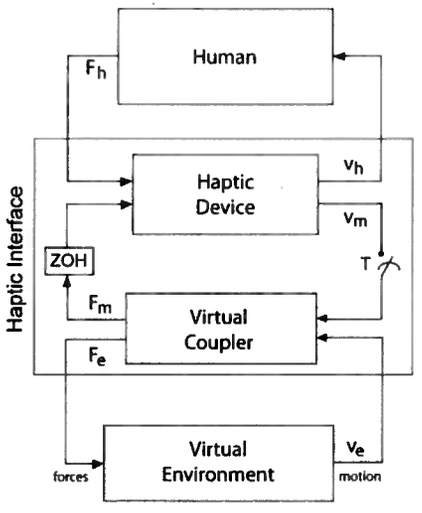


Figure 15.3. A block diagram of haptic rendering according to an impedance display architecture. There are three major blocks in the diagram modeling, input/output characteristics of the human, the haptic interface, and the virtual environment.

The various components in Figure 15.2, including the human user, haptic device, virtual coupler, and virtual environment, form a coupled dynamical system whose behavior depends on the force/motion relationship established by the interconnection of the components. Figure 15.3 shows these components interconnected in a block diagram, where the additional indication of causality has been made. Causality expresses which force and motion variables are inputs and which are outputs for each component. For example, the human operates on the velocity v_h (common to the finger and end-effector) to produce the force F_h imposed on the haptic device. The haptic device is a two-port that operates on the force F_h imposed by the human and the force F_m produced by its motors to produce the velocities v_h and v_m. Usually, by careful haptic device transmission design, v_h and v_m are the same, and are measured with a single encoder. Intervening between the human and haptic device, which live in the continuous, physical world and the virtual coupler and virtual environment, which live in the discrete, computed world, are a sampling operator T and *zero-order hold* (ZOH). The virtual coupler is shown as a two-port that operates on velocities v_m and v_e to produce the motor command force F_m and force F_e imposed on the virtual environment. Forces F_m and F_e are usually equal and opposite. Finally, the virtual environment is shown in its forward dynamics form, operating on applied forces F_e to produce response motion v_e. Naturally,

the haptic device may use motors on its joints, so the task-space command forces F_m must first be mapped through the manipulator Jacobian before being applied to the motors.

Note that the causality assumption for the human is by itself rather arbitrary. However, causality for the haptic device is essentially determined by electro-mechanical design, and causality for the virtual coupler and virtual environment is established by the implementation of a discrete algorithm. The causality assumptions in Figure 15.3 correspond to impedance display. Impedance display is the most common, but certainly not the only, possible implementation. See [Adams and Hannaford 99] for a framework and analysis using network diagrams (that do not indicate causality), which is more general.

15.2 Single-Point Rendering of 3D Rigid Objects

Typically, a haptic rendering algorithm is made of two parts: (a) collision detection and (b) collision response (see Figure 15.4). As the user manipulates the probe of the haptic device, the new position and orientation of the haptic probe are acquired, and collisions with the virtual objects are detected (i.e., *collision detection*). If a collision is detected, the interaction

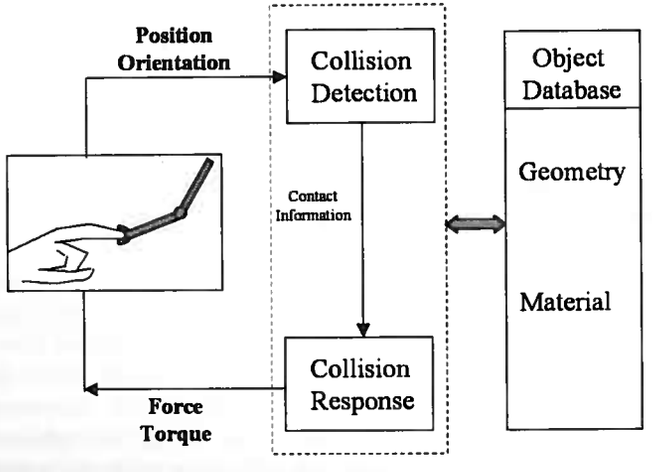
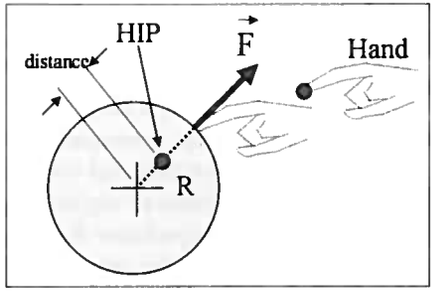


Figure 15.4. A haptic interaction algorithm is typically made of two parts. (a) Collision detection. (b) Collision response. The haptic loop seen in the figure requires an update rate of around 1 kHz for stable force interactions. Computationally fast collision detection and response techniques are necessary to accommodate this requirement.




```
void calculate force (Vector &force)
  {
     float X, Y, Z, distance;
     float R = 20.0;

     X = HIP[0]; Y = HIP[1]; Z = HIP[2];
     distance = sqrt(X*X + Y*Y + Z*Z;

     if (distance < R) // collision check
     {
        force[0] = X/distance * (R-distance);
        force[1] = Y/distance * (R-distance);
        force[2] = Z/distance * (R-distance);

     }

  }
```

Figure 15.5. Haptic rendering of a 3D sphere in virtual environments. The software code presented on the right-hand side calculates the direction and the magnitude of the reaction force for the sphere discussed in the example. The sphere has a radius of 20 units and is located at the origin.

forces are computed using preprogrammed rules for *collision response* and conveyed to the user through the haptic device to provide him/her with the tactual representation of 3D objects and their surface details. Hence, a haptic loop, which updates forces around 1 kHz (otherwise, virtual surfaces feel softer, or, at worst, instead of a surface, it feels as if the haptic device is vibrating), include at least the following function calls:

- Position and/or orientation of the end-effector
  `get_position (Vector &position);`
- User-defined function to calculate forces
  `calculate_force (Vector &force);`
- Calculate joint torques and reflect forces back to the user
  `send_force (Vector force);`

To describe the basic concepts of haptic rendering, let us consider a simple example: haptic rendering of a 3D frictionless sphere, located at the origin of a 3D virtual space (see Figure 15.5). Let us assume that the user can only interact with the virtual sphere through a single point that is the end point of the haptic probe, also known as the Haptic Interaction Point (HIP). In the real world, this is analogous to feeling the sphere with the tip of a stick. As we freely explore the 3D space with the haptic probe, the haptic device will not reflect any force to the user until a contact occurs. Since our virtual sphere has a finite stiffness, the HIP will penetrate into the sphere at the contact point. Once the penetration into the virtual sphere is detected and appropriate forces to be reflected back to the user are computed, the device will reflect opposing forces to our hand, to resist

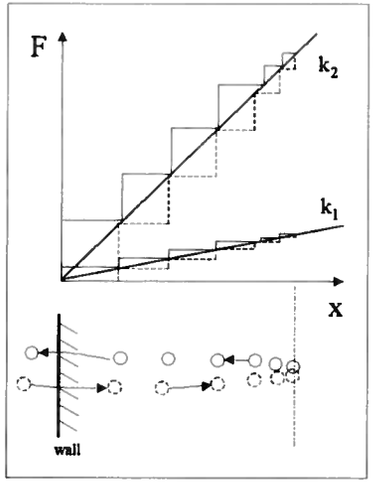


Figure 15.6. Force-displacement curves for touch interactions with real and virtual walls. In the case of a real wall, the force-displacement curve is continuous. However, we see the "staircase" effect when simulating touch interactions with a virtual wall. This is due to the fact that a haptic device can only sample position information with a finite frequency. The difference in the areas enclosed by the curves that correspond to penetrating into and out of the virtual wall is a manifestation of energy gain. This energy gain leads to instabilities as the stiffness coefficient is increased (compare the energy gains for stiffness coefficients k_1 and k_2). On the other hand, a low value of the stiffness coefficient generates a soft wall, which is not desirable, either.

further penetration. We can easily compute the magnitude of the reaction force by assuming that it is proportional to the depth of penetration. Assuming no friction, the direction of this force will be along the surface normal, as shown in Figure 15.5.

As it can be seen from the example given above, a rigid virtual surface can be modeled as an elastic element. Then, the opposing force acting on the user during the interaction will be

$$\vec{F} = k\,\Delta\vec{x} \tag{15.1}$$

where k is the stiffness coefficient and $|\Delta\vec{x}|$ is the depth of penetration. While keeping the stiffness coefficient low would make the surface feel soft, setting a high value can make the interactions unstable by causing undesirable vibrations. Figure 15.6 depicts the changes in force profile with respect to position for real and virtual walls. Since the position of the probe tip is sampled digitally with certain frequency during the simulation of a virtual wall, a "staircase" effect is observed. This staircase effect leads to

energy generation (see the discussions and suggested solutions in [Colgate and Brown 94, Ellis et al. 96, Gillespie and Cutkosky 96]).

Although the basic recipe for haptic rendering of virtual objects seems easy to follow, rendering complex 3D surfaces and volumetric objects requires more sophisticated algorithms than the one presented for the sphere. The stringent requirement of updating forces around 1 kHz leaves us very little CPU time for computing the collisions and reflecting the forces back to the user in real time when interacting with complex shaped objects. In addition, the algorithm given above for rendering of a sphere considered only "point-based" interactions (as if interacting with objects through the tip of a stick in real world), which is far from what our hands are capable of in the real world. However, several haptic rendering techniques have been developed to simulate complex touch interactions in virtual environments. The existing techniques for haptic rendering with force display can be distinguished based on the way the probing object is modeled: (1) a point [Zilles and Salisbury 95, Adachi et al. 95, Avila and Sobierajski 96, Ruspini et al. 97, Ho et al. 99], (2) a line segment [Basdogan et al. 97, Ho et al. 00], or (3) a 3D object made of group of points, line segments, and polygons [McNeely et al. 99, Nelson et al. 99, Gregory et al. 00b, Johnson and Willemsen 03, Laycock and Day 05, Otaduy and Lin 05]. The type of interaction method used in simulations depends on the application.

In point-based haptic interactions, only the end point of the haptic device interacts with virtual objects. Each time the user moves the generic probe of the haptic device, the collision detection algorithm checks to see if the end point is inside the virtual object. If so, the depth of indentation is calculated as the distance between the current HIP and the corresponding surface point, also known as the *ideal haptic interface point* (IHIP), god-object, proxy point, or surface contact point. For exploring the shape and surface properties of objects in VEs, point-based methods are probably sufficient and could provide the users with force feedback similar to that experienced when exploring the objects in real environments with the tip of a stylus.

An important component of any haptic rendering algorithm is the collision response. Merely detecting collisions between 3D objects is not enough for simulating haptic interactions. How the collision occurs and how it evolves over time (i.e., contact history) are crucial factors in haptic rendering to accurately compute the interaction forces that will be reflected to the user through the haptic device [Ho et al. 99, Basdogan and Srinivasan 02]. In other words, the computation of IHIP relies on the contact history. Ignoring contact history and always choosing the closest point on the object surface as our new IHIP for a given HIP would make the user feel as if he or she is pushed out of the object. For example, Figure 15.7(a) shows a thin object with the HIP positioned at three successive time steps.

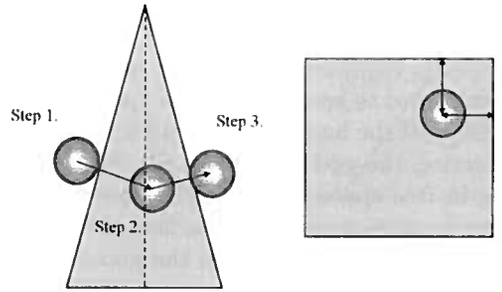


Figure 15.7. (a) The Haptic Interface Point can be forced out of the wrong side of thin objects. (b) The Haptic Interface Point is equidistant to both faces. The algorithm is unable to decide which face is intersected first by looking at a single time step.

Step 1 shows the HIP on the left hand side just coming into contact with the thin object. At Step 2, the HIP has penetrated the thin object and is now closer to the right face of the object. The HIP will be forced out the other side, producing an undesired result. A similar problem will occur if the HIP is located equidistant to two faces of the virtual object. Figure 15.7(b) illustrates this case, and it is unclear which face normal to choose by looking at a single time step. The HIP could easily be forced out of the object in the incorrect direction. To overcome these problems, an approach is required to keep a contact history of the position of the HIP. The next section discusses techniques that, among other advantages, overcome these problems.

The algorithms developed for 3-DOF point-based haptic interaction depend on the geometric model of the object being touched: (1) surface models and 2) volumetric models. The surface models can be also grouped as 1) polygonal surfaces, (2) parametric surfaces, and (3) implicit surfaces.

15.2.1 Polygonal Surfaces

Virtual objects have been modeled using polygonal models in the field of computer graphics for decades, due to their simple construction and efficient graphical rendering. For similar reasons, haptic rendering algorithms were developed for polygonal models and triangular meshes in particular. Motivating this work was the ability to directly augment the existing visual cues with haptic feedback, utilizing the same representations.

The first method to solve the problems of single point haptic rendering for polygonal models was developed at the Massachusetts Institute of Technology (MIT) [Zilles and Salisbury 95]. The method enabled a con-

tact history to be kept, and at the time it was able to provide stable force feedback interactions with polygonal models of 616 triangular faces on a computer with a 66 MHz Pentium processor. A second point known as the "god-object" was employed to keep the contact history. It would always be collocated with the HIP if the haptic device and the virtual object were infinitely stiff. In practice, the god-object and the HIP are collocated when the HIP is moving in free space. As the HIP penetrates the virtual objects, the god-object is constrained to the surface of the virtual object. An approach is subsequently required to keep the god-object on the surface of the virtual object as the HIP moves around inside. Constraint based approaches that keep a point on the surface sometimes refer to this point as the *surface contact point* (SCP). The position of the god-object can be determined by minimizing the energy of a spring between the god-object and the HIP, taking into account constraints represented by the faces of the virtual object. By minimizing L in Equation (15.2) the new position can be obtained. The first line of the equation represents the energy in the spring and the remaining three lines represent the equations of three constraining planes. The values x, y, and z are the coordinates of the god-object, and x_p, y_p, and z_p represent the coordinates of the HIP:

$$\begin{aligned} L =& \frac{1}{2}(x - x_p)^2 + \frac{1}{2}(y - y_p)^2 + \frac{1}{2}(z - z_p)^2 \\ &+ l_1(A_1 x + B_1 y + C_1 z - D_1) \\ &+ l_2(A_2 x + B_2 y + C_2 z - D_2) \\ &+ l_3(A_3 x + B_3 y + C_3 z - D_3), \end{aligned} \tag{15.2}$$

where, L = value to be minimized, l_1, l_2, l_3 = Lagrange multipliers, and A, B, C, D = coefficients for the constraint plane equations.

To efficiently solve this problem, a matrix can be constructed and used in Equation (15.3) to obtain the new position of the god-object, as given by x, y, and z. When there are three constraining planes limiting the motion of the god-object, the method only requires 65 multiplications to obtain the new coordinates. The problem is reduced as the number of constraint planes is reduced:

$$\begin{pmatrix} 1 & 0 & 0 & A_1 & A_2 & A_3 \\ 0 & 1 & 0 & B_1 & B_2 & B_3 \\ 0 & 0 & 1 & C_1 & C_2 & C_3 \\ A_1 & B_1 & C_1 & 0 & 0 & 0 \\ A_2 & B_2 & C_2 & 0 & 0 & 0 \\ A_3 & B_3 & C_3 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ l_1 \\ l_2 \\ l_3 \end{pmatrix} = \begin{pmatrix} x_p \\ y_p \\ z_p \\ D_1 \\ D_2 \\ D_3 \end{pmatrix} \tag{15.3}$$

It was noted by [Ruspini et al. 97] that small numerical inconsistencies can cause gaps in polygonal models, which enable the god-object to slip

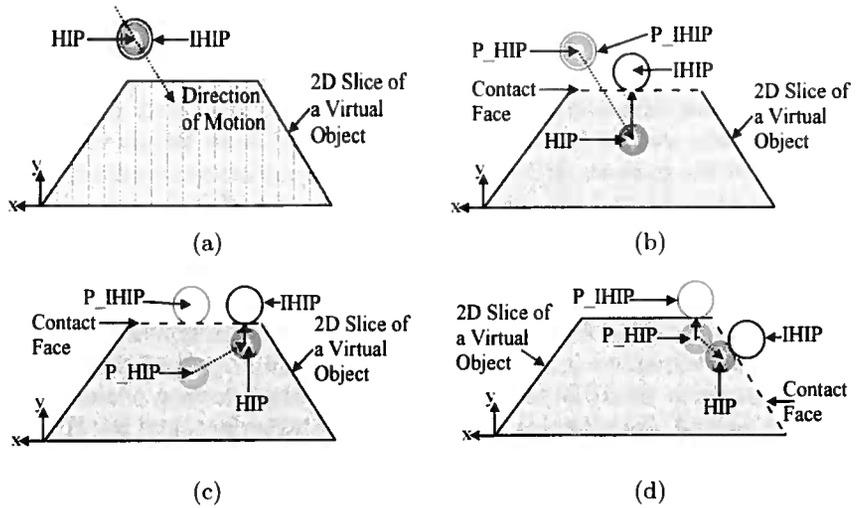


Figure 15.8. (a) The HIP and IHIP illustrated moving down towards the top of the two-dimensional slice of the virtual object. (b) The line segment between the P_HIP and the HIP intersects with the virtual objects. The contact face is shown with the dotted line and the IHIP is constrained to the surface. (c) The IHIP is tracked along the surface of the virtual object. (d) The HIP is now closer to a new feature of the virtual object, and so the contact face is updated.

into the virtual objects. To overcome this, the topology of the surface must be reconstructed. To avoid this reconstruction phase, a strategy similar in style to the previous approach was developed, which permitted the constrained point to have finite size [Ruspini et al. 97]. The algorithms employed by Ruspini et al. were originally developed for the robotics field. The term *virtual proxy* is used to refer to the spherical object constrained to the surface of the virtual objects. Its motion is akin to the motion of a robot greedily attempting to move towards a goal, in this case the HIP. Configuration space obstacles are constructed by wrapping the virtual object by a zone equal to the radius of the virtual proxy. Doing this enables a single point to be incorporated as the haptic probe once more. Lagrange multipliers can then be used, as described by [Zilles and Salisbury 95], to obtain the new position of the virtual proxy.

A more procedural constraint-based approach was developed by [Ho et al. 99] for single-point rendering. They state that it increases the servo-rate, facilitates stable haptic interactions, and importantly enables the servo rate to be independent of the number of polygons. They refer to their constrained point as the *ideal haptic interface point* (IHIP), with a force being sent to the haptic device, based on a spring between the

IHIP and HIP. Figure 15.8 illustrates an overview of the algorithm. A two-dimensional slice of a three-dimensional object has been included to represent the virtual object. Initially, the IHIP and the HIP are set at the same position, identical to the god-object and virtual proxy approaches. At each time step, a line segment is constructed from the previous HIP, P_HIP, to the current HIP. If there exists an intersection point between this line segment and the virtual object, then the IHIP is constrained to the closest point to the current HIP on the face nearest to the P_HIP. This is depicted in Figure 15.8(b). As the HIP moves, the IHIP is tracked over the surface of the mesh by choosing the closest feature to the HIP. For efficiency, only those features (edge, vertex, face) that bound the current feature are tested, as is depicted in Figures 15.8(c) and 15.8(d). When the vector from the IHIP to the HIP points in the direction of the current feature normal, then there is no contact with the surface, and the HIP and IHIP are once again collocated.

15.2.2 Parametric Surfaces

Polygonal representations are perfect for displaying simple objects, particularly those with sharp corners, but are limited when it comes to representing highly curved objects. In such case a large number of polygons would be required to approximate the curved surface, resulting in higher memory requirements. To overcome this, parametric surfaces have been used and are very important in 3D modeling packages and computer aided design (CAD) packages.

To directly render parametric models without performing a difficult conversion into a polygonal representation, haptic rendering algorithms have been developed to allow direct interaction with NURBS surfaces. In 1997, at the University of Utah [Thompson et al. 97] developed a technique for the haptic rendering of NURBS surfaces. The motivation was to be able to interact with a CAD modeling system using the Sarcos force-reflecting exoskeleton arm. The algorithm is broken into two phases. Firstly, the collision detection between the HIP and the surfaces is undertaken, and secondly they employ a direct parametric tracing algorithm to constrain a point to the surface as the HIP is permitted to penetrate the surface. The constrained point will be referred to as the surface contact point, SCP. The first stage of the collision detection uses bounding boxes encompassing the surfaces to aid in trivial rejection. If the HIP is inside the bounding box, then the HIP is projected onto the control mesh. The parameters (u, v) are defined for each vertex of the control mesh. The (u, v) parameters for the projected point can be obtained by interpolating the parameter values at the vertices. The distance between the HIP and the point on the surface can then be obtained. The direct parametric tracing method tracks the position of the HIP on the surface. As the HIP moves, it is projected onto the surface tangent plane, tangential to the gradient of the surface at the previous location of the SCP. The new SCP and tangent plane are then found by parametric projection, using the projected HIP [Thompson et al. 97]. The force returned to the device is based on a spring damper model between the HIP and the surface.

Alternatively, the minimum distance between convex parametric surfaces may be determined by formulating a nonlinear control problem and solving it with the design of a switching feedback controller [Patoglu and Gillespie 04,Patoglu and Gillespie 05]. The controller simply has the job of stabilizing the integration of the differential kinematics of the error vector that connects two candidate points, one drawn from each of two interacting surfaces. The controller manipulates the parameters (u, v) and (r, s) of the candidate points until the projections of the error vector onto all four surface tangents are driven to zero. With the design of a suitable feedback control law, the simulation of the differential kinematics produces an asymptotically convergent algorithm. While algorithms based on Newton's Iteration have a limited region of attraction, the algorithm built around the control formulation guarantees global uniform asymptotic stability, hence dictates that any pair of initial points belonging to the convex surface patches will converge to the closest point solution without ever leaving the patches [Patoglu and Gillespie 05]. Global convergence for a narrow phase algorithm greatly simplifies the design of a multi-phase algorithm with global convergence. The algorithm may be run as the surface patches move, where it becomes a tracking algorithm. Other notable features include no requirement for matrix inversion, high computational efficiency, and the availability of analytic limits of performance. Together with a top-level switching algorithm based on Voronoi diagrams, this closest point algorithm can treat parametric models formed by tiling together surface patches [Patoglu and Gillespie 05].

One of the most promising applications of rendering parametric surfaces is in free-form design. Free-form surfaces are defined by parametric functions, and the conventional methods of design using these surfaces require tedious manipulation of control points and careful specification of constraints. However, the integration of haptics into free form design improves the bandwidth of interactions and shortens the design cycle. [Dachille et al. 99] developed a method that permits users to interactively sculpt virtual B-spline objects with force feedback. In this approach, point, normal, and curvature constraints can be specified interactively and modified naturally using forces. Nowadays, commercial packages based on the free-form design concept (FreeForm Concept and FreeForm Modeling Plus from Sensable Tech.) offer alternative solutions to the design of jew-

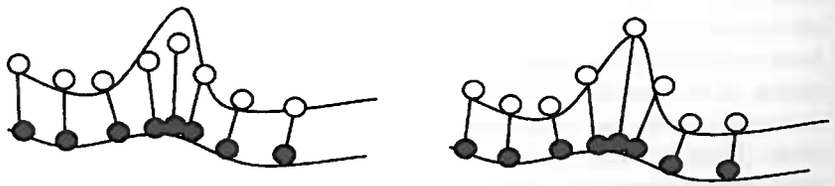
elry, sport shoes, animation characters, and many other products. Using these software solutions and a haptic device, the user can carve, sculpt, push, and pull instead of sketching, extruding, revolving, and sweeping as in traditional design.

15.2.3 Implicit Surfaces

An early approach for the haptic rendering of implicit surfaces was developed by [Salisbury and Tarr 97]. Their approach used implicit surfaces defined by analytic functions. Later, [Kim et al. 02a] developed a technique where an implicit surface is constructed that wraps around a geometric model to be graphically rendered. To ensure the surface can be accurately felt, a virtual contact point is incorporated. This point is constrained to the surface, as shown in Figure 15.9.

The method was used for virtual sculpting. The space occupied by the object is divided into a three-dimensional grid of boxes, in a similar strategy to the volume rendering techniques that will be discussed in the next section. Collision detection using implicit surfaces can be performed efficiently, since by using implicit surfaces it is possible to determine whether a point is interior, exterior, or on the surface by evaluating the implicit function with a point. The potential value is determined for each grid point, and then it can be used with an interpolation scheme to return the appropriate force. The surface normal at each grid point is calculated from the gradient of the implicit function. The surface normal for any point can then be determined by interpolating the values of the surface normals at the eight neighbors.

The previous two methods permit only one side of the surface to be touched. For many applications this is not sufficient, since users often require both sides of a virtual object to be touched. [Maneewarn et al. 99] developed a technique using implicit surfaces that enabled the user to interact with the exterior and interior of objects. The user's probe is restricted by the surface when approached from both sides.



(a) (b)

Figure 15.9. Computing the force magnitude. (a) Shows an approximate method. (b) Illustrates the approach by Kim et al. (Adapted from [Kim et al. 02a].)

15.2.4 Volumetric Models

Volumetric objects constructed from individual voxels can store significantly more information than a surface representation. The ability to visualize volume data directly is particularly important for medical and scientific applications. There are a variety of techniques for visualizing the volume data, such as the one proposed by [Lacroute and Levoy 94] for shearing, warping, and compositing two-dimensional image slices. In contrast, a method termed *splatting* can be used where a circular object can be rendered in each voxel [Laur and Hanrahan 91]. Each circular object is aligned to the screen and rendered to form the final image. Volume data can also be visualized indirectly by extracting a surface representation using methods such as Marching Cubes. Once the surface is extracted, haptic interaction can then take place using the surface-based methods discussed earlier [Eriksson et al. 05,Körner et al. 99]. However, the process of surface extraction introduces a number of problems. As it requires a preprocessing step, the user is prevented from modifying the data during the simulation, and it also can generate a large number of polygons. Furthermore, by only considering the surface, it is not possible to incorporate all the structures present in a complex volume. To create a complete haptic examination of volume data, a direct approach is required. [Iwata and Noma 93] were the first to enable haptic feedback in conjunction with volume data using a direct volume rendering approach, which they termed *volume haptization.* The approach illustrated ways of mapping 3D vector or scalar data to forces and torque. The mapping must determine the forces at interactive rates, and typically the forces must directly relate to the visualization of the data. This form of direct volume rendering is particularly useful for scientific visualization [Lawrence et al. 00a]. [Iwata and Noma 93] used their approach for the haptic interaction of data produced in computational fluid dynamics simulations. In this case, force could be mapped to the velocity and torque mapped to vorticity.

Utilizing computed tomography (CT) or magnetic resonance imaging (MRI) as a basis for volume rendering enables a three-dimensional view of patient specific data to be obtained. Enabling the user to interact with the patient data directly is useful for the medical field, particularly since surgeons commonly examine patients through their sense of touch. [Gibson 95] developed a prototype for the haptic exploration of a 3D CT scan of a human hip. The CT data is converted to voxels, with each voxel incorporating information for both haptic rendering and graphical rendering. The human hip is then inserted into an occupancy map, detailing where

the model is located in the voxel grid. The occupancy map consists of a regularly spaced grid of cells. Each cell either contains a null pointer or an address of one of the voxels representing an object in the environment. The size of the occupancy map is set to encompass the entire virtual environment. The fingertip position controlled by the haptic device is represented by a single voxel. Collision detection between the fingertip and the voxels representing the human hip are determined by simply comparing the fingertip voxel with the occupancy map for the environment.

[Avila and Sobierajski 96] developed a technique for the haptic rendering of volume data, where the surface normals were obtained analytically. The method works by decomposing the object into a three-dimensional grid of voxels. Each voxel contains information such as density, stiffness, and viscosity. An interpolation function is used to produce a continuous scalar field for each property. They present one example of interacting with a set of dendrites emanating from a lateral geniculate nucleus cell. The data was obtained by scanning with a confocal microscope. The additional functionality was developed to enable the user to visualize and interact with the internal structure. [Bartz and Gürvit 00] used distance fields to enable the direct volume rendering of a segment of an arterial blood vessel derived from rotational angiography. The first distance field is generated by first computing a path from a given starting voxel to a specified target voxel in the blood vessel. This path is computed using Dijkstra's Algorithm. This creates a distance field that details the cost of traveling to the target voxel. A second field is based on the Euclidean distance between each voxel and the surface boundary. A repulsive force can then be rendered, based on the distance between the haptic probe and the surface. The effects of the two distance fields are controlled using constant coefficients. A local gradient at a point can then be obtained using trilinear interpolation of the surrounding voxels. The coefficients of the distance fields must be chosen carefully to avoid oscillations.

Several researchers have investigated cutting and deforming volumetric data representing anatomical structures [Agus et al. 02, Eriksson et al. 05, Kusumoto et al. 06, Petersik et al. 02, Gibson et al. 97]. [Gibson et al. 97] segmented a series of MRI images by hand for the simulation of arthroscopic knee surgery. The deformation of the model was calculated using an approach that permits a volume to stretch and contract in accordance to set distances [Gibson 97]. The physical properties of the material are also useful for sculpting material represented by volume data. [Chen and Sun 02] created a system for sculpting both synthetic volume data and data obtained from CT, MRI, and ultrasound sources. The direct haptic rendering approach utilized an intermediate representation of the volume data [Chen et al. 00]. The intermediate representation approach to haptic rendering was inspired from its use in rendering geometric models [Mark et al. 96]. The sculpting tools developed by Chen and Sun were treated as volumes, allowing each position in the tool volume to affect the object volume data. They simulated a variety of sculpting effects including melting, burning, peeling, and painting.

When interacting with the volume data directly, an approach is required to provide stiff and stable contacts in a similar fashion to the rendering achieved with geometric representations. This is not easily accomplished when using the techniques based on mapping volume data directly to forces and torques. One strategy is to use a proxy constrained by the volume data instead of utilizing an intermediate representation, as in the previous example [Ikits et al. 03, Lundin et al. 02, Palmerius 07]. [Lundin et al. 02] presented an approach aimed at creating natural haptic feedback from density data with solid content (CT scans). To update the movements of the proxy point, the vector between the proxy and the HIP was split into components: one along the gradient vector (f_n) and the other perpendicular to it (f_t). The proxy could then be moved in small increments along f_t. Material properties such as friction, viscosity, and surface penetratability could be controlled by varying how the proxy position was updated. [Palmerius 07] developed an efficient volume rendering technique to encompass a constraint-based approach with a numerical solver and importantly a fast analytical solver. The proxy position is updated by balancing the virtual coupler force, $\vec{f}$, against the sum of the forces from the constraints, $\vec{F}_i$. The constraints are represented by points, lines, and planes. The balancing is achieved by minimizing the residual term, $\vec{\varepsilon}$, in the following equation:

$$\vec{\varepsilon} = -\vec{f}(\vec{x}_{proxy}) + \sum_i \vec{F}_i(\vec{x}_{proxy}). \tag{15.4}$$

By modifying the effects of the constraints in the above equation, different modes of volume exploration can take place, such as surface-like feedback and 3D friction. Linear combinations of the constraint effects can be used to obtain the combined residual term. An analytical solver may then be used to balance the equation and hence find the position of the proxy. The analytical solver is attempted first for situations where the constraints are orthogonal, however, if this fails, a numerical solver is utilized. This combination of techniques is available in the open-source software titled Volume Haptics Toolkit (VHTK).

15.3 Surface Details: Smoothing, Friction, and Texture

Haptic simulation of surface details such as friction and texture significantly improves the realism of virtual worlds. For example, friction is

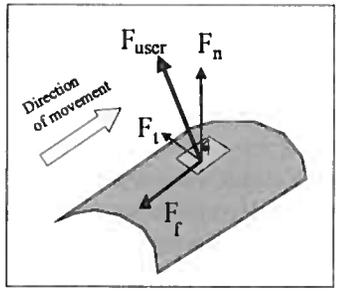


Figure 15.10. Forces acting on the user ($F_{user} = F_n + F_t + F_f$) during haptic simulation of friction and textures. The normal force can be computed using a simple physics-based model such as Hooke's law ($F_n = k\ \Delta x$, where Δx is the depth of penetration of the haptic probe into the virtual surface). To simulate Coulomb friction, we need to create a force ($F_f = \mu F_n$, where μ is the coefficient of friction) that is opposite to the direction of the movement. To simulate texture, we change the magnitude and direction of the normal vector (F_n) using the gradient of the texture field at the contact point.

almost impossible to avoid in real life, and virtual surfaces without *friction* feel "icy-smooth" when they are explored with a haptic device. Similarly, most surfaces in nature are covered with some type of *texture* that is sensed and distinguished quite well by our tactile system. Haptic texture is a combination of small-scale variations in surface geometry and its adhesive and frictional characteristics. Oftentimes, displaying the detailed geometry of textures is computationally too expensive. As an alternative, both friction and texture can be simulated by appropriate perturbations of the reaction force vector computed using nominal object geometry and material properties. The major difference between the friction and the texture simulation via a haptic device is that the friction model creates only forces tangential to the nominal surface in a direction opposite to the probe motion, while the texture model can generate both tangential and normal forces in any direction (see Figure 15.10).

15.3.1 Smoothing

A rapid change in surface normals associated with sharp edges between joining surfaces or between faces in a polygonal model causes force discontinuities that may prove problematic during haptic rendering. Incorporating techniques to blend between the surface normals can alleviate these problems. Without this type of technique, high numbers of polygons would be required to simulate surfaces with smooth curved areas. Strategies analogous to Gouraud or Phong shading, used for interpolating

normals for lighting, can be developed for haptic rendering. The paper by [Morgenbesser and Srinivasan 96] was the first to demonstrate the use of force shading for haptic rendering. Using a similar technique to Phong shading [Salisbury et al. 95] found that a smooth model could be perceived from a coarse three-dimensional model. This is akin to visualizing a smooth three-dimensional object using Phong shading when a relatively low number of triangles are used in the underlying geometry.

[Ruspini et al. 97] also incorporated a force shading model, which interpolated the normals similar to Phong shading. A two-pass technique was utilized to modify the position of the virtual proxy. The first stage computes the closest point, CP, between the HIP and a plane that runs through the previous virtual proxy position. The plane's normal is in the same direction as the interpolated normal. The second stage proceeds by using the CP as the position of the HIP in the usual haptic rendering algorithm described in the previous section. They state that the advantages of this method are that it deals with the issue of force shading multiple intersecting shaded surfaces, and that by modifying the position of the virtual proxy, the solution is more stable.

In some approaches, changes in contact information, penetration distance, and normals can affect the force feedback significantly between successive steps of the haptic update loop. These changes can cause large force discontinuities, producing undesirable force feedback. [Gregory et al. 00b] encountered this problem and employed a simple strategy to interpolate between two force normals. Their strategy prevents the difference between previous and current forces becoming larger than a pre-defined value, $F_{\max}$. This simple approach provides a means of stabilizing forces.

15.3.2 Friction

In Section 15.2 the methods for computing forces that act to restore the HIP to the surface of the virtual object have been discussed. If this force is the only one incorporated, then the result is a frictionless contact, where the sensation perceived is analogous to moving an ice cube along a glassy surface [Salisbury et al. 95]. However, this interaction is not very realistic in most cases, and can even hinder the interaction as the user slips off surfaces accidentally. Several approaches have been developed to simulate both static and dynamic friction to alleviate this problem [Salcudean and Vlaar 94,Salisbury et al. 95,Mark et al. 96,Ruspini et al. 97,Kim et al. 02a]. By changing the mean value of friction coefficient and its variation, more sophisticated frictional surfaces, such as periodic ones [Ho et al. 99], and various grades of sandpaper [Green and Salisbury 97], can be simulated as well.

[Salisbury et al. 95] developed a stick-slip friction model enhancing the feedback from their god-object approach. The model utilizes Coulomb friction and records a stiction point. The stiction point remains static until an offset between the stiction point and the user's position is exceeded. At this stage the stiction point is moved to a new location along a line that connects the previous stiction point and the user's position. [Kim et al. 02a] enabled friction to be incorporated with their implicit surface rendering technique. By adjusting the position of the contact point on the surface, a component of force tangential to the surface could be integrated. To achieve this, a vector, V, is obtained between the previous and new positions of the contact point on the surface. A friction coefficient can be integrated to determine a point, P, along V. The surface point that is intersected by a ray emanating from the HIP position passing through P is chosen as the new contact point.

At the University of North Carolina, Chapel Hill, [Mark et al. 96] developed a model for static and dynamic friction. The surfaces of the objects are populated by snags, which hold the position of the user until they push sufficiently to leave the snag. When the probe moves further than a certain distance from the center of the snag, the probe is released. While stuck in a snag, a force tangential to the surface pulls the user to the center of the snag, and when released, a friction force proportional to the normal force is applied. It is easily envisaged that this technique is appropriate for representing surface texture by varying the distribution of the snags. Surface texture will be described in the next section.

15.3.3 Texture

Perception and display of textures in virtual environments require a thorough investigation, primarily because the textures in nature come in various forms. Luckily, graphics texturing has been studied extensively, and we can draw from that experience to simulate haptic textures in virtual environments. There exists a strong correlation between the friction of a surface and its surface roughness, or texture. However, texture enriches the user's perception of a surface to a higher extent than friction, as extra details about the surface can be perceived. Integrating texture into haptic rendering algorithms presents more information to the user about the virtual object than applying images to the surface of objects for graphical rendering, using texture mapping. Surface texture is important when humans interact with objects, and therefore it is important for the haptic rendering of virtual objects. Many researchers have investigated the psychophysics of tactile texture perception. [Klatzky et al. 03] investigated haptic textures perceived through the bare finger and through a rigid probe. [Choi and Tan 04] investigated the perceived instabilities in haptic texture rendering

and concluded that the instabilities may come from many sources, including the traditional control instability of haptic interfaces, as well as inaccurate modeling of environment dynamics, and the difference in sensitivity to force and position changes of the human somatosensory system. [Minsky et al. 90] simulated the roughness of varying degrees of sandpaper. Users were then asked to order the pieces of simulated sandpaper according to their roughness. A texture depth map was created, utilized by the haptic device by pulling the user's hand into low regions and away from high regions.

A strategy similar to that of bump mapping objects, utilized in graphical rendering, was employed by [Ho et al. 99] to render haptic textures. Statistical approaches have been also used to generate haptic textures [Siira and Pai 96,Fritz and Barner 96a,Basdogan et al. 97]. [Fritz and Barner 96a] developed two methods for rendering stochastic-based haptic textures. The lattice texture approach works by constructing a 2D or 3D grid where a force is associated to each point. The second method, labeled *local space approach*, also uses a lattice defined in the texture space coordinate system. In this case the forces are determined for the centers of the grid cells. For implicit surfaces, [Kim et al. 02a] enabled Gaussian noise and texture patterns to directly alter the potential values stored in the points forming three-dimensional grids. The three-dimensional grids encompass the virtual objects. The adaptation could be incorporated without increasing the overall complexity of the haptic rendering algorithm. Fractals are also appropriate for modeling natural textures, since many objects seem to exhibit self-similarity. [Ho et al. 99] have used the fractal concept in combination with the other texturing functions, such as Fourier series and pink noise in various frequency and amplitude scales to generate more sophisticated surface details.

Recently, haptic texturing has also been employed between two polygonal models. This approach can be applied to the haptic rendering techniques for object-object interactions. [Otaduy et al. 04] developed a technique to estimate the penetration depth between two objects described by low resolution geometric representations and haptic textures created from images that encapsulate the surface properties.

15.4 Summary and Future

The goal of 3-DOF haptic rendering is to develop software algorithms that enable a user to touch, feel, and manipulate objects in virtual environments through a haptic interface. Three-DOF haptic rendering views the haptic cursor as a point in computing point-object interaction forces. However, this does not restrict us to simulate tool-object or multifinger interactions.

For example, a 3D tool interacting with a 3D object can be modeled as dense cloud of points around the contact region to simulate tool-object interactions. Many of the point-based rendering algorithms have been already incorporated into commercial software products such as the Reachin API,[2] GHOST SDK, and OpenHaptics.[3] Using these algorithms, real-time haptic display of shapes, textures, and friction of rigid and deformable objects has been achieved. Haptic rendering of dynamically moving rigid objects, and to a lesser extent, linear dynamics of deformable objects, have also been accomplished. Methods for recording and playing back haptic stimuli, as well as algorithms for haptic interactions between multiple users in shared virtual environments, are emerging.

In the future, the capabilities of haptic interface devices are expected to improve primarily in two ways: (1) improvements in both desktop and wearable interface devices in terms of factors such as inertia, friction, workspace volume, resolution, force range, and bandwidth; and (2) development of tactile displays to simulate direct contact with objects, including temperature patterns. These are expected to result in multifinger, multihand, and even whole body displays, with heterogeneous devices connected across networks. Even with the current rapid expansion of the capabilities of affordable computers, the needs of haptic rendering with more complex interface devices will continue to stretch computational resources. Currently, even with point-based rendering, the computational complexity of simulating the nonlinear dynamics of physical contact between an organ and a surgical tool, as well as surrounding tissues is very high (see the review in [Basdogan et al. 04]). Thus there will be continued demand for efficient algorithms, especially when the haptic display needs to be synchronized with the display of visual, auditory, and other modalities. Similar to graphics accelerator cards used today, it is quite likely that much of the repetitive computations will need to be done through specialized electronic hardware, perhaps through parallel processing. Given all the complexity and need for efficiency, in any given application, the central question will be how good does the simulation need to be to achieve a desired goal.

[2]http://www.reachin.se
[3]http://www.sensable.com

16 Six-Degree-of-Freedom Rendering of Rigid Environments

M. Ortega, S. Redon, and S. Coquillart

Chapter 15 describes three-degree-of-freedom methods for haptic display of the interaction of a point and a virtual object, such as the one introduced by Zilles and Salisbury [Zilles and Salisbury 95]. Three-DOF rendering methods are effective for single-point interaction, but designing similarly effective methods for object-object interaction becomes a remarkable challenge, due to the high computational requirements. The approach of Zilles and Salisbury for 3-DOF rendering presents two main benefits: (1) a non-penetrating simulation of the motion of the point as it slides on the surface of the obstacles; (2) a constraint-based computation of the force applied to the user, which results in a force orthogonal to the constraints. These features are highly desirable, in that non-interpenetration of virtual objects is known to increase their perceived stiffness [Srinivasan et al. 96], and that an incorrect orientation of the force has been shown to perturb the perceived orientation of the virtual surfaces [Sachtler et al. 00].

However, early 6-DOF haptic rendering methods [McNeely et al. 99, Johnson et al. 03, Gregory et al. 00b, Kim et al. 03, Otaduy and Lin 03b, Hasegawa and Sato 04, Constantinescu et al. 04, Wan and McNeely 03, Nelson et al. 99] do not preserve all of the properties of the initial 3-DOF approach introduced by Zilles and Salisbury [Zilles and Salisbury 95]: these methods might use penalty-based response (See Section 8.3) and allow the virtual objects to interpenetrate, or they use some form of virtual coupling [Colgate et al. 95] (see Section 8.3.2) which can lead to disturbing force artifacts by modifying the orientation of the force applied to the user. This chapter presents a 6-DOF constraint-based method that prevents both these visual and haptic artifacts. This method has three essential characteristics:

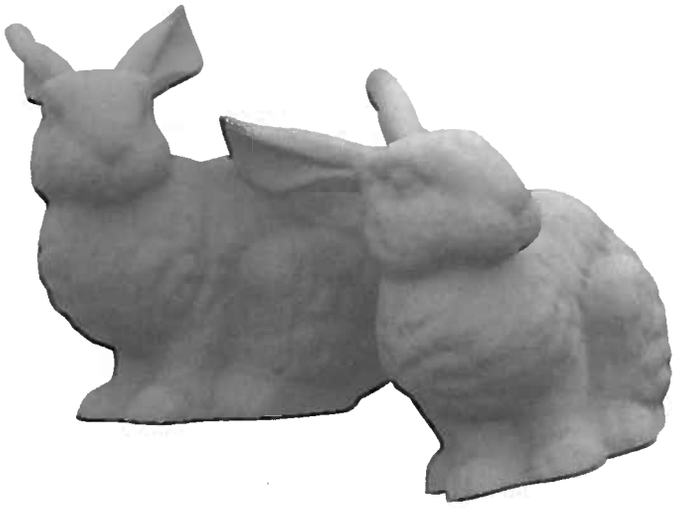


Figure 16.1. Haptic interaction with Stanford bunnies. The approach described in this chapter allows us to provide a user with high-quality haptic display of contacting rigid bodies (here, two Stanford bunnies containing about 27,000 triangles each). The constraint-based force computation method presented in this chapter allows the manipulated object to come in contact with and slide on the environment obstacles without penetrating them, while providing the user with precise haptic display, where each vertex, edge, and face can potentially be felt.

- *Six-degree-of-freedom god-object method.* The presented method is an extension of the three-degree-of-freedom god-object method proposed by Zilles andSalisbury [Zilles and Salisbury 95] to six-degree-of-freedom haptic interaction between rigid bodies.

- *High-quality god-object simulation.* The god-object simulation method prevents any interpenetration between the virtual objects, while allowing the god-object to precisely contact and slide on the surface of the obstacles. This results in highly detailed haptic rendering of the object's geometries and increases the perceived stiffness of the virtual objects [Srinivasan et al. 96].

- *Constraint-based force computation.* A novel constraint-based quasi-static approach is presented to compute the motion of the god-object and the force applied to the user. The constraint-based approach is physically based, handles any number of simultaneous contact points, and yields constraint forces that are orthogonal to the constraints, thereby rendering correct surface orientations to the user.

This chapter is organized as follows. After an overview of the constraint-based approach in Section 16.1, Section 16.2 describes how the motion of the god-object is computed, in order to ensure realistic haptic interaction with rigid bodies. Section 16.3 presents the constraint-based quasi-static approach to computing the force applied to the user. Section 16.4 discusses methods for producing haptic effects for surface perception, such as force shading and textures. Section 16.5 demonstrates the approach described in this chapter on several benchmarks and shows how it provides the user with high-quality haptic display of contacting rigid bodies. This section also discusses the benefits and limitations of the current approach. Finally, Section 16.6 concludes and details several future research directions.

16.1 Overview

The method described here extends the classical three-degree-of-freedom constraint-based method by Zilles and Salisbury [Zilles and Salisbury 95] by employing a *six-degree-of-freedom god-object*, i.e., an idealized representation of the haptic device that is constrained to remain on the surface of the environment obstacles when the haptic device penetrates the environment obstacles (see Figure 16.2). At each time step, the god-object simulation algorithm attempts to reduce the discrepancy between two rigid reference frames: one attached to the haptic device, and one attached to the virtual object. The origin of the virtual reference frame is positioned at the center of gravity of the virtual object, although any point can be chosen.

Only the god-object is displayed (and not the actual configuration of the haptic device), so that even when the haptic device penetrates the environment obstacles, the user only sees the rigid body that he or she manipulates

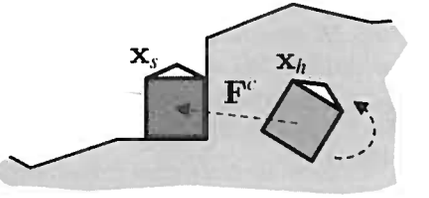


Figure 16.2. Six-degree-of-freedom god-object. Although the haptic device penetrates the environment obstacles (configuration $\mathbf{x}_h$), the god-object is constrained to remain on the surface of the obstacles (configuration $\mathbf{x}_s$). The algorithms presented in this chapter compute the motion of the god-object and the force applied to the user, based on the discrepancy between these two configurations.

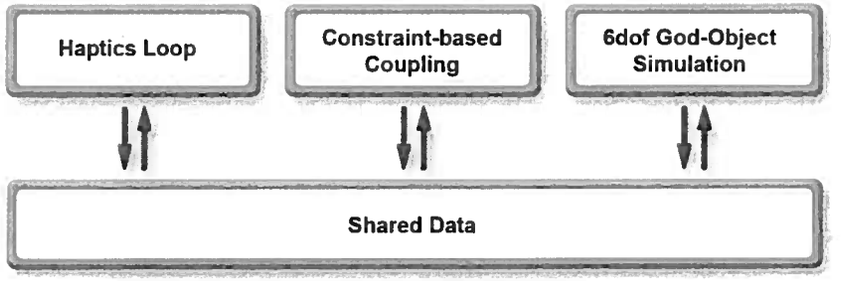


Figure 16.3. Schematic representation of the constraint-based approach. This haptic display method is divided in three asynchronous blocks.

in a realistic, contacting-only configuration. As a result, the user feels that the rigid body he or she is manipulating is correctly sliding on the surface of the obstacles. The motion of the god-object and the force applied to the user are computed from the discrepancy between the configurations of the god-object and the haptic device, thanks to a novel constraint-based quasi-static approach that suppresses visual and haptic artifacts typically found in previous approaches. The haptic rendering method is divided in three asynchronous loops: (1) the god-object simulation loop, which updates the configuration of the god-object based on the configuration of the haptic device and the environment obstacles; (2) the constraint-based coupling loop, which determines the constraint-based force applied to the user based on the configurations of the god-object and the haptic device, as well as the current set of contact points and normals; (3) the haptics loop, which controls an impedance-like haptic device that reads the force that has to be applied to the user and writes the current configuration of the haptic device (see Figure 16.3). The haptics loop is considered as a generic black box, and this chapter focuses on the two other processes, i.e., the god-object simulation loop and the constraint-based coupling loop.

16.2 Six-Degree-of-Freedom God-Object Simulation

16.2.1 Overview

The motion of the god-object is computed based on the relative configurations of the haptic device and the god-object, as well as the current set of contact points. Precisely, the quasi-statics of the god-object are simulated according to the following *god-object simulation algorithm*:

1. *Data retrieval.* The six-dimensional configuration $\mathbf{x}_h$ of the haptic device is retrieved from the shared data (see Figure 16.3).

2. *Unconstrained acceleration computation.* The unconstrained six-dimensional acceleration $\mathbf{a}^u$ of the god-object is computed from $\mathbf{x}_h$ and the six-dimensional configuration $\mathbf{x}_s$ of the god-object:

$$\mathbf{a}^u = k_s(\mathbf{x}_h - \mathbf{x}_s),$$

   where k_s is a coupling constant ($k_s = 0.5$ in our implementation). This is similar to the virtual coupling method [Colgate et al. 95], except that the coupling is performed on the acceleration of the god-object. Because the motion of the god-object is quasi-static, this amounts to directly control of the displacement of the god-object.

3. *Constraint-based quasi-static computations.* The constrained acceleration $\mathbf{a}^c$ of the god-object is computed based on the current contact information (i.e., the one resulting from the previous god-object simulation step) and the unconstrained acceleration $\mathbf{a}^u$. This involves forming the 6×6 god-object mass matrix $\mathbf{M}$ and the $6 \times m$ contact Jacobian $\mathbf{J}$, where m is the number of contact points (see details below).

4. *Collision detection.* The *target configuration* of the god-object is computed from its constrained acceleration using an explicit Euler integration step. The continuous collision detection algorithm introduced by Redon et al. [Redon et al. 02b] is used to detect collisions on a path interpolating the current and target god-object configurations. If the interpolating path is free of collisions, the god-object is placed in the target configuration. If a new contact occurs, however, the continuous collision detection algorithm determines the first contacting configuration along the interpolating path, as well as the new contact positions and normals. The configuration reached by the god-object at the end of this step is the *new god-object configuration.*

5. *Constraints transmission.* The matrices $\mathbf{M}$ and $\mathbf{J}$ corresponding to the new god-object configuration are written to the shared data, so that they can be retrieved by the constraint-based coupling loop to compute the constraint-based force applied to the user.

The god-object simulation loop ensures that the god-object attempts to reach the same configuration (position and orientation) as the haptic device. Continuous collision detection and constraint-based quasi-statics allow the god-object to slide on virtual obstacles without penetrating them as it tries to reach the haptic device. The following section describes how Gauss' least constraint principle is used to derive the constraint-based quasi-statics of the god-object.

16.2.2 Constraint-Based God-Object Quasi-Statics

Let $\mathbf{a} = (\mathbf{a}_G, \alpha)^T$ denote the generalized (six-dimensional) acceleration of the god-object, where $\mathbf{a}_G$ and α are respectively the linear acceleration and the angular acceleration of the god-object. The set of *possible* accelerations is easily determined from the contact positions and normals provided by the continuous collision detection algorithms. Let I_k and $\mathbf{n}_k$ respectively denote the position and normal of the k-th contact point, $1 \leqslant k \leqslant m$. Assuming the normal $\mathbf{n}_k$ is directed towards the exterior of the environment obstacle, the acceleration of the god-object must satisfy the following *non-penetration constraint* [Baraff 89]: $\mathbf{a}_G^T \mathbf{n}_k + \alpha^T (GI_k \times \mathbf{n}_k) \geqslant 0$, where GI_k is the vector from the center of inertia G of the god-object to the contact point I_k. Note the absence of a velocity-dependent term in the non-penetration constraint, as the quasi-static assumption implies that the velocity of the god-object is zero at all times. These m non-penetration constraints can be concatenated to form a single constraint on the generalized acceleration of the god-object: $\mathbf{Ja} \geqslant \mathbf{0}$, where $\mathbf{J}$ is a $m \times 6$ *Jacobian*.

Gauss' principle states that the constrained generalized acceleration $\mathbf{a}^c = (\mathbf{a}_G^c, \alpha^c)^T$ of the god-object minimizes the function [Gauss 29]

$$\mathcal{G}(\mathbf{a}) = \frac{1}{2}(\mathbf{a} - \mathbf{a}^u)^T \mathbf{M}(\mathbf{a} - \mathbf{a}^u) = \frac{1}{2}||\mathbf{a} - \mathbf{a}^u||_{\mathbf{M}}^2, \tag{16.1}$$

that is, the *kinetic distance* $||\mathbf{a}^c - \mathbf{a}^u||_{\mathbf{M}}$ between the constrained acceleration $\mathbf{a}^c$ and the unconstrained acceleration $\mathbf{a}^u$, over the set of possible accelerations $\{\mathbf{a} : \mathbf{Ja} \geqslant \mathbf{0}\}$. In other words, the constrained acceleration $\mathbf{a}^c$ is the (non-euclidean) projection of the unconstrained acceleration $\mathbf{a}^u$ onto the set of possible accelerations. This projection problem is solved using Wilhelmsen's projection algorithm [Wilhelmsen 76]. Note that the matrices $\mathbf{M}$ and $\mathbf{J}$ contain all the necessary and sufficient information to compute the constrained motion of the god-object.

16.3 Constraint-Based Force Computation

The constraint-based coupling loop determines the forces applied to the user based on the configuration of the haptic device and the contact information sent by the god-object simulation loop. Essentially, the constraint-based coupling loop performs the same constraint-based quasi-static computations as in the god-object simulation loop, but *assuming the configuration of the god-object is fixed.* This suppresses the need for collision detection in the constraint-based coupling loop and allows us to compute the constraint-based force applied to the user within a few *microseconds* (see Section 16.5). Precisely, the constraint-based force applied to the user

is computed according to the following *constraint-based force computation algorithm*:

1. *Data retrieval.* The configuration $\mathbf{x}_h$ of the haptic device and the configuration $\mathbf{x}_s$ of the god-object are read from the shared data, as well as the matrices $\mathbf{M}$ and $\mathbf{J}$, computed in the god-object simulation loop, which describe the local quasi-statics of the god-object.

2. *Unconstrained acceleration computation.* As in the god-object simulation loop, the unconstrained six-dimensional acceleration $\mathbf{a}^u$ of the god-object is computed from $\mathbf{x}_h$ and the six-dimensional configuration $\mathbf{x}_s$ of the god-object ($\mathbf{a}^u = k_s(\mathbf{x}_h - \mathbf{x}_s)$).

3. *Constraint-based force computation.* The constrained acceleration $\mathbf{a}^c$ of the god-object is computed from the unconstrained acceleration $\mathbf{a}^u$ and the matrices $\mathbf{M}$ and $\mathbf{J}$ retrieved from the shared data, by solving Gauss' projection problem. The constraint-based force to be applied to the user is then $\mathbf{F}^c = k_h \mathbf{M}(\mathbf{a}^c - \mathbf{a}^u)$, where k_h is a coupling constant.[1]

4. *Force transmission.* the constraint-based force $\mathbf{F}^c$ is written to the shared data. It will be read by the haptic loop, for application to the user.

Figure 16.4 demonstrates this algorithm in the case of a god-object in contact with an obstacle. For clarity, only two degrees of freedom are allowed: a vertical translation and a rotation whose axis is orthogonal to the plane of the figure. Figure 16.4(a) shows the god-object contacting the obstacle (in dark gray), and four successive configurations of the haptic device (in light gray), as well as the resulting unconstrained accelerations $\mathbf{a}_1^u$, ..., $\mathbf{a}_4^u$. Figure 16.4(b) shows the corresponding two-dimensional *motion-space*, i.e., the space of accelerations, and the linearized non-penetration constraint resulting from the contact point (the diagonal line). The possible accelerations are above this diagonal line. Projecting the unconstrained accelerations $\mathbf{a}_1^u, \ldots, \mathbf{a}_4^u$ on the set of possible accelerations yields the constrained accelerations $\mathbf{a}_1^c, \ldots, \mathbf{a}_4^c$, as well as the corresponding constraint forces $\mathbf{F}_1^c, \ldots, \mathbf{F}_4^c$ applied to the user. Haptic configurations 1 and 2 result in a force and a torque, which attempt to bring the haptic device back to a configuration reachable by the god-object, while haptic configurations 3 and 4, which correspond to accelerations satisfying the non-penetration constraint, do not generate any force.

[1] Different constants can be used for the translational and rotational parts, but this might lead to constraint forces that are not orthogonal to the non-penetration constraints (see Section 16.5).

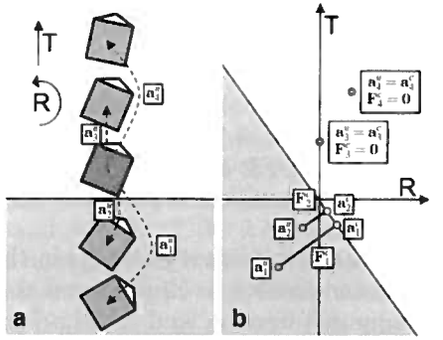


Figure 16.4. Constraint-based force computation. The method presented in this chapter uses Gauss' least constraints principle to compute the constrained motion of the god-object and the constraint-based force applied to the user (see Sections 16.2 and 16.3).

Note that because the configuration $\mathbf{x}_s$ of the god-object is not updated in the constraint-based coupling loop, the matrices $\mathbf{M}$ and $\mathbf{J}$ do not have to be updated either.[2] Hence, only the configuration of the haptic device changes, and the main computation involved is the determination of the constrained acceleration $\mathbf{a}^c$, which can be performed very efficiently (see Section 16.5).

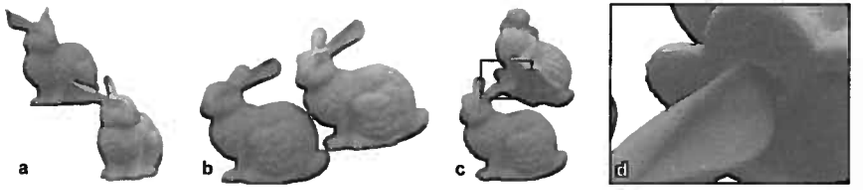


Figure 16.5. Haptic interaction with Stanford bunnies. The user manipulates the light gray bunny. (a) the ear of the light gray bunny slides in a ridge of the dark gray bunny. (b) continuous collision detection and constraint-based quasi-statics allows the manipulated object to precisely contact and slide on the obstacles. (c)–(d) the user can precisely feel the contact between pairs of triangles, resulting in highly detailed haptic display of contacting rigid bodies.

[2] In our implementation, a flag is used to signal the arrival of a new set of constraints to the constraint-based coupling loop. This flag, written to the shared data by the god-object simulation loop, allows us to avoid rereading the matrices $\mathbf{M}$, $\mathbf{J}$, and the god-object configuration $\mathbf{x}_s$, which further speeds up the constraint-based coupling loop.

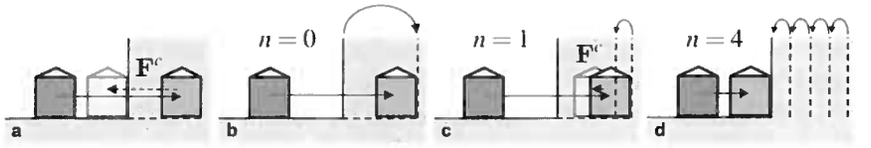


Figure 16.6. Constraints adaptation. When a new constraint (here a vertical plane) appears which would create too large a constraint force, it is first translated so that the constraint is satisfied by the current haptic device configuration, then progressively returned to its initial position. This helps us smooth the force felt by the user, while ensuring that small discontinuities signaling new contact points are felt.

When a new set of constraints is available, some of the new non-penetration constraints might not be satisfied by the current configuration of the haptic device (see Figure 16.6(a). This might create a large constraint force if the user has largely penetrated those new constraints. In order to smooth the constraint-based force applied to the user and reduce potentially large forces created by delays in the update of the set of constraints, a generalization of the method introduced by Mark et al. [Mark et al. 96] can be used. Assume a new constraint $\mathbf{J}_k\mathbf{a} \geqslant 0$ on the acceleration $\mathbf{a}$ of the god-object occurs, where $\mathbf{J}_k$ is a six-dimensional row vector (a row of the Jacobian). Assume that this constraint is not satisfied at time 0, when the new set of constraints becomes available, i.e., that the configuration of the haptic device is such that $\mathbf{J}_k\mathbf{a}^u = d_k < 0$. Initially, an *offset* is added to this constraint: the constraint becomes $\mathbf{J}_k\mathbf{a}^u \geqslant f_k(t)$, where f_k is a monotonously increasing time-dependent function such that $f_k(0) = d_k$ and $f_k(\Delta t) = 0$. This constraint is thus satisfied when the set of constraints is progressively turned into the constraint that should be enforced (i.e., after a time Δt; see Figure 16.6(b)–(d). In order 16.6(b)–(d). In order to provide the user with a slight force discontinuity and improve the perception of new constraints, however, this interpolation is performed only if $d_k \leqslant \varepsilon$, where ε acts as a user-defined discontinuity threshold ($\varepsilon < 0$).

The combination of the god-object simulation loop and the constraint-based coupling loop results in the perception of six-degree-of-freedom constraint forces as the user manipulates the virtual object and slides on the virtual obstacles.

16.4 Haptic Surface Properties

The six-degree-of-freedom constraint-based method presented here provides a force orthogonal to the non-penetration constraints. No force artifacts

are felt by the user, such as artificial friction or a sticking effect. The force vector direction can now be controlled and perturbed for providing haptic surface properties like force shading or texture. The two following sections demonstrate how such effects can be added, by modifying either the constraints or the force applied to the user.

16.4.1 Smooth Surfaces

Our current implementation uses a continuous collision detection method suitable for polygonal objects. As a result, smooth-shaped objects approximated by polygonal meshes feel like polyhedral surfaces, due to the discontinuity at the polygon edges. To avoid that, Morgenbesser and Srinivasan have been the first to adapt the well known Phong method [Phong 75] for smoothing polygonal meshes [Morgenbesser and Srinivasan 96].
They demonstrated that a similar haptic effect, called *force shading* and discussed in Section 15.3.1 in this book, can give the illusion of a haptically smooth shape. More recently, Ruspini et al. [Ruspini et al. 97] also proposed to adapt the graphical methods using the virtual proxy approach. Compared to the Morgenbesser approach, their force shading method allows them to handle situations involving multiple intersections between the proxy and shaded surfaces at the same time.

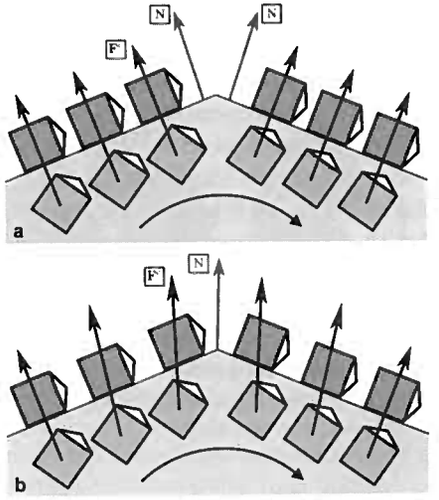


Figure 16.7. Smoothing effect. (a) The edge is felt because of the discontinuous force implied by the change in the normal direction. (b) Thanks to the use of the vertex normal **N**, the force shading method avoids the discontinuity, and smoothes the edge.

Like the Ruspini et al. approach, the constraint-based method proposed in this chapter allows to adapt the Phong method [Phong 75]. At each point on a mesh polygon, a new vector is computed by interpolating the normals from the vertices of the polygon. This new normal is used to compute the illumination of the model at this point. Consequently, the edges of the polygonal mesh do not appear, and the shape appears to be smooth. The same idea is used for force shading.

The following sections explain the link between the force vector direction and the surface normal, followed by the description of the force shading algorithm. Finally, they show how force shading can be efficiently computed in our asynchronous algorithm.

Surface normal and force rendered. As described in Sections 16.2 and 16.3, the computation of the force directly results from the computation of the constrained acceleration, which itself uses both the unconstrained acceleration and the contact information (or constraint space). The latter is mainly defined by the surface normal for each contact point between the god-object and the shape. Consequently, changing the surface normals in the contact information will change the direction of the force vector.

Basic algorithm. Using contact positions, similarly to the Phong approach, the algorithm proceeds by first computing the interpolated contact normals at each position of the contact points. These vectors are used to create a new constraint space, called the *force shading constraint space*. The rest of the algorithm consists of two computation passes (cf. Figure 16.8), i.e., the computation of the new direction of the force vector and the computation of the new god-object configuration.

- *Force vector direction.* First, a force-shading-constrained acceleration is computed from the unconstrained acceleration and the force shading constraint space. Next, the computation of the force is done with this new acceleration and the original unconstrained acceleration. At this point, and without the next stage, the force rendered by the haptic device will give the illusion of a non-flat mesh polygon, but the edges are still felt. The next stage explains how to avoid that.

- *Constrained acceleration.* As seen in Figure 16.7(a), with the six-degree-of-freedom god-object method, a discontinuity occurs when the user reaches an edge of the shape. Indeed, such an effect is provided by the computation of the constraint acceleration, which is always as close as possible to the unconstrained acceleration. Even with the computation of the perturbed force direction described in

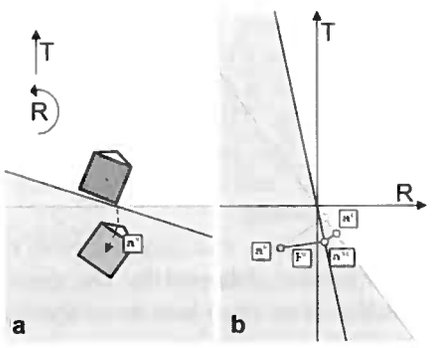


Figure 16.8. Two computation passes. The Gauss least constraints principle is used twice to compute the force-shading-constrained acceleration a^{sc} and the final constrained acceleration a^{c}. The first pass uses the force shading constraint space, while the second pass computes the motion of the god-object with the original constraint space and the force-shading-constrained acceleration as an unconstrained acceleration.

the stage before, this sudden change in the configuration of the god-object makes the user feel the edges of the polygonal mesh. To avoid that, the force shading constraint acceleration is used as an unconstrained acceleration and combined with the original constraint space to compute a final acceleration for the god-object (cf. Figure 16.8). Figure 16.7(b) shows the successive god-object configurations when such an approach is used.

Optimization with the asynchronous process. The computation described above is a time-consuming computation, because of the double constraint-based quasi-static computation. This can be optimized by exploiting the asynchronous aspect of the proposed algorithm, and by implementing one step in each process (i.e., the simulation and the coupling loops).

The force shading constraint space is created by the simulation loop and written to the shared data. In parallel, the coupling loop uses the last force shaded constraint space retrieved and computes the force-shading-constrained acceleration, which is also written in the shared data. Consequently, instead of the unconstrained acceleration, the force-shading-constrained acceleration is computed by the simulation loop using the original constraint space to create the new constraint acceleration of the god-object.

16.4.2 Textures

Chapter 18 describes some recent methods for six-degree-of-freedom haptic texture rendering [Otaduy et al. 04, Otaduy and Lin 04], but they are not

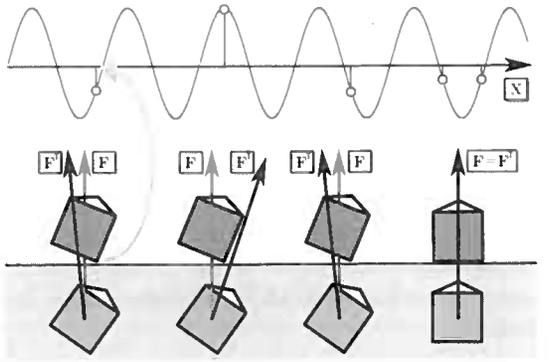


Figure 16.9. Bump and hole texture. The direction of the force vector $\mathbf{F}$ is perturbed by a sine function. The $\mathbf{x}$ position of the contact point is an entry of the sine function to find a value for perturbing the direction of the force. The perturbed force $\mathbf{F}^T$ is transmitted to the haptic device, allowing the user to feel the bumps and holes defined by this function.

applicable in conjunction with the constraint-based method described in this chapter. Earlier approaches, summarized in Section 15.3.3 in this book, proposed to explore textured surfaces in three degrees of freedom [Siira and Pai 96, Ho et al. 99, Pai et al. 01]. Minsky [Minsky 95] was the first to introduce a system to synthesize high-frequency textures for a haptic device. Only in 2D, they used a texture-map method. This approach is an adaptation of the bump-mapping graphical method proposed by Blinn [Blinn 78]. The approach combines the haptic device location and the map to provide a surface property and a force feedback. This produces a convincing effect of high-frequency textures.

A similar effect can be produced by perturbing the force computed by the six-degree-of-freedom constraint-based god-object approach, using a discrete or continuous function at the contact point position. For example, a sine function along one axis could be sufficient for providing bumps and holes along this axis (cf. Figure 16.9). In the case of multiple contact points, the perturbation vector used to modify the force vector direction is defined by averaging the perturbation vector at each contact point.

This method provides high-frequency textures and can be mixed with the force shading effect described above. However, similar to the Minsky et al. approach, if the speed of the god-object is too high, or the update rate of the simulation loop is too low, the contact point positions can pass from a hole directly to another one without feeling the bump in between. This implies a limitation in the texture frequency according to the exploration speed and the update rate of the simulation loop.

16.5 Results and Discussion

The validation of the presented approach is performed on a Stringed Haptic Workbench in which the SPIDAR-G, a tension-based six-degree-of-freedom force-feedback device [Kim et al. 02b], allows a user to interact intuitively on a large two-screen display [Tarrin et al. 03]. The entire algorithm is executed on a 3.2 GHz dual-processor Xeon PC, to which the haptic device is connected. This PC communicates with a cluster of PCs only dedicated to the stereo display on both screens of the Stringed Haptic Workbench. The communication between the Xeon PC and the cluster of PCs is ensured by UDP protocols.

Each of the three main loops is launched in its separate thread. The haptic device thread frequency is fixed by the device: the constraint-based force computed by the constraint-based coupling loop is read from the shared data and applied to the user at 1000 Hz. The frequencies of the constraint-based coupling thread and the god-object simulation thread vary over time, depending on the complexity of the models and the task being performed (see below).

16.5.1 Peg-in-a-Hole Benchmark

First, the quality and the stability of the haptic interaction is evaluated in a simple but classical case: the peg-in-a-hole benchmark (see Figure 16.10). This benchmark is well known because, although it involves only very simple geometry (here, 288 triangles for the peg and 280 triangles for the box), it has typically been a challenge to provide a stable and realistic haptic display of the insertion of the peg, due to the multiple and potentially redundant contact points occurring during the task [Gregory et al. 00b].

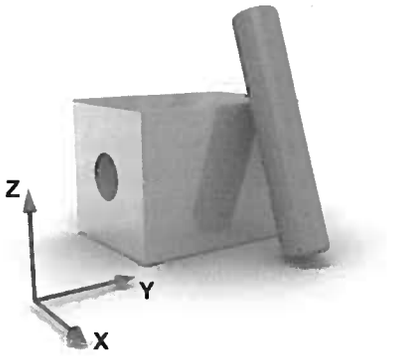


Figure 16.10. The models used in the peg-in-a-hole benchmark. The peg contains 288 triangles, while the hole contains 280 triangles. The hole is aligned with the y-axis.

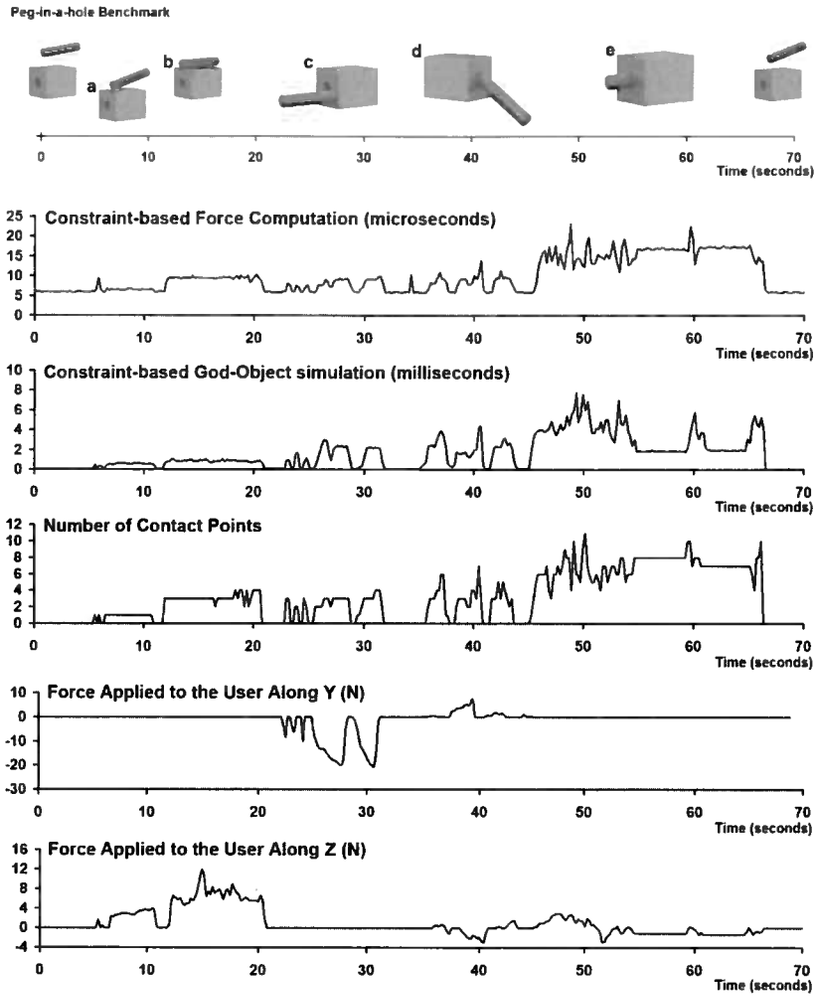


Figure 16.11. Performance of our approach in the peg-in-a-hole benchmark. The method computes a constraint-based force within a few *microseconds*, while a peg configuration update requires only a few milliseconds, which is sufficient to prevent visual lag in the simulation.

Figure 16.11 reports several timings and statistics measured during a typical interaction. The first row reports several key configurations tested during the interaction, including (a) sliding the tip of the peg on the top

side of the box, (b) laying the peg on the top side of the box and sliding it on the box, (c) pushing on the left side of the box, (d) exploring the right extremity of the hole, and (e) inserting the peg in the hole. The second row reports the time required to compute the constraint-based force (see Section 16.3) during the interaction. It can be seen that the constraint-based force is computed in less than 25 *microseconds* throughout the manipulation. The third row shows that the time required to update the configuration of the god-object is always smaller than 10 milliseconds, which is sufficient to prevent any visual lag throughout the manipulation. The fourth row reports the number of simultaneous contact points during the interaction, which can be seen to be fairly limited throughout the manipulation. This can be easily explained by the fact that (a) new contact points rarely occur exactly simultaneously, and (b) compared to other approaches using the interpenetration between virtual objects, constraint-based quasi-static computations tend to limit the apparition of new contact points, since at most 12 of them can be independent (each constraint removes half a degree of freedom). This greatly contributes to the efficiency of the constraint-based coupling loop. Finally, the fifth and the sixth rows report the Y and Z components of the constraint-based force applied to the user during the interaction. As expected, the Y component is non-zero only when the user pushes the peg on the left side of the box or explores the right extremity of the hole (steps (c) and (d)), and remains equal to zero whenever the peg is sliding on the top side of the box or inside the hole. The Z component has high values when the user pushes the peg on the top side of the box and has little variations when the peg is inside the hole, due to user movement precision. In other words, the user does not feel any artificial friction force or any artificial sticking during the manipulation (e.g., the Y component of the force is never positive during step (c)).

Overall, the combination of continuous collision detection, constraint-based quasi-statics, and constraint-based force computation makes it very easy for the user to accomplish the task, by allowing the peg to slide on the surface of the box and the hole, while providing the user with a high-quality haptic display.

16.5.2 Stanford Bunnies Benchmark

The second benchmark involves two Stanford bunnies (27,000 triangles per bunny, see Figure 16.12). One bunny is static, and the second bunny is manipulated by the user. Figure 16.5 shows several key steps of the interaction: Figure 16.5(a) shows the ear of the mobile bunny sliding in a ridge of the static bunny; Figure 16.5(b) demonstrates how the constraint-based god-object simulation provides realistic contacting configurations during the interaction; similarly, Figure 16.5(c)–(d) show how our approach is able to

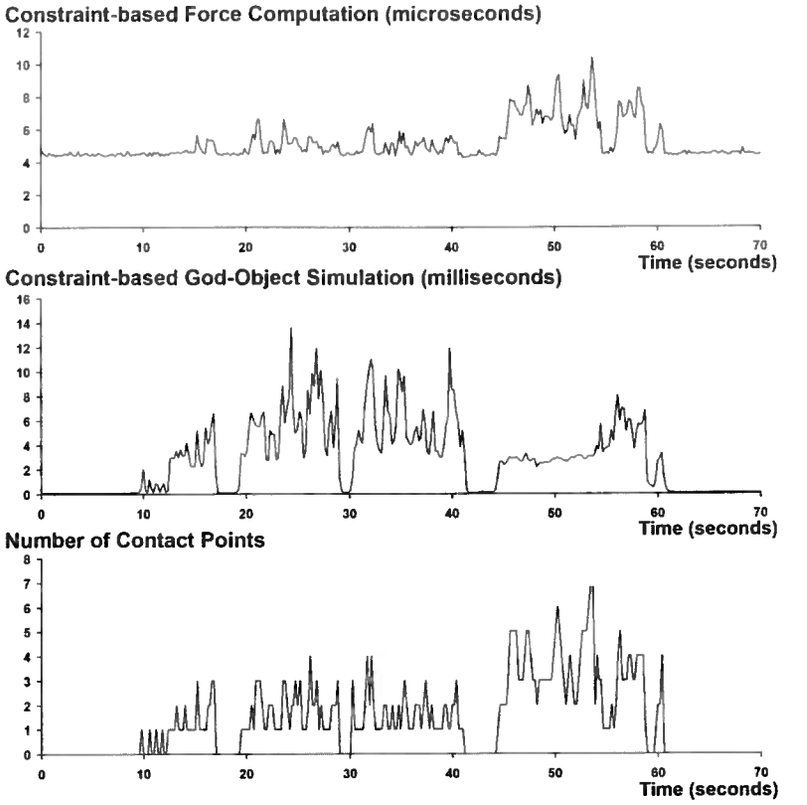


Figure 16.12. Performance of the approach in the Stanford Bunny benchmark. Even in this complex benchmark (27,000 triangles per bunny), the presented method is able to compute a constraint-based force within a few microseconds. The simulation of the god-object, which includes collision detection and constraint-based quasi-statics computations, is performed in less than 15 milliseconds, which is sufficient to prevent visual lag during the interaction.

provide the user with high-quality haptic display of contacting rigid bodies, where the details of the geometry can be felt by the user.

Figure 16.12 reports on the performance of the approach during a typical interaction session with the bunnies, which includes the configurations represented in Figure 16.5. Again, the force applied to the user is computed within a few microseconds, while an update of the configuration of the mobile bunny, which includes continuous collision detection and constraint-based quasi-statics, is performed within a few milliseconds, resulting in the absence of any visual lag during the interaction.

16.5.3 Discussion

Benefits. The main benefits of the presented approach stem from the combination of three key elements:

- *Continuous collision detection* allows the user to feel the details of the geometry of the rigid bodies and potentially feel the contact between vertices, edges, and faces of the contacting objects. Furthermore, the ability to produce visually convincing non-penetrating, but tangent contacting configurations (e.g., Figure 16.5(b)) helps improve the perceived stiffness of the objects [Srinivasan et al. 96].

- *Asynchronous updates* of the configuration of the god-object and the force applied to the user help satisfy the different update rates required by the haptic and the visual displays.

- *Constraint-based quasi-statics* allows the user to slide on the environment obstacles, and haptically feel the reduced motion sub-space resulting from the simultaneous non-penetration constraints, thus providing the user with a realistic haptic display of surfaces, corners, ridges, and object/object contact in general.

Especially, the physically-based computation of the force applied to the user guarantees that no *artificial friction or sticking is felt*, and that no force is applied when the god-object is in free space. This is to be contrasted to what would occur if some kind of virtual coupling were involved in the computation of the force applied to the user. Figure 16.13 shows such a comparison, in which the god-object (dark gray) is constrained to remain above the surface of the obstacle. In the case depicted in Figure 16.13(a), where the haptic device (light gray) has penetrated the environment, a virtual coupling would attempt to bring the haptic device back to the configuration of the god-object, which would result in an artificial tangential friction applied to the user.

As mentioned before, this would degrade the perceived orientation of the surface of the obstacle [Sachtler et al. 00]. In contrast, the constraint-based approach guarantees that the perceived orientation is correct, since the contact forces are always orthogonal to the constraints.[3] Furthermore, in the case depicted in Figure 16.13(b), where the user moves away from the obstacle, a virtual coupling would attempt to bring the god-object back to the surface of the obstacle, which would result in a sticky feeling. In this case, however, the constraint-based approach yields the correct force

[3]Since the constrained acceleration of the god-object $\mathbf{a}^c$ minimizes the kinetic distance $||\mathbf{a}^c - \mathbf{a}^u||_\mathbf{M}$ to the unconstrained acceleration $\mathbf{a}^u$ among the possible accelerations, it is such that $(\mathbf{a}^c - \mathbf{a}^u)^T\mathbf{M}\mathbf{a}^c = 0$, which implies that $(\mathbf{F}^c)^T\mathbf{a}^c = 0$.

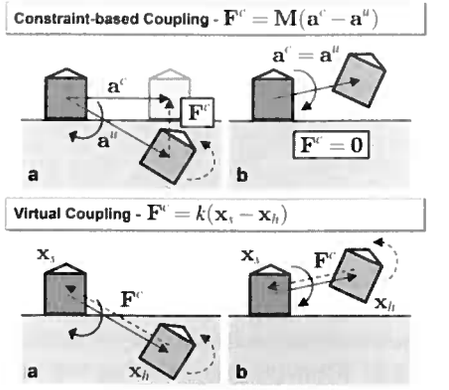


Figure 16.13. Benefits of the constraint-based approach. The constraint-based approach introduced in this chapter allows to remove force artifacts typically found in previous methods (see Section 16.5).

($\mathbf{F}^c = \mathbf{0}$), since moving away from the obstacle's surface satisfies the non-penetration constraint (hence, $\mathbf{a}^c = \mathbf{a}^u$).

Finally, it can be shown that the simulation of the god-object is purely dissipative, i.e. that the force $\mathbf{F}^u = \mathbf{M}\mathbf{a}^u$ applied to the god-object is such that

$$(\mathbf{F}^u)^T\mathbf{a}^c \leqslant (\mathbf{F}^u)^T\mathbf{a}^u.$$

Thus, the non-penetration constraints can only dissipate the energy transmitted to the god-object.[4] The tests have shown that the user is able to e.g., release the handle of the haptic device while the peg is inside the hole (cf. Figure 16.11, step (e)).

Limitations. The approach has two main limitations:

- *Linearized constraints.* In order to efficiently compute the quasi-statics of the god-object and the constraint-based force applied to the user, the non-penetration constraints are linearized. This might reduce the quality of the force applied to the user when a large discrepancy between the configurations of the god-object and the haptic device occurs. It would be interesting to investigate some more sophisticated force computation methods to address this problem, in-

[4]The proof is straightforward. Indeed, $(\mathbf{F}^u)^T(\mathbf{a}^c - \mathbf{a}^u) = (\mathbf{a}^u)^T\mathbf{M}(\mathbf{a}^c - \mathbf{a}^u) = -||\mathbf{a}^c - \mathbf{a}^u||^2_\mathbf{M} + (\mathbf{a}^c)^T\mathbf{M}(\mathbf{a}^c - \mathbf{a}^u)$. Since $(\mathbf{a}^c - \mathbf{a}^u)^T\mathbf{M}\mathbf{a}^c = 0$ (see Footnote 3), $(\mathbf{F}^u)^T\mathbf{a}^c \leqslant (\mathbf{F}^u)^T\mathbf{a}^u$. Note that the product of the force and the acceleration is used because the approach deals with the quasi-static case. This is the equivalent of the product of the force and the velocity used in typical analyses.

volving, for example, an implicit formulation of the non-penetration constraints.

- *Potentially low update rate of the set of constraints.* There is no guarantee that the approach is able to update the set of non-penetration constraints at 1000 Hz. This might lead to missing some high-frequency details when the user slides rapidly on the surface of the environment obstacles.

The potentially low update rate of the set of constraints is the main reason for the separation of the god-object simulation and the constraint-based force computation into asynchronous processes, in this approach and several previous ones (e.g., [Constantinescu et al. 04, Mark et al. 96]). Because the complexity of any collision detection method that reports all the contacting features is output dependent, however, it seems arguable that whichever collision detection method is used, it will always be possible to find a scenario such that the time required to determine all the contact points will take more than one millisecond. It was preferred to rely on a god-object simulation method that offers precise interaction with rigid bodies, and, especially, precisely contacting configurations. Although this might limit the rate at which the set of non-penetration constraints is updated (sometimes as low as 70 Hz in the Stanford bunnies benchmark, and about 300 Hz on average), this approach allows us to compute a constraint-based force consistent with the current set of simultaneous constraints at extremely high rates (always higher than 80,000 Hz in the Stanford bunnies benchmark). Furthermore, it should be emphasized that the constraint-based computations performed in the constraint-based coupling loop *implicitly include some collision detection.* Returning to the example depicted in Figure 16.4, it can be seen that if, between two updates of the set of non-penetration constraints, the haptic device switches from a state where all currently known non-penetrating constraints are satisfied (in which case $\mathbf{F}^c = \mathbf{0}$) to one where at least one of the currently known non-penetrating constraint is not satisfied (in which case $\mathbf{F}^c \neq \mathbf{0}$), the user will feel this collision. In summary, collision detection is implicitly performed in the constraint-based coupling loop, for the current set of non-penetration constraints, at extremely high rates.

16.6 Summary

This haptic rendering method described in this chapter generalizes the classical three-degree-of-freedom god-object method, introduced by Zilles and Salisbury [Zilles and Salisbury 95], to six-degree-of-freedom haptic display of contacting rigid bodies. With the current approach, a rigid god-object

is able to contact and slide on the environment obstacles without penetrating them, and the forces applied to the user are orthogonal to the non-penetration constraints (in the kinetic norm sense). The proposed approach has been successfully tested on the classically difficult peg-in-a-hole benchmark and on some more complex models—two Stanford bunnies with 27,000 triangles each. It has been shown that the presented method is able to provide a high-quality haptic display of contacting rigid bodies in both cases with basic surface properties (e.g., textures and force shading). The constraint-based approach ensures that no force artifacts are felt by the user.

The approach presented here could be extended in several directions. One possibility would be to extend the approach to multiple dynamic objects (although it can be argued that quasi-static interaction is preferable for the simulation of many tasks, as few manipulation tasks seem to require using the inertia of the manipulated object to accomplish the task). One possible direction to do this could be to generalize the approach suggested by Niemayer and Mitra [Niemeyer and Mitra 04] to six-degree-of-freedom haptic interaction. Finally, actual industrial scenarios such as virtual prototyping and assembly tasks could be investigated.

Acknowledgments

The authors would like to express their profound appreciation for the support and feedback from the PSA Peugeot Citroën representatives involved in the project. They also wish to thank Dr. Ming C. Lin and Dr. Miguel A. Otaduy for insightful discussions, and Stanford University for the original bunny models.

This work was partially supported by PERF-RV2 and by the INTUITION European Network of Excellence (IST NMP-1-507248-2).

17
Rendering of Spline Models

D. E. Johnson and E. Cohen

While faceted models are in widespread use, for example in games, other applications, such as computer-aided design (CAD), computer-aided manufacture (CAM), and higher-end animation require more exact model representations. A *de facto* standard in these areas are spline models, which use higher-degree, rational, parametric surfaces to represent shape. This chapter will provide some introduction to spline basics and show how to apply this theory for the demanding computational task of haptic rendering. A representative model is shown in Figure 17.1.

17.1 The Spline Representation

Splines are a subset of parametric equations. The line segment interpolating between two points P_1 and P_2 is easily described in parametric form as

$$L(t) = (1-t)P_1 + tP_2, 0 \leq t \leq 1. \tag{17.1}$$

This is also a linear spline. In spline terminology, the points P_1 and P_2 are *control points* and the functions that weight the points are the *basis*

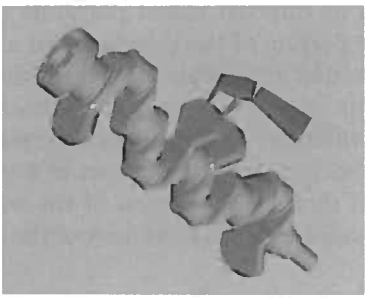


Figure 17.1. This CAD model of a crank consists of multiple NURBS surfaces joined by piecewise linear trimming loops [Thompson II and Cohen 99]. (© 1999 ASME)

functions. The domain of the function is set by the *knot vector*. Of course, splines are much more general than line segments; they describe piecewise polynomial curves and surfaces of controllable continuity. In CAD, rational splines are popular because they can precisely represent conic sections, such as arcs. This rational representation is known as *non-uniform rational B-spline (NURBS)*.

17.1.1 NURBS models

Non-uniform rational B-spline (NURBS) surfaces are highly compact and yet very expressive representations for modeling. A NURBS surface is a bivariate vector-valued piecewise rational function of the form

$$S(u,v) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} P_{i,j} w_{i,j} B_{j,k_v}(v) B_{i,k_u}(u)}{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{i,j} B_{j,k_v}(v) B_{i,k_u}(u)}, \tag{17.2}$$

where the $\{P_{i,j}\}$ form the set of control points known as the *control mesh*, the $\{w_{i,j}\}$ are the weights, and the $\{B_{i,k_u}\}$ and $\{B_{j,k_v}\}$ are the basis functions defined on the knot vectors $\{u\}$ and $\{v\}$ for a surface of order k_u in the u direction and k_v in the v direction.

The various properties of a NURBS surface, including a local convex hull property, and the ability to evaluate surface points, normals and tangents, along with its intuitive control characteristics, make it a good representation for modeling and design. These properties have led to NURBS becoming a *de facto* industry standard for the representation and data exchange of geometric models.

Trimmed NURBS models are constructed by cutting away portions of a NURBS surface, using trimming curves in parametric space. These trimming curves require their own representation. One approach is to store trimming information as directed closed polygons called *trimming loops*. Each individual linear portion of the loop is called a *segment*. A collection of connected segments that represents shared boundary between two surfaces is referred to as an *edge*. Portions of the surface domain to the left of a loop are considered cutaway, while pieces to the right are deemed part of the model. Note that each surface that is part of a model contains at least one trimming loop. If there is no portion of the surface being cut away, then this loop simply surrounds the boundary of the domain of the surface.

17.2 Distance and Orthogonal Projection

The previous chapter discussed computing forces by finding the distance between the haptic interface point (HIP) and the constrained proxy point

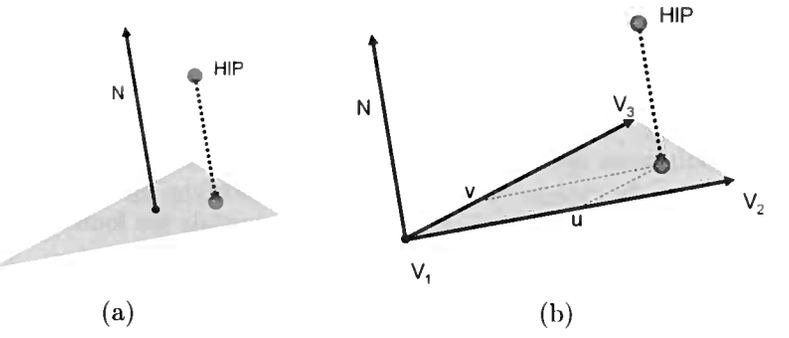


Figure 17.2. (a) The closest point on a triangle is found by projecting the haptic interface point down along the triangle normal. (b) There is an equivalent parametric representation of the surface.

on the surface. At the level of the triangle primitives, the closest point on the triangle face is found by projecting the haptic interface point along the triangle normal onto the surface. This operation is seen in Figure 17.2. While this operation may make intuitive sense, it is worth looking at the underlying mathematics in more detail, so that this process can be adapted for spline models.

The triangle lies on an infinite plane sharing the same normal, N, and this plane will be used in the following discussion. Instead of using a geometric projection operation, the closest point finding operation can be expressed in a different form. A plane can be defined as going through a set of vertices V_1, V_2, V_3, which create an internal coordinate system with axis vectors $(V_2 - V_1, V_3 - V_1, N)$. This system then defines the plane in parametric form, $T(u,v)$, where

$$T(u,v) = V_1 + u(V_2 - V_1) + v(V_3 - V_1). \tag{17.3}$$

Figure 17.2(b) shows this parametric setup. The distance, D, between the HIP and every point on the plane is then

$$D(u,v) = ||P_{HIP} - T(u,v)||. \tag{17.4}$$

The closest point on the plane is the minimum of Equation (17.4). Minima, and really all extrema, occur at common zeroes of the partial derivative of an equation. Since the distance, as expressed above, involves finding vector magnitude with a square root, a common trick is to use the squared distance instead. The squared distance shares roots with the Euclidean

distance and has a simplified system of partial derivatives. Therefore,

$$D^2(u,v) = ||P_{HIP} - T(u,v)||^2 \quad (17.5)$$
$$= (P_{HIP} - T(u,v)) \cdot (P_{HIP} - T(u,v)). \quad (17.6)$$

The minimum distance occurs at simultaneous zeroes of $\mathbf{F}$, the system of partial derivatives of $D^2(u,v)$. In the following equation, partials are denoted by a subscripted parameter. The partials are found by using the chain rule on $D^2(u,v)$.

$$\mathbf{F} = \begin{bmatrix} D_u^2(u,v) \\ D_v^2(u,v) \end{bmatrix} = \begin{bmatrix} 2(P_{HIP} - T(u,v)) \cdot -T_u(u,v) \\ 2(P_{HIP} - T(u,v)) \cdot -T_v(u,v) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \quad (17.7)$$

This probably does not seem like a very natural way to find the distance to a plane, and it would be computationally inefficient to use it directly. However, it does provide justification for the geometric projection operation used earlier. Looking at the system of partials, F, it describes the conditions that need to be met where there is a minimum in distance. The condition is that the vector between the HIP, P_{HIP}, and the proposed solution point on the plane, $T(u,v)$, must be orthogonal to both the surface tangents at $T(u,v)$. This is because the dot product between that vector and each tangent must equal zero for F to be a root. An equivalent way of stating these constraints is that the vector between the HIP and the proposed solution point must be parallel to the normal, since the normal is orthogonal to both surface tangents. Therefore, the geometric "project along the normal" concept comes directly from trying to minimize the distance between a point and a plane.

Haptic rendering of spline models works directly with the parametric form of the distance equation. The complexity of haptic rendering algorithms then depends on finding the appropriate type of numeric or symbolic solver to update these closest points fast enough, and reliably enough, for use with a haptic interface.

17.3 Local Minima in Distance versus the Virtual Proxy

Recall that simple application of finding the closest point on a model's surface to the HIP is not enough for realistic haptic rendering. For polygonal models, the concept of the virtual proxy is used to track the HIP's history, and to prevent unpleasant artifacts such as being accelerated through thinner models and sharp changes in forces. In essence, this additional state information and the application of constrained minimization is used to maintain a local minima in distance for computing the restoring force.

Such a local minima is a natural result of applying numerical methods to the parametric distance equations. In a typical numerical root finder, an initial guess must be used to start the method. For systems with a state that evolves over time, a simple approach uses the result from the previous time step to initialize the solution for the current time step. This is exactly analogous to the virtual proxy used on polygoal models.

17.4 3-DOF Haptic Rendering of Spline Models

Haptic rendering of spline models shares similar approaches to that for polygonal models. There is unconstrained motion in free space, estimates of potential contact locations through distance measures, transitioning from free motion to penetration into a model with concurrent force generation, continued motion with updated forces, and transitioning back into free motion.

The earliest haptic rendering of spline models was heavily constrained by available computation power. One approach out of the Ford Motor company [Stewart et al. 97] mirrored early polygonal approaches by slowly updating an *intermediate plane* [Adachi 93], which was used to compute an aproximate distance to the model. Another approach, Direct Parametric Tracing (DPT), by Thompson et al. [Thompson et al. 97] directly used the spline surface, but linearized the distance update during motion of the HIP to increase speed. This approximation was necessitated by the embedded processors used to compute forces in their haptic system. The DPT approach does illustrate many of the operations needed for successful haptic rendering of spline models and is detailed in the following section.

17.5 Direct Parametric Tracing

The DPT method used an approximation known as *nodal mapping* [Snyder 95] to find a first-order approximation to the closest point on the surface (Figure 17.3). The HIP is projected onto the control mesh of the NURBS surface, resulting in a point Q. Each vertex of the control mesh has an associated (u,v) parametric value that is called the *node* [Cohen and Schumaker 85]. An approximate (u,v) for Q is determined by interpolating between node values, using the barycentric coordinates of Q. The surface is evaluated at the interpolated (u,v) point and the distance between $S(u,v)$ and E is used as the surface proximity distance. With the additional compute power of modern machines, root polishing with numerical methods would be advisable and would yield a more accurate result. Nodel

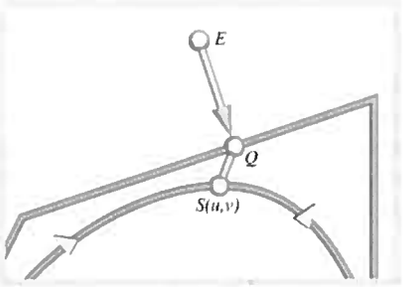


Figure 17.3. The projected distance along the control polygon is used as the parametric distance between associated nodes [Thompson et al. 97]. (© 1997 ACM)

mapping can then be thought of as a heuristic to find a starting point for the numerical method.

17.5.1 Tracking Phase

When a surface becomes close enough to the HIP, the approximate closest point from nodal mapping initializes the DPT local closest point tracking method (Figure 17.4). Following the derivation in [Thompson et al. 97], the DPT method is shown for a B-spline curve, rather than for a surface. Some definitions used are the previous point on the curve, $\gamma(u)$: the tangent vector at $\gamma(u)$, $\gamma'(u)$; and the current HIP location, E. These elements determine a new approximate closest point on the curve.

The basic idea is to linearly approximate motion along $\gamma'(u)$ to a change in parameter along the curve. At the limit, $\gamma'(u)$ relates changes in position along the curve in Euclidean space to changes in position in parametric space:

$$\gamma'(u) = \frac{d\gamma}{du} \approx \frac{\Delta\gamma}{\Delta u}. \tag{17.8}$$

Given a Euclidean movement along γ, the corresponding movement in the parametric space of the curve is calculated as

$$|\Delta u| \approx \frac{\|\Delta\gamma\|}{\|\gamma'(u)\|}. \tag{17.9}$$

In order to use Equation (17.9) as a closest point tracking method, movement of the end-effector needs to be related to movement of the closest point on the curve. The exact $\Delta\gamma$, corresponding to movement of the closest point along the curve, clearly involves finding the desired new closest point. Instead of finding an exact $\Delta\gamma$, a first-order Taylor series approximation to the curve, the tangent $\gamma'(u)$ is used to compute an approximate $\Delta\gamma$.

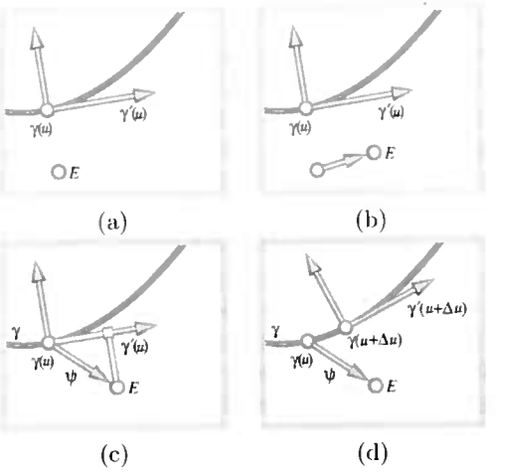


Figure 17.4. (a) Initial state. (b) HIP moves. (c) Projection of HIP position onto surface tangent plane. (d) New surface point and tangent plane found via parametric projection [Thompson et al. 97]. (© 1997 ACM)

The movement of the end-effector can now be related to movement of the closest point along the curve by projecting the offset vector, ψ, formed by subtracting $\gamma(u)$ from E, onto the curve tangent vector (Figure 17.4(c)). Thus,

$$\Delta\gamma \approx \frac{\langle\, \psi\,,\, \gamma'(u)\,\rangle}{\|\gamma'(u)\|^2}\,\gamma'(u). \tag{17.10}$$

Fortunately, these elements are all efficiently computable on B-spline curves through the curve evaluation done at the previous time step. That curve refinement yields new control polygon points P_i and P_{i-1}, which are the curve evaluated at the previous closest point parameter, and a point along the tangent vector, respectively. The final result,

$$\Delta u \approx \frac{\langle\, \psi\,,\, (P_{i^*+1} - P_{i^*})\,\rangle}{\|P_{i^*+1} - P_{i^*}\|^2}\left(\frac{u_{i^*+k} - u_{i^*+1}}{k-1}\right), \tag{17.11}$$

shows that the change in parameter as the HIP moves can be found with very few arithmetic operations. On more modern machines, such an update can be computed several hundred thousand times per second.

The new curve location, $\gamma(u^* + \Delta u)$, is a good approximation to the closest point to E. The new closest point is evaluated through multiple knot insertions at $u^* + \Delta u$, which maintains the conditions needed to use Equation (17.11) at the next time step (Figure 17.4(d)).

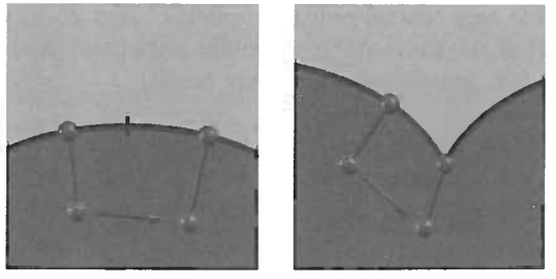
Essentially, we make a first order approximation of the closest point movement in Euclidean space with the tangent projection. The closest point movement is converted into parametric movement through a first order approximation to the parametric velocity at the previous closest point. The new closest point is then converted back into Euclidean space through curve refinement and evaluation. For small step sizes and penetration depths, this provides an excellent approximation.

For surfaces, the method is essentially the same, although the projection step now requires projection onto the tangent plane, $S'(u, v)$, of the surface. Barycentric coordinates are used to derive Δu and Δv. In the original DPT paper, the DPT method used to trace a single surface ran at 1400 Hz on a Motorola 68040 processor, barely fast enough for haptic rates, but significantly faster than more sophisticated iterative numerical methods.

17.5.2 Surface Transitions

The closest point update equations are only valid for single surfaces. Realistic models are formed out of multiple surfaces connected at their parametric boundaries, or by trimming curves, so updates to the local closest point must be able to transition over surface boundaries onto the new surface. In practice, trimming curves are used to join surfaces at their parametric boundaries, as well at the interior of the domain, so the trimming curve case is the only one that must be considered.

Trimming curves complicate closest point tracking in the following ways. In the parametric domain, trims remove portions of the domain. Thus, for each update of the closest point, the tracking algorithm needs to check if the updated point is still within the valid domain. Update steps that cross a trim boundary can be thought of as moving onto a new surface, so the



(a) (b)

Figure 17.5. (a) Transitioning across a trimming edge and onto another surface. (b) Transitioning onto the intersection of two surfaces [Thompson II and Cohen 99]. (© 1999 ASME)

parameter value of the closest point needs to be converted from one surface to the new surface. Additionally, trims can form C^1 discontinuities on the model. The image of the trim curve forms a curve in Euclidean space along a sharp boundary between adjoinng surfaces. Each point on this Euclidean trim curve encompasses a range of normals from one surface normal to the other surface normal. Following the idea of orthogonal projection, it follows that a range of HIPs can project onto the Euclidean trim curve, so the closest point on the model may lie on the trim curve, rather than on any particular surface.

A general transition from one surface to the next may take the form of detecting a closest point in the trimmed-away domain, finding the closest point on the Euclidean trim curve and possibly moving on the trim for a number of updates, then transitioning onto the adjoining surface and resuming DPT on that surface. These elements each require their own approaches.

17.5.3 Trim Intersection

Discrete movement along the surface correlates to a directed line segment in parametric space. This segment is constructed using the current closest point's parametric coordinates and the next location calculated using direct parametric tracing. If this segment, or *movement vector*, intersects any of the surface's trimming segments then a boundary has been crossed. The location of the intersection is determined by selecting the intersection point closest to the current contact point.

Since the number of trimming segments per surface can be very large, it is not possible to check every segment for intersection. Multiple acceleration data structures are reasonable choices for speeding this problem.

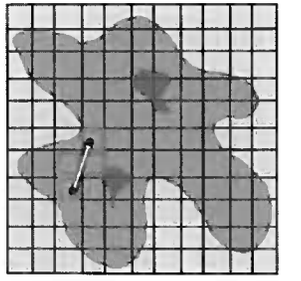


Figure 17.6. A spatial grid reduces the number of trim segments that must be checked for intersection. Only the highlighted cells intersecting the movement vector need to be processed.

In [Thompson II and Cohen 99], a spatial grid in the parametric domain was used. Each cell in the grid contains the trim segments that lie within or intersect it. Each call to to the trim intersection test results in only checking those segments lying within the cells that the movement vector intersects. In addition, a grid walking algorithm checks the cells in the order the movement vector traverses through them. Figure 17.6 shows an example movement through the spatial grid structure. The intersection checks conclude at the first valid intersection, further cutting down on the number of intersection checks performed. In [Museth et al. 05], a hierachical oriented bounding box test was used to find the intersection point. In practice, trimming loops tend not to exhibit pathological behavior, such as repeated self-intersections, so almost any efficiency structure will probably perform well.

17.5.4 Adjacency

In order to smoothly transition from one surface to another, it is necessary to calculate an accurate transition point on the neighboring surface. Our system maintains an *edge adjacency table* for each surface. This table allows efficient determination of the adjacent surface, as well as the appropriate trimming loop and edge onto which the transition should occur. Not all CAD file formats retain this topological connectivity information, in which case it would have to be reconstructed through repeated sampling of the closest point from a point on one surface to the other along the trim curve. Given such an edge adjacency table, finding the corresponding point on the adjoining surface is just a table lookup and interpolation along the trim edge.

17.5.5 Edge Tracing and Release

Tracing along a trim edge is closely related to tracing along the surface. The edge tracing algorithm must slide along the edge in Euclidean space to a point locally close to the probes position. Since the trim loop has connectivity information, a hill-climbing algorithm that slides along the trim to a new local minimum is fast and sufficient.

Once the local closest point is found, the algorithm checks to see if the tracked point should release from the trim onto the surface. If DPT performed on the surfaces adjoining the new location moves onto a surface, then the surface point is used. If the updated surfaces points all lie in the invalid, trimmed-away domain, then the closest point remains on the trim curve.

17.6 Stability of Numerical Closest Point Methods

Computing power has increased since the development of the DPT approach, so it is worthwhile examining more accurate closest point update methods. Equation (17.7) showed the system of equations whose roots were the closest points on a plane. By replacing the parametric equation of the plane with a spline surface $S(u, v)$, the system describing the closest point on a surface is obtained. In this form, the scaling factor of -2 is also dropped for simplicity.

$$\mathbf{F}(u,v) = \begin{bmatrix} (P_{HIP} - S(u,v)) \cdot S_u(u,v) \\ (P_{HIP} - S(u,v)) \cdot S_v(u,v) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{17.12}$$

Any number of numerical methods can be applied to solve such a system. Multidimensional Newton's method is a powerful and popular method for such root finding. Given an initial guess at the solution, $x = (u_{init}, v_{init})$, Newton's method iterates finding a Δx that moves $\mathbf{F}(x)$ closer to the root. Multidimensional Newton's method takes the form

$$\mathbf{J}(u,v) \cdot \Delta x^T = -\mathbf{F}(u,v), \tag{17.13}$$

where $\mathbf{J}(u, v)$ is the Jacobian of $\mathbf{F}(u, v)$, or the matrix of partial derivatives. The change in parameter is then found by taking the inverse of the Jacobian and multiplying that with the system of equations:

$$\Delta x^T = -\mathbf{J}(u,v)^{-1}\mathbf{F}(u,v). \tag{17.14}$$

Looking at all the elements of a Newton's method update step, the expanded form is

$$\begin{bmatrix} (P_{HIP} - S) \cdot S_{uu} + S_u \cdot S_u & (P_{HIP} - S) \cdot S_{uv} + S_u \cdot S_v \\ (P_{HIP} - S) \cdot S_{uv} + S_u \cdot S_v & (P_{HIP} - S) \cdot S_{vv} + S_v \cdot S_v \end{bmatrix} \cdot \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = -\mathbf{F} \tag{17.15}$$

where S is the surface evaluated at the current root estimate. Additional geometric insight into degeneracy conditions for Newton's method can be obtained by rewriting the vector between the HIP and the current estimated closest point on the surface $P_{HIP} - S$ in terms of a local coordinate system on the surface, using the tangent plane and surface normal evaluated at the root estimate,

$$P_{HIP} - S = xS_u + yS_v + zN. \tag{17.16}$$

During haptic updates, each discrete step of the HIP is very small. In this case, x and y tend to zero. At the limit, then,

$$P_{HIP} - S = zN. \tag{17.17}$$

Substituting this form into Equation (17.15), the Jacobian used in an update of Newton's method becomes

$$\begin{bmatrix} zN \cdot S_{uu} + S_u \cdot S_u & zN \cdot S_{uv} + S_u \cdot S_v \\ zN \cdot S_{uv} + S_u \cdot S_v & zN \cdot S_{vv} + S_v \cdot S_v \end{bmatrix}. \tag{17.18}$$

Important intrinsic properties of surfaces are called the *first and second fundamental forms* of a surface, **G** and **L**. They are defined as

$$\mathbf{G} = \begin{bmatrix} S_u \cdot S_u & S_u \cdot S_v \\ S_u \cdot S_v & S_v \cdot S_v \end{bmatrix} = \begin{bmatrix} E & F \\ F & G \end{bmatrix}, \tag{17.19}$$

$$\mathbf{L} = \begin{bmatrix} S_{uu} \cdot N & S_{uv} \cdot N \\ S_{uv} \cdot N & S_{vv} \cdot N \end{bmatrix} = \begin{bmatrix} L & M \\ M & N \end{bmatrix}. \tag{17.20}$$

These forms share common elements with the Jacobian of the update step in Equation (17.18), which can be rewritten in terms of the elements of **G** and **L**, or

$$\begin{bmatrix} zL + E & zM + F \\ zM + F & zN + G \end{bmatrix}. \tag{17.21}$$

The matrix inversion step is degenerate when the determinant of the Jacobian is zero. The determinant of the Jacobian equals zero at the roots of

$$z^2 \left(\frac{LN - M^2}{EG - F^2} \right) + z \left(\frac{LG + EN + 2MF}{EG - F^2} \right) + 1 = 0. \tag{17.22}$$

This unwieldy equation actually shares the forms of the sums and product of the principal curvatures of a surface. The principal curvatures, κ_1 and κ_2, define the maximum and minimum curvatures of curves passing through a point on a surface. The sums and products of these curvatures are exactly the coefficients of the determinant of the Jacobian, so the determinant can be written in terms of the principal curvatures as

$$z^2 \kappa_1 \kappa_2 + z(\kappa_1 + \kappa_2) + 1 = 0, \tag{17.23}$$

with roots

$$z = -\frac{1}{\kappa_1} \text{ and } z = -\frac{1}{\kappa_2}. \tag{17.24}$$

Getting back to the original point of all this rewriting, a HIP that is along the normal of the last closest point and at a distance of one of the principal radii of curvature will cause a degenerate update of Newton's method. Points in the neighborhood of these degeneracies will have very poor condition numbers, leading to numerically poor matrix inversions. In light of this, haptic algorithms using numerical methods need to be aware of potential degeneracies and have tests and fallback methods available to

safely update the position when needed. Recent approaches have sought more reliable updates of the closest point on a surface. These approaches include geometric hierarchies based on normal cones [Johnson and Cohen 05], feedback control of the update [Patoglu 05], and symbolic analysis of critical points with numerical updates [Seong et al. 06].

17.7 6-DOF Haptic Rendering of Spline Models

The system of equations describing the conditions for a local minimum distance between the HIP and a spline surface can be expanded to consider the local minimum distance between two surfaces. When the haptic device controls the position and orientation of a model rather than a point, such distance measures are necessary.

A straightforward extension of 3-DOF haptic rendering considers finding the extrema of the distance between two surfaces $A(u, v)$ and $B(s, t)$:

$$D(u, v, s, t) = ||A(u, v) - B(s, t)||. \tag{17.25}$$

As before, extrema occur at simultaneous zeroes of the system of partial derivatives, $\mathbf{F}(u, v, s, t)$,

$$\mathbf{F}(u, v, s, t) = \begin{bmatrix} (A(u, v) - B(s, t)) \cdot A_u(u, v) \\ (A(u, v) - B(s, t)) \cdot A_v(u, v) \\ (A(u, v) - B(s, t)) \cdot B_s(s, t) \\ (A(u, v) - B(s, t)) \cdot B_t(s, t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{17.26}$$

Analogous to the point case, extrema occur when the line between closest points on the surfaces is normal to each surface. The problem with using this system directly in Newton's method is that during model interpenetration, roots not only correctly yield the penetration of the two models but also a curve of zero distance, where the two surfaces intersect. This curve matches the term $A(u, v) - B(s, t)$ going to zero in the set of equations. Local updates can very easily "slide" into these extraneous solutions.

17.7.1 Extremal Distance Formulation

A more robust solution is proposed in [Nelson et al. 99]. The extremal distance between parametric surfaces $A(u, v)$ and $B(s, t)$ may be described by the following equation:

$$E(u, v, s, t) = (A(u, v) - B(s, t)) \cdot N_A, \tag{17.27}$$

where N_A is the surface normal at $A(u,v)$. Extrema of E are when

$$\begin{bmatrix} A_u(u,v)\cdot N + (A(u,v) - B(s,t))\cdot N_u \\ A_v(u,v)\cdot N + (A(u,v) - B(s,t))\cdot N_v \\ -B_s(s,t)\cdot N \\ -B_t(s,t)\cdot N \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{17.28}$$

Noting that the normal N is orthogonal to the tangent plane formed by the partials A_u and A_v, the terms $A_u \cdot N$ and $A_v \cdot N$ are always zero. Additionally, the partials of N lie in the tangent plane of A, as shown by the Weingarten equations. The equivalent constraint may be formulated by replacing these normal partials with the partials of $A(u,v)$. These substitutions form a simplified set of equations

$$\begin{bmatrix} N\cdot B_s \\ N\cdot B_t \\ (A(u,v) - B(s,t))\cdot A_u \\ (A(u,v) - B(s,t))\cdot A_v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{17.29}$$

The first two equations constrain the solution to collinear normals at the surface points forming the solution and the second two maintain colinearity of the vector connecting the surface points with the surface normals.

This system of equations may be locally solved through incremental updates of the parameters, using multi-dimensional Newton's method,

$$\Delta\mathbf{u} = \mathbf{J}^{-1}(-\mathbf{F}), \tag{17.30}$$

where $\mathbf{J}$ is the Jacobian of $\mathbf{F}$.

This formulation still contains some extraneous zeroes, since there may be multiple locations where the surfaces' tangent planes are parallel and are at a local distance extremum. However, these undesired roots are less common than the extraneous roots from the straightforward extension of the minimum distance formulation. Repeated application of the extremal distance update tracks the penetration depth as two models interpenetrate, as seen in Figure 17.7.

While these equations may be used directly by the haptic rendering system, additional efforts at stability have developed in [Nelson et al. 99]. This approach used a differential parametric contact formulation that evolved the pair of points forming the penetration depth, using surface velocities and the local curvature properties of the surfaces to stably update the penetration depth, using numerical integration of the point movement. Numerical methods guarded against drift of the updated solution.

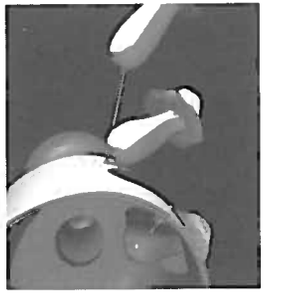


Figure 17.7. As the haptically controlled model pushes into the other model, the extremal distance equations can track the penetration depth between the two models [Nelson et al. 99]. (© 1999 ASME)

17.8 Conclusion

Haptic rendering of spline models shares common characteristics with haptic interaction between polygonal models, but casts the problem into a symbolic form, rather than a geometric one. As distance methods to and between spline models become faster and more robust, these advances can be directly applied to improving the haptic rendering of these models.

match those of the task at hand. Just as zooming too far in or out in the graphics display can produce nonuseful views, zooming the haptic space can cause problems. In the case of the haptic display, there are two dimensions to this scaling: force and spatial. A force scale that is too small cannot be perceived by the scientist, whereas one that is too large can cause instability. This interacts with the spatial scaling: too-large motions of the stylus cause instability at a given force scale, whereas a spatial scale that is too small does not enable the scientist to reach far enough. For many haptic applications, stable and high-fidelity feedback requires a spatial scale that is smaller than the entire data set; this requires systems to include navigation controls to let scientists move to different regions as they explore their entire data set.

23.3 Summary

A number of applications of haptic display for scientific visualization have been described, along with the particular benefits to scientists of adding haptic display. Many of these applications included display techniques that go beyond the straightforward mapping of force from their data sets. Two classes of such forces were described, along with particular examples of useful mappings.

The applications results indicate the real benefits that can be had by adding haptic display to a scientific visualization, both to enable bidirectional coupling and to explore volumetric data without occlusion. Haptic display has been helpful in training, exploration, and experiment steering. The technique discussion points towards the most effective current and future mapping techniques for haptic display in scientific visualization.

24 Haptics in Medical Applications

M. Harders

24.1 Overview

Indisputably the haptic sense plays a paramount role in the medical profession. Be it the simple checking of a pulse, the guidance of a biopsy needle during a lumbar puncture, the palpation of soft tissue for cancer screening, or the detection of pulsating arteries during open surgery, medical practitioners are often required to "see" with their hands. Therefore, it is not surprising that the usage of computer haptics in medicine has been suggested and explored in the past for almost all stages of a patient's treatment process, ranging from the initial diagnostic steps to the concluding rehabilitation phase. A number of possibilities to categorize the various approaches exist, a few of which are briefly presented here.

Tool- versus hand-based interaction. With regard to hardware requirements as well as to the diversity of perceivable sensory cues, interaction directly with one's hand differs considerably from probing via instruments. Usually, the former has a higher order of complexity, with currently no satisfactory solution available. An example would be the simulation of open surgical procedures in comparison to minimally-invasive interventions.

Abstract data versus real entities. The presentation of haptic feedback can vary largely, depending on the respective application. One end of the spectrum would be the highly accurate replication of soft tissue behavior in surgical simulation, which tries to make the virtual object indistinguishable from the real entity, while the other end could be the display of haptics cues to a user to support human-computer interaction in an interactive medical segmentation system, where the target would be the maximization of information flow, and not the faithful reproduction of the realistic feeling of organs.

Augmented versus virtual interaction. Similar to the notion of augmented versus virtual reality, haptic feedback could either be used to enhance or augment real sensations encountered during telemanipulation, or to present completely virtual objects, such as guiding cues, during surgical planning. Since in the former case the real world represents a reference frame onto which additional information is overlaid, more rigorous requirements with regard to stability or latency have to be met.

The following section tries to provide a general overview of numerous related activities of haptics in medicine, grouped by the specific stages in the medical treatment process. Three more detailed examples of medical applications are presented thereafter in the remainder of this chapter.

Data segmentation and visualization. Radiological imaging is a central component in current medical practice, especially in the diagnostic process. A key problem in this area is the automatic extraction of information from the medical image data, which requires an initial data segmentation. Since subsequent higher-level interpretation steps, such as object recognition and classification, feature extraction, or automatic quantification, depend on the quality of the segmentation, considerable effort has been put into improving the latter. In order to support a user in extracting the information buried in the enormous flood of image data, haptically enhanced human-computer interaction systems for computerized medical image analysis and visualization have been a topic of investigation in the past. In [Harders and Szekely 03, Harders et al. 02], a visuo-haptic tool for segmentation of the small intestine has been described. Force fields were generated based on MRI or CT data, as well as from the underlying segmentation algorithms. Processing time could be significantly reduced by providing haptic feedback.

Related work focusing on semi-automatic segmentation has also been presented in [Vidholm and Nyström 05, Vidholm et al. 06]. Various haptic interaction techniques, such as gradient vector flow rendering, have been examined to support segmentation initialization, such as placement of seed points or positioning approximate outlines of objects in the dataset. Similar techniques are also briefly discussed in [Senger 05]. Apart from the extraction of objects of interest from the data, the visualization of medical images can also be supported by haptic feedback. In [Bartz and Gürvit 00], navigation through segments of arterial blood vessels is enhanced with force feedback. A related but more advanced visuo-haptic visualization system for vascular data has also been described in [Yi and Hayward 02]. The system allows the haptic display of vessel connectivity and guides a user in tracing vessel branches.

While the mentioned systems have largely been proven to improve the segmentation process, the integration into clinical practice is still in its infancy. Reasons for this can be found in the considerable cost of haptic devices—a situation which has been only recently ameliorated—and the fact that standard radiological image interpretation is still mired in a 2D display and analysis paradigm.

Telediagnosis. Another focus of current investigations mainly focusing on diagnostic settings is haptically-enhanced telemedical systems that can be used to remotely interact with patients. A major line of research in this respect is remote palpation. In [Kim et al. 05], a device for measuring and presenting pressure-based human vital signs has been presented. Signals are acquired with a piezoelectric sensor and fed back to a user via a PHANTOM haptic device. However, the system is still at a developmental stage. Another strategy is the combination of teletaction systems with robotic manipulators to perform active remote palpation of patient tissue. An anthropomorphic robotic hand for breast cancer diagnostics with tactile sensing and haptic feedback has, for instance, been described in [Methil-Sudhakaran et al. 05]. Tissue compliance is acquired with an optical tactile sensor integrated into a robotic hand and then displayed to a physician through electrotactile stimulation. Related to this, a haptic sensor-actuator system for remote examination has also been presented in [Khaled et al. 03]. The mechanical consistency of an object is determined via ultrasound elastography and then displayed to a user via an actuator array based on electrorheological fluids. Another related setup for multifingered tactile feedback from virtual or remote environments has also been proposed in [Kron and Schmidt 03]. Early work in this direction focusing on augmentation of minimally invasive palpation to localize arteries or tumors has already been reported in the 1990s in [Howe et al. 95]. The underlying idea is to equip surgical instruments with tactile sensors at the tip and tactile displays in the handle, to enhance a surgeon's perception during minimally invasive surgery. This class of approaches will be covered in more detail below in the context of intra-operative support.

Surgery and therapy planning. A step usually following the initial diagnosis and visualization phase is the planning of the necessary therapeutical procedures. In this context, haptic feedback has been applied to support surgical planning. In [Giess et al. 98], haptic volume rendering is provided to assist a user in distinguishing transitions between different liver segments for resection planning. The additional cues aid the radiologist in the setting of landmarks directly in 3D, thus avoiding the more time-consuming search for optimal slices in 2D. In [Tsagarakis et al. 06], a multimodal interface for preoperative planning of hip arthroplasty has been introduced, which integrates immersive stereo display with a prototype haptic device. Rendered forces assist the surgeon in evaluating access to the surgical site

and in placement of implant material. Other related approaches have been suggested for bone cutting in maxillofacial surgery [Burgert et al. 00] or for adjusting doses in stereotactic radio-surgery [Olofsson et al. 04]. The former system provides haptic feedback during the removal or addition of fatty or bony tissue to optimize the visual appearance of a patient undergoing plastic surgery, while the latter system renders forces based on dose distributions to optimize radiation treatment. Unfortunately, all these systems have not left the prototypical stage and are not used in daily clinical practice.

Intra-operative support. Providing support during an intervention following diagnosis and planning steps has been a very active area of investigation, especially in the field of teleoperated robot-assisted surgery. Excellent surveys of the numerous existing research activities have been compiled in the medical robotics literature, e.g., [Taylor and Stoianovici 03, Pott et al. 05, Cleary and Nguyen 01, Howe and Matsuoka 99]. Nevertheless, a few selected key activities focusing on haptic feedback will be examined in the following paragraphs.

Telesurgical robotic systems extend a surgeon's ability to perform small-scale manipulation tasks and help to cancel out hand tremor, as for instance reported in [Taylor et al. 99]. Nevertheless, the lack of haptic feedback is often seen as a major limitation of these set-ups. While it has been argued that several interventions have been successfully performed without feedback, e.g., as reported in [Shennib et al. 98, Mohr et al. 01], operation times are often found to be longer and exerted forces higher. Therefore, equipping minimally invasive tools with sensors and actuators has been an active area of investigation. Examples for enhanced surgical instruments can be found in [Yao et al. 05, Rosen et al. 03, Scilingo et al. 97], while work in the context of telesurgical robots has been reported in [Madhani et al. 98, Hoshino et al. 01, Okamura 04]. Related to this work are projects examining haptic mechanisms that provide active guidance and augmentation by working cooperatively with a physician. For instance, the system described in [Hagmann et al. 04] combines virtual reality techniques with haptic rendering to support blind needle placement into tissue.

Other work examined the integration of force feedback to provide virtual fixtures [Rosenberg 93] during interventions. This concept has been tested in the context of robot-assisted coronary artery bypass graft procedures [Park et al. 01] or microsurgical applications [Kragic et al. 05]. An intelligent tool has been presented in [Nojima et al. 02]. A scalpel was equipped with a photosensor to detect interfaces between materials. A haptic mechanism provided forces to prevent a user from penetrating the detected interfaces. This tool has been used to guide a user while cutting through a boiled egg, avoiding damage to the yolk.

Rehabilitation. Haptically enhanced systems have also been proposed to support and assess progress during physical rehabilitation after therapy. Enhancement of patient attention and motivation is a key issue in this respect. As an example, in [Deutsch et al. 01] a Stewart-platform-type haptic interface has been used in rehabilitation. Improved clinical measures of strength and endurance resulted for patients working with the system. Another example is reported in [Loureiro et al. 01], where patients with arm impairment following stroke used a haptic system. More details on the use of haptics in rehabilitation are covered in Chapter 25.

Medical education. Virtual-reality-based simulators are an appealing option to supplement educational curricula in the medical domain [Liu et al. 03, Basdogan et al. 07]. First attempts at using computer-based simulations for education of prospective surgeons had already been carried out at the beginning of the 1990s, e.g., [Green et al. 91, Satava 93]. An extensive online repository of past and present surgical simulator projects has recently been established, as indicated in [Leskovsky et al. 06]. These simulation endeavors have had a considerable influence on the development of the field of haptics, which is reflected in the number of proprietary, as well as commercial, devices available specifically for these medical training setups.

The majority of these interfaces aim at laparoscopic interventions. A four-degrees-of-freedom spherical haptic device, the PantoScope, has been introduced in [Baumann et al. 98]. It has been developed for the simulation of minimally invasive interventions in laparoscopy. This prototype was later on improved and commercialized by the Swiss company Xitact. Another proprietary input device with five DOF using actual surgical tools for cholecystectomy has been described in [Kuehnapfel et al. 95]. In [Hayward et al. 97], the Freedom 7S has been presented—a high fidelity force feedback device providing seven degrees of freedom including force feedback for an interchangeable scissors grip. This prototype is now distributed through the Canadian company MPB Technologies. Moreover, another prototype system providing seven degrees of freedom has been discussed in [Tholey and Desai 06]. A proprietary haptic interface for hysteroscopic interventions has been used in [Montgomery et al. 01]. It was later taken over by the US company Immersion, resulting in the Hysteroscopy AccuTouch simulator system. They also built the Laparoscopic Impulse Engine, an early product which provides four-DOF feedback for surgical simulations. The device was later on replaced by the Laparoscopic Surgical Workstation, which incorporates a bi-manual interface with haptic feedback. Furthermore, in [Payandeh and Li 03], a number of design concepts for haptic devices usable in minimally invasive surgery have been surveyed. Additional developments focusing on endoscopic tools have been reported in [Vlachos et al. 03, Spaelter et al. 04, Trantakis et al. 04]. A number of specialized

devices were also developed for medical application areas other than laparoscopy, e.g., for catheter insertion in interventional radiology [Anderson et al. 02, Ilic et al. 05, Cotin et al. 00], lumbar punctures [Singh et al. 94], colonoscopy [Ikuta et al. 99, Koerner and Maenner 03, Yi et al. 06], or endoscopic retrograde cholangio-pancreatography [Peifer et al. 96].

In addition to rendering contact forces, a focus has also been on providing feedback for tool handles, e.g., uniaxial forces during insertion of epidural needles [Brett et al. 97] or feedback during cutting with scissors [Okamura et al. 03]. In contrast to this, some work examined interactive patient mannequins. In [Riener et al. 04], a mechanical actuator has been attached to a passive phantom limb to provide force feedback, while also allowing direct manual exploration of the mockup. Finally, some groups also examined the connection of surgical instruments to commercially available haptic devices—almost exclusively using the PHANTOM device—e.g., for simulation of laparoscopy [Szekely et al. 00], lumbar punctures [Gorman et al. 00, Dang et al. 01], spine biopsy [Kyung et al. 01, Ra et al. 02, Lathan et al. 00], or catheter insertion [Zorcolo et al. 00].

For integration of haptic feedback into a surgical simulator system, the haptic hardware is only one of the necessary elements. In order to render appropriate feedback, a number of components are needed. The replication

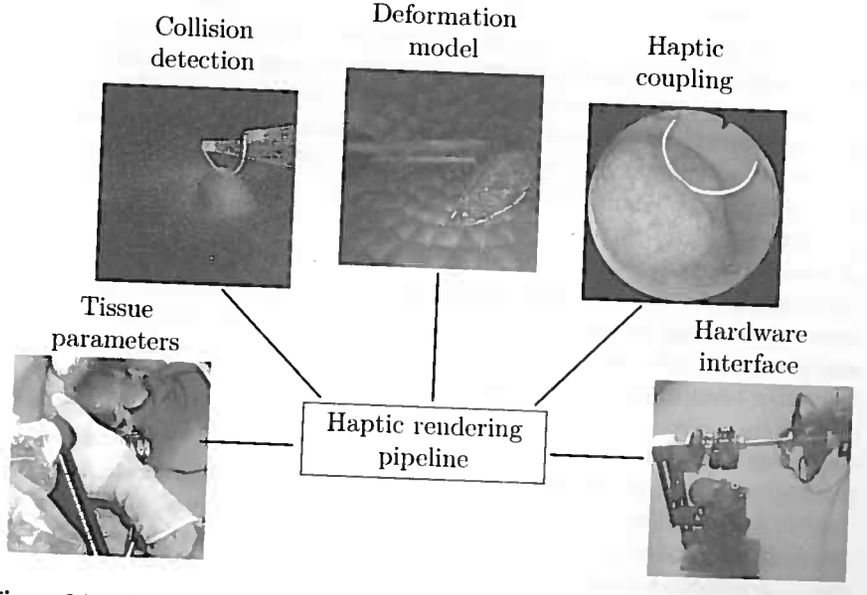


Figure 24.1. Haptic rendering pipeline for feedback generation during soft tissue interaction.

of soft tissue interaction, which is the most common in surgical simulation, requires methods for real-time collision detection and response, soft tissue deformation algorithms, appropriate tissue parameter setting, and coupling between the physics simulation and the haptic feedback loop. These elements can be considered as a haptic rendering pipeline in a surgical simulator, as depicted in Figure 24.1. More details on collision detection can be found in Chapter 9, while soft tissue interaction is discussed in Chapter 20.

24.2 Visuo-Haptic Segmentation of Radiological Data

Digital radiological imaging is an indispensable element of modern medicine. The newest generation of medical image acquisition devices is capable of producing 3D patient datasets with several thousand high resolution images. These leaps in the area of image acquisition are, however, not reflected in the process of image analysis and visualization. In spite of considerable efforts during the past decades, medical image segmentation is still a major bottleneck. Neither purely manual nor fully automatic approaches are appropriate for the correct, efficient, and reproducible identification of organs in volume data. In the current practice, interactive approaches, which try to merge the advantages of the former techniques, are still the only robust option. Therefore, extensive research has been invested in recent years into improving interactive segmentation algorithms.

However, the human computer interface, a substantial part of any interactive setup, is only seldomly addressed in the medical context. In order to alleviate the limitations of visual-only systems, haptically enhanced human

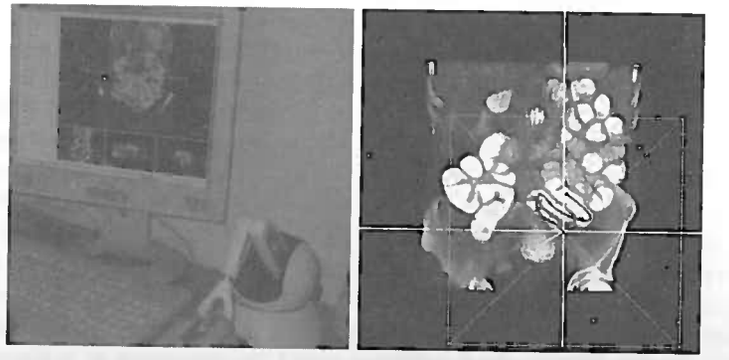


Figure 24.2. Visuo-haptic segmentation system for the extraction of the small bowel and its centerline.

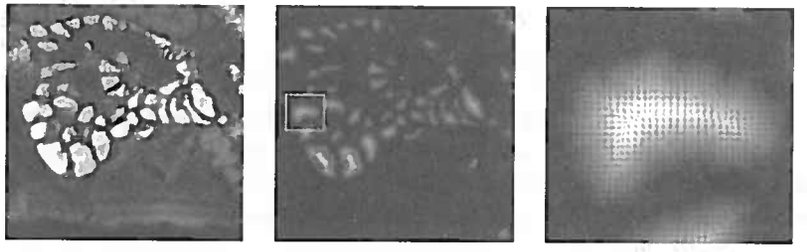


Figure 24.3. Force fields obtained from CT or MRI data guide a user during the semi-automatic segmentation process.

computer interaction for computerized medical image analysis and visualization has been a topic of recent research. One of the first systems [Harders and Szekely 03, Harders et al. 02], developed at ETH Zurich in collaboration with the University Hospital Zurich, targeted the highly complex task of segmentation of the small intestine (Figure 24.2). No satisfying conventional solution existed for this problem. The underlying idea of the project was to provide guiding force cues to users navigating the tubular structure of the intestinal system. Such a technique can be compared to the notion of virtual fixtures, which is sometimes applied in teleoperation to guide a user in carrying out manual tasks (see e.g. [Rosenberg 93, Sayers and Paul 94]). In the system, force maps for haptic rendering are based on gradient fields of 3D Euclidean distance maps of voxel data intensities (Figure 24.3). The guiding forces support a user during navigation of the tubular structure of the small intestine and facilitate the initialization of semi-automatic segmentation methods, such as deformable surfaces.

The usability of this multimodal approach has been shown in several different studies. In a path tracing experiment through an artificial dataset, users showed a statistically significant performance improvement in the trial time when using haptically enhanced interaction. In addition, in the haptic condition, the quality of segmentation was always superior to the one without force feedback. Similar results were also obtained with real clinical data. In a pilot study with radiologists, guiding forces were used to haptically assist the extraction of the centerline of the small intestine. Based on the latter, a deformable surface model was initialized for the subsequent automatic segmentation. In the latter stage, the surface mesh is deformed, subject to a thin-plate-under-tension model. Due to the fidelity of the haptically assisted initialization, only a few steps were needed to approximate the desired organ shape. Using the system, topologically correct models of the small intestine could be extracted in a fraction of previously reported manual segmentation times. Based on the extracted centerlines and segmentations, virtual fly-throughs of the small intestines

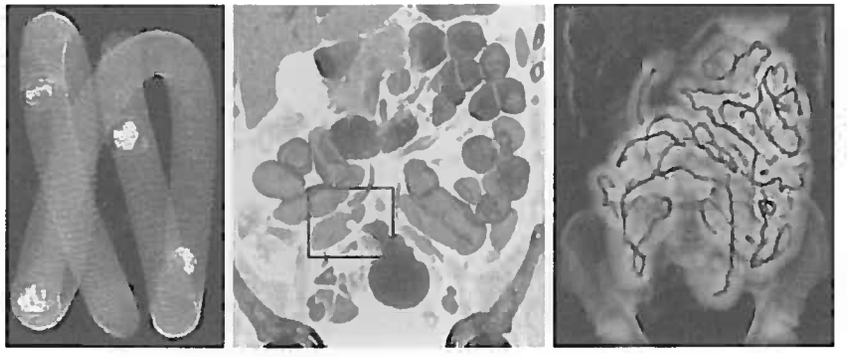


Figure 24.4. User studies with the visuo-haptic segmentation system. CT scan of wooden spheres in phantom tube (left). Artificial lesions in real CT scan (middle). Extracted centerline through small bowel (right).

were created, thus providing a new tool for diagnostics of gastrointestinal diseases.

In an additional study, the accuracy of the described system was evaluated with regard to centerline definition and distance measurements, both in a bowel phantom and in patients. For the phantom study, wooden spherical particles were placed at defined intervals within a polyethylene tube. After obtaining CT slices of the phantom (Figure 24.4(left)), test participants from the Zurich University Hospital Radiology department assessed the locations of the artificial lesions. The relative distances of the latter in the artificial bowel could be precisely reproduced with the system. In addition to this study, artificial lesions were also added to datasets obtained from real patients (Figure 24.4(middle)). The task of the test participants was to detect and localize these lesions, either with conventional medical imaging software or with the haptically enhanced segmentation tool. The complementary approach of visual and haptic user interaction allowed reliable and complete centerline path definitions of the small bowel (Figure 24.4(right)). Moreover, all simulated small bowel polyps were readily detected. Applying the system resulted in slightly shorter review times; however, the differences were not significant.

24.3 Immersive Virtual-Reality-Based Hysteroscopy Training

The great potential of surgical simulation has consistently been recognized; however, the formal integration of VR-based training systems into the med-

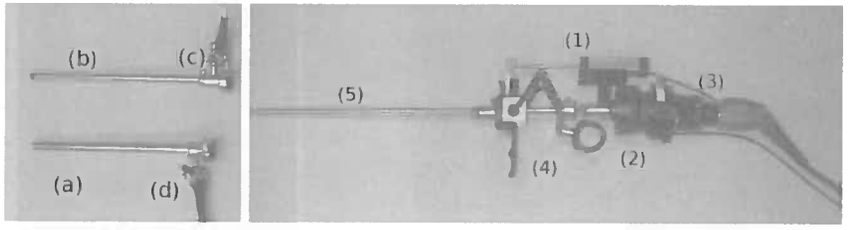


Figure 24.5. (a)–(d) Hysteroscope inflow and outflow tubes and valves. Resectoscope mechanism, with cutting set-up (1,4), camera (2,3), and instrument shaft (5).

ical curriculum is still lacking. It is often hypothesized that the lack of a reasonable level of realism hinders the widespread use of this technology. In a collaborative project of ETH Zurich, EPF Lausanne, University Hospital Zurich, and ZHW Winterthur, this situation was tackled with a reference surgical simulator of highest possible fidelity for procedural training. The focus of these endeavors is the development of a training system for hysteroscopic interventions. Hysteroscopy is the standard procedure for visualization and treatment of the inner uterine surface, and is commonly used in gynecological practice. Although rare, serious complications such as uterine wall perforation, intrauterine bleeding, or fluid overload syndrome exist. To reduce the complication rate, specialized simulator training could be applied to enable rehearsal of manipulative, as well as cognitive, skills. In order to provide the necessary fidelity, several specialized components had to be developed. In this respect, the sense of presence plays an important role in the training effect, which can be achieved. To enable user immersion into the training environment, the interaction metaphors should be the same as during the real intervention. A key component in this respect is computer haptics.

The haptic module of the simulator framework serves two major functions—it provides the interface with which the simulation is controlled, and it displays force feedback to the user [Harders et al. 07]. An actual surgical instrument has been modified in order to allow natural control of the intervention. Moreover, a haptic mechanism providing force feedback and allowing complete removal of the instrument has been integrated [Spaelter et al. 04].

The interface of the simulation is an original surgical tool, which was slightly adapted for the system. Figure 24.5 shows the modified hysteroscope with sensors for the inlet and outlet valve positions, camera angle, camera focus, and cutting tool position. Signal and power cables of the sensors are hidden in the unused fluid tubes or standard instrument cables.

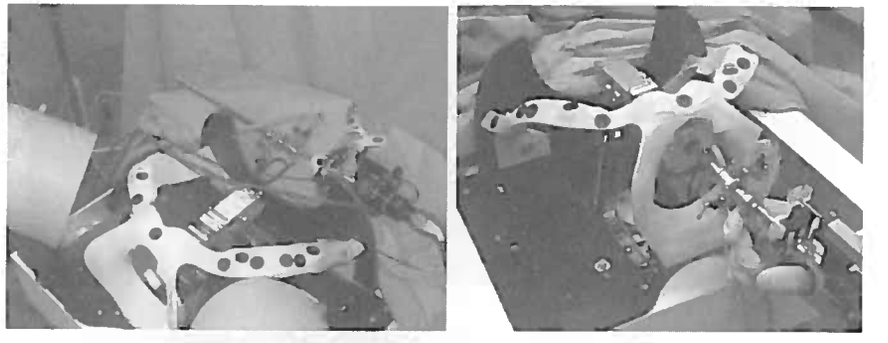


Figure 24.6. Haptic interface for hysteroscopy simulation—hysteroscope removed from hidden mechanism (left), inserted surgical tool (right).

The tool can be completely disassembled and reassembled—for instance, at the start of a training session, the fluid flow tubes have to be fitted to the instrument shaft. Moreover, the tool is not fixed to the force feedback frame. Since complications can already occur during tool insertion into the cervix, this step is included in the training process. Force feedback is generated by a haptic mechanism, into which the tool can be seamlessly inserted. The treatment of the uterus demands a large workspace ($\pm 60°$ pitch and yaw), especially as the anatomy of the uterus can vary within a wide range between individuals. At the same time, the device has to be compact due to the confined space within the female dummy torso (Figure 24.6). The base linkage has two degrees of freedom for spherical displacement around a virtual pivot point. Inertia is reduced by fixing the actuators of the parallel structure to the base. The virtual pivot can be placed in free space without mechanical connection to linkages. This allows one to hide the mechanical structure inside the patient dummy torso. A serially attached head provides tracking and force feedback for the tool translation along and rotation around the tool axis. A friction drive [Spaelter et al. 06] provides smooth and slip-free tool translation and rotation during insertion or complete removal of the surgery tool, which can occur at simulation start, as well as during the training session. The manipulator can transmit pitch and yaw torques up to 0.5 Nm, roll torques of 0.02 Nm, and translational forces of 2 N.

A low-level control scheme tracks the displacement of the surgery tool and hands it over to the virtual environment via a UDP socket connection. The control loop for stable and transparent haptic interaction runs at >1 kHz under the real-time operating system RTAI-Linux. Although active human motion control capabilities rarely exceed 10 Hz, tactile perception can detect vibrations at much higher frequencies, thus making high

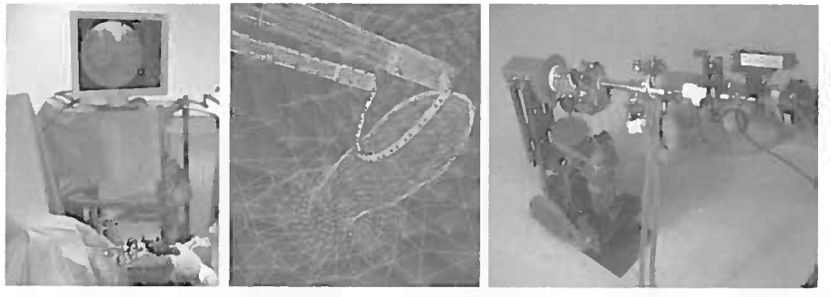


Figure 24.7. Elements of haptic interface module: view of complete setup (left), example scene with dual mesh representation (middle), haptic interface (right).

update rates necessary [Sharpe 88]. Virtual coupling techniques provide a data exchange between the fast haptic control loop and the slower virtual environment. In order to increase haptic realism, gravitation, inertia, and friction of the haptic device are compensated or reduced by control.

For the generation of force feedback, a point-based haptic proxy paradigm is followed [Ruspini et al. 97]. This technique is applied to single, as well as multiple, interaction points. The objects in the simulation have a dual representation—tetrahedral meshes are used for collision detection and the calculation of tissue deformation, while surface meshes are employed for visualization and local proxy update [Tuchschmid et al. 06]. Moreover, tools are approximated by a collection of collision points. Collisions are detected via a spatial hashing algorithm [Teschner et al. 03]. If a collision takes place, the force applied to the deformable object and the appropriate haptic feedback is determined.

Proxy points in the applied model are updated according to the movement of the surgical tool, and, in case of a collision, restricted to the surface of the virtual object, while locally minimizing the distance to the tool. Based on the penetration depths, the interaction force can be obtained. It is distributed to the nodes of the contacted surface triangles and provides the external force vectors for the computation of object deformation. Components of the haptic rendering pipeline of the simulator system are shown in Figure 24.7.

24.4 Multimodal Augmented Reality for Open Surgery Training

In contrast to simulation of minimally invasive interventions, open surgery simulators are still in their infancy. Open procedures are considerably more

difficult to simulate, since the surgeon usually has direct visual and haptic contact with the operation site, and his interaction is much less restricted. So far, only unsatisfactory and strongly limited systems have been developed. A number of related projects focused on suturing tasks and wound debridement. In [O'Toole et al. 99], a training framework for vascular anastomosis was introduced. Visuo-haptic collocation was achieved using a mirror set-up, including stereo rendering of the scene from a fixed viewpoint. Bimanual interaction was possible via two haptic devices. Other projects focusing on open surgery incisions and suturing were also carried out, e.g., [Webster et al. 01, Berkley et al. 04, Bielser and Gross 02]. However, visuo-haptic collocation, full display of the surgical scene, a user-controlled viewing position, and direct manual interaction are usually not integrated. A more immersive and complete simulation of open surgery was attempted in [Bro-Nielsen et al. 98]. A monitor for scene rendering was mounted horizontally into a special purpose stand with the head and legs of a mannequin attached, thus including passive haptic feedback of the patient to a trainee. Nevertheless, visuo-haptic collocation was not provided. Most of these open surgery simulators lack immersiveness, since visual and haptic cues from the virtual patient are strongly limited.

A related successful category of systems are manikin trainers for anesthesia [Cooper and Taqueti 04]. These setups provide—usually inside a real OR environment—life-sized dummy patients that are capable of producing physiologic signals, react to anesthetic interventions (e.g., administration of drugs), and can be interfaced to standard anesthetic equipment. Moreover, a number of critical anesthetic situations can be initiated via external control stations. While effective training for anesthesia personnel can be provided with these systems (e.g., [Chopra et al. 94, Gaba et al. 98]), surgical training is not accommodated.

Recent work at ETH Zurich examined the possibility of providing an environment where open surgery training can be carried out in an immersive fashion. This endeavor targets the extension of anesthesia simulators with *augmented reality* (AR) technology [Azuma 97]. Using the latter, the virtual operation site can be augmented onto a real patient dummy. To provide multimodal feedback in the simulation, haptic interfaces need to be integrated, thus requiring high accuracy and stability of the overlay process. Misalignment of augmented virtual objects would greatly compromise manipulative fidelity and the sense of presence, and thus reduce the overall training effect.

The basic paradigm of a multimodal AR setup is to capture a view of the real scene with a head-mounted camera, superimpose virtual objects in the image, and display the augmented scene with a head mounted display. To ensure exact alignment between the real and virtual worlds, the system needs to determine the relative position between the virtual objects and the

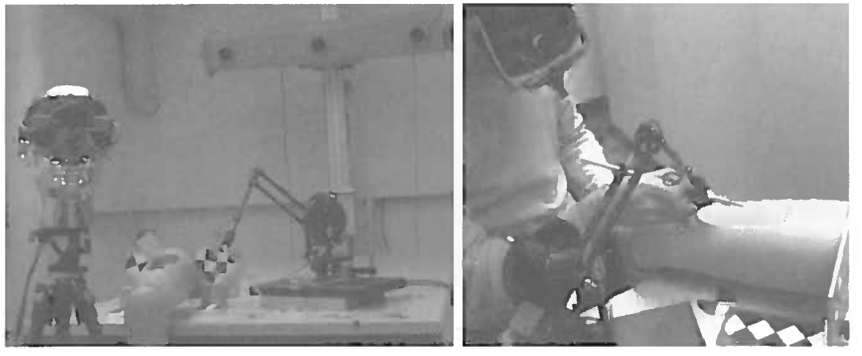


Figure 24.8. Set-up for an augmented reality visuo-haptic training system.

user's head. Therefore, accurate estimation of the head pose, with respect to an arbitrary world coordinate system in which the virtual objects are placed, is necessary. The developed AR system comprises an optical position tracking device, the Optotrak 3020 manufactured by Northern Digital Inc., a head-mounted FireWire camera, and a camera-mounted marker. An overview of the main components and a typical interaction are depicted in Figure 24.8. The optical tracker consists of three fixed linear cameras, which detect the infrared LEDs attached to a marker. By triangulation, the optical system measures the 3D LED position with an RMS accuracy of 0.2 mm at an optimal distance of 2.25 m. From these measurements, the orientation and position of the marker are computed. Since the camera and marker are rigidly attached to each other, the camera-marker transformation is fixed and can thus be estimated by an offline process using hand-eye calibration [Bianchi et al. 05]. Experiments resulted in a back-projection error of approximately two pixels, which is a sufficient starting point for applying hybrid tracking with image-space error minimization. Given the camera-marker transformation and the marker pose, the AR system can estimate the camera pose within the tracker coordinate frame. The IR tracking data are inherently noisy due to the inaccurate measurements of the LED positions. The precision of the measurements becomes even more limited when the head-mounted marker moves. As a consequence, the registration between the real and the virtual world is affected, causing instabilities of virtual objects in the augmented images. Therefore, the estimated camera pose of the IR optical tracker is corrected with a vision-based approach [Bianchi et al. 06a]. A back-projection error of less than 0.7 pixels can be achieved in less than 1 ms computation time. Moreover, using additional refinement of 3D landmark positions, the error can be further reduced to about 0.3 pixels. This estimated camera pose then finally allows the visual alignment of the virtual and the real world.

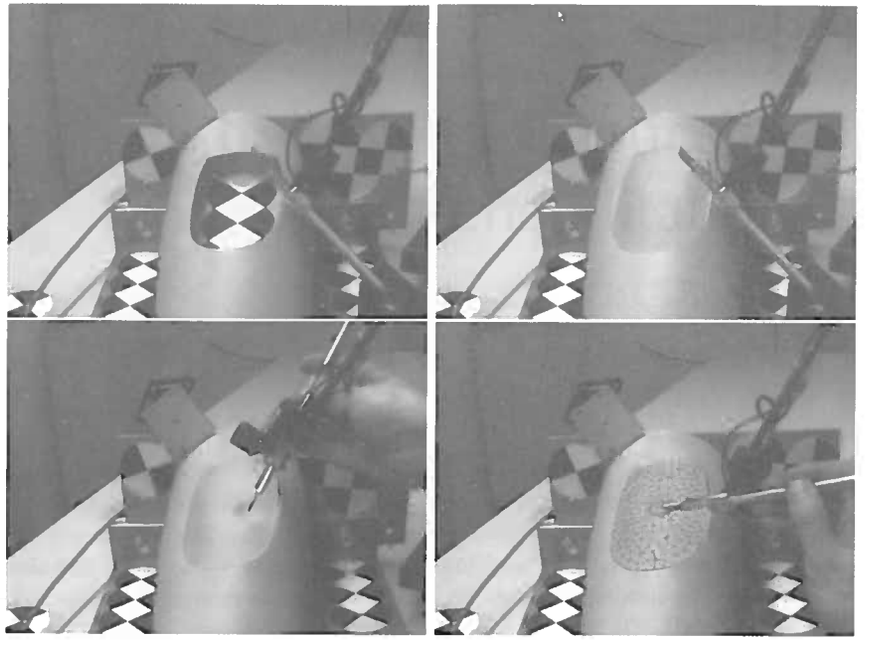


Figure 24.9. Interaction in visuo-haptic augmented reality with real and virtual scene objects via a scalpel mounted to a PHANTOM haptic interface.

In order to allow simultaneous interaction with real and virtual objects in the multimodal augmented reality environment, visuo-haptic collocation is a prerequisite. This, for instance, allows interaction with virtual and real objects in the augmented scene via the same tool. In order to align the virtual representation of the haptic interaction point with the correct physical location in the real world, the relationship between the haptic and the world coordinate system needs to be determined. The first step of the calibration procedure is to collect 3D point measurements in the coordinate systems of the haptic device and the optical tracker. After acquiring 3D point correspondences, the absolute orientation problem needs to be solved [Bianchi et al. 06b]. Since additional errors in the estimation of the haptic-world transformation are introduced due to inaccuracies in haptic encoder initialization, a two-staged optimization process is followed. Using this approach, the final calibration results yield an alignment error below 1.5 mm within the whole workspace of the haptic device. Figure 24.9 shows the interaction with virtual soft tissue embedded into a dummy leg via a real scalpel attached to the haptic device.

25
The Role of Haptics in Physical Rehabilitation

G. C. Burdea

While the majority of today's haptic interfaces and applications are targeted at the able-bodied user, a rapidly growing field of science studies the use of this technology in physical rehabilitation. There are many reasons the reader may wish to take a closer look at this application domain. One reason concerns societal impact, as there are about 70 million people with disabilities in the European Union [Bühler 97]. Such therapy is needed by various patient populations ranging from post-stroke survivors, to those with traumatic brain injury, cerebral palsy, spinal cord injuries, musculo-skeletal deficits, and others. The United States alone spends about $30 billion every year on physical rehabilitation [Patton et al. 06]. Of the above-mentioned costs, the majority represent labor costs (therapist time), and economic pressures tend to make rehabilitation interventions shorter than in prior years.

Rehabilitation science, in contrast to current rehabilitation practice, has recently shown that intense and longer physical therapy will benefit even chronic patients through the phenomenon of "brain plasticity." By repeating meaningful limb movements, similar to those done in *activities of daily living* (ADL), dormant neurons are recruited into new neural paths, and patients regain some of their lost function. Here robots are ideal, since they can train patients for the required long duration without tiring (unlike human therapists), and may eventually lead to a reduction in labor costs.

Robotic systems coupled with virtual reality simulations bring additional improvements to today's conventional physical therapy methods, since they introduce objective measures of performance. Data on total exercise time, speed and smoothness of movement, peak and average velocities, mechanical work, and endurance are among the variables that can be stored transparently and used to objectively gauge a patient's progress. This is a clear departure from the subjective therapist's evaluation of a patient, which is prevalent today.

When robotics is coupled with virtual reality, the resultant rehabilitation becomes fun, since patients can practice in the form of a video game play. They can also be challenged according to their specific abilities and can be given auditory or graphics rewards for their performance. The flexibility of virtual reality also means that a number of different simulations and haptic effects can be produced by the same hardware, thus creating variety and progression of therapeutic games difficulty to challenge each patient. It is intuitive that any therapy that motivates the patient will produce better outcomes, compared to approaches where the patient is disinterested, bored, and otherwise mentally detached from the task she/he is asked to perform.

A more subtle reason to look at haptic applications in physical therapy is the dual use of the same technology for able-bodied individuals. Such users will benefit from techniques presented in this chapter by augmenting their capabilities and thus improving their task performance in virtual reality or telerobotics applications. After all, disability is a question of degree, and we are all disabled to some extent.

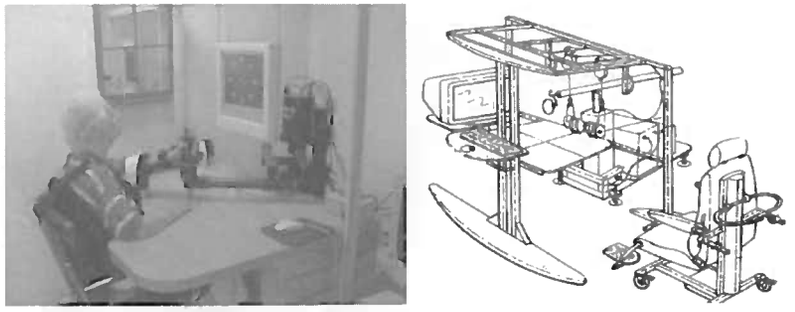
This chapter starts with a review of robotic systems used in physical rehabilitation (Section 25.1), followed by a discussion of the specifics of haptics targeted at the disabled (Section 25.2). Safety issues are clearly important in systems, such as those described in this chapter, where users are in close proximity to the haptic interface or robot. Safety issues for the disabled, which are reviewed in Section 25.3, are even more important, since patients often have degraded hand-eye coordination or cognitive or reflex capabilities, and thus are at higher risk compared to able-bodied users. A look at the future use of haptics in physical rehabilitation concludes this chapter (Section 25.4).

25.1 Robotic Systems for Physical Rehabilitation

The terms *upper extremity* and *lower extremity* are commonly used by physical therapists to refer to either the upper or the lower limbs. Thus, upper extremity rehabilitation aims at improving the patient's shoulder, elbow, wrist, and fingers (and the patient's ADLs). Lower extremity training refers to exercising the patient's knee, ankle, foot, or the whole leg in walking. Robots have been used in physical rehabilitation for more than a decade, and they target all of the above areas of therapy.

25.1.1 Robots for Upper Extremity Physical Rehabilitation

One of the earliest applications of haptics in rehabilitation is the MIT MANUS system shown in Figure 25.1(a) [Krebs et al. 04]. It consists of



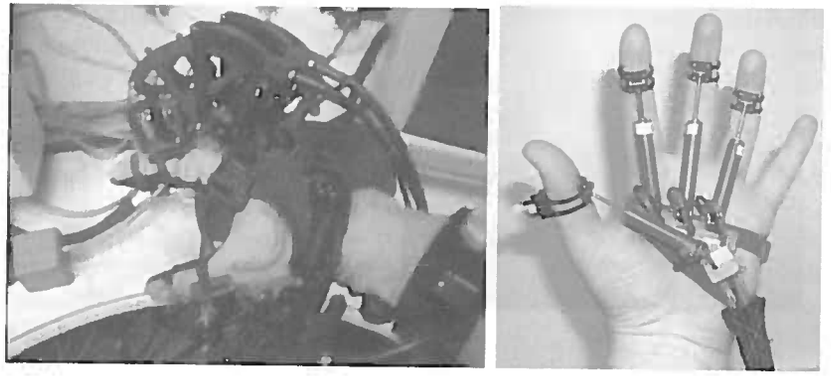
(a) (b)

Figure 25.1. Haptic systems for shoulder rehabilitation: (a) Commercial version of the MIT MANUS [Krebs et al. 04] (Open source material). (b) The Haptic Master [Loureiro et al. 04]. Reprinted by permission.

a direct-drive SCARA two-degree-of-freedom robot that trains the patient arm in a plane while monitoring forces at the end effector. The patient rests the forearm on a special support with safety coupling that detaches in case of excessive forces. The patient is strapped in a chair in order to prevent compensatory torso leaning and faces a monoscopic display controlled by a PC. The robot has its own controller, which implements a back-drivable impedance control aimed at increasing the patient's safety. More recent versions of the MIT MANUS allow the integration of modules for additional degrees of freedom.

Figure 25.1(b) [Loureiro et al. 04] illustrates the adaptation of the Haptic Master, a general-purpose haptic interface, for use in physical rehabilitation. The robot differs from the MIT MANUS, as it has three degrees of freedom and a cylindrical work envelope. Its control is also different, since the Haptic Master uses an *admittance controller* which moves the robot in response to forces applied by the patient on its end effector. Similar to the MIT MANUS setting, the patient is strapped in a chair and faces a monoscopic display showing graphics generated by a PC. These scenes are updated based on the data received by the PC from the Haptic Master. Since the work envelope and output forces of this robot are larger than those of the MIT MANUS, a much more complex apparatus is used to offload gravity-induced forces from the patient's extended arm.

Neither of the above robots is able to train the patient's fingers, which are essential in ADLs. The only commercially available haptic glove is the CyberGrasp (shown in Figure 25.2(a)) [McLaughlin et al. 05]. It consists of an exoskeleton worn on the back of the hand, and five actuators, which apply one degree of force feedback for each finger through a combination of cables and pulleys. Finger sensing is done by the CyberGlove on which



(a) (b)

Figure 25.2. Robots for finger rehabilitation: (a) The CyberGrasp [McLaughlin et al. 05]. Reprinted by permission; (b) the Rutgers Master II [Bouzit et al. 02](©Rutgers University). Reprinted by permission.

the CyberGrasp exoskeleton is retrofitted, and adjustments need to be made for various hand sizes, using mechanical stops on the exoskeleton cable guides. When applied in a physical rehabilitation setting, the weight of the CyberGrasp (about 400 grams) becomes a problem, since patients who need rehabilitation have a diminished arm weight-bearing capability. Furthermore, this weight is placed (by necessity) away from the body, which creates a mechanical amplifier effect.

The requirement for reduced weight is addressed in the prototype Rutgers Master II glove shown in Figure 25.2(b) [Bouzit et al. 02], which weighs about 100 grams. Similar to the CyberGrasp, the Rutgers Master II has an exoskeleton that provides one degree of force feedback per finger (less the pinkie). However, it does not require a separate sensing glove, as its exoskeleton incorporates non-contact position sensors. The glove uses a direct-drive configuration and compressed air, such that each fingertip is resisted in flexion with up to 16 N force. The lack of a separate glove makes its donning faster and easier than the CyberGrasp.

25.1.2 Robots for Lower Extremity Physical Rehabilitation

While robots for upper extremity rehabilitation have existed for over a decade, those used to train the patient's walking and ankle control are more recent. Among them, the best known (and commercially available) is the Lokomat [Frey et al. 06, Riener et al. 06] shown in Figure 25.3(a), used for gait training. Patients with spinal cord injury or post-stroke patients have diminished weight-bearing capacity, which hampers walking. Therefore, therapists use treadmills and passive *body weight supports* (BWSs) in the form of a harness and elastic element to reduce the weight the patient's legs have to support by 60 to 80%. The Lokomat uses the same treadmill + BWS approach, but adds two important elements. The first is a pair of leg exoskeleton robots, which assist the gait cycle with speeds up to about 3 km/h. The robots greatly reduce the therapist's physical effort and thus allow longer therapy than otherwise possible. The second improvement over non-robotic approaches to gait training is the addition of an active (actuator) based BWS in addition to the passive one. The combination of passive + active BWS results in much more uniform weight unloading during walking, and optimal gait training. Recently, the Lokomat has added advanced biofeedback, which immerses the patient in a virtual environment. The patient views the scene of a hiking trail and obstacles that need to be negotiated. If the foot is not lifted high enough, haptic and sound feedback of the collision with the obstacle are produced. A fan provides tactile feedback (in the form of wind) proportional with the patient's walking speed. Thus the patient trains in a meaningful environment, which is adjustable to his/her performance and helps highlight proper walking patterns.

(a) (b) (c)

Figure 25.3. Robotic systems for walking rehabilitation: (a) the Lokomat [Riener et al. 06] (©IEEE). Reprinted by permission. (b) The HapticWalker [Schmidt et al. 05] (©ACM). Reprinted by permission. (c) The Mobility Simulator [Boian 05] (©Rutgers University). Reprinted by permission.

Treadmill training cannot realistically reproduce walking on uneven terrain, such as up and down the stairs. A system that addresses this limitation is the HapticWalker seen in Figure 25.3(b) [Schmidt et al. 05]. Similar to the Lokomat, the HapticWalker consists of two exoskeleton robots that move the patient's legs, coupled with a BWS. The robots incorporate direct-drive electric motors capable of assisting walking up to a speed of 5 km/h. The HapticWalker design uses hybrid serial (large workspace) and parallel (large payload) kinematics. Two actuators connected in parallel

move the foot either up/down or front/back. A third actuator is used to tilt the foot.

Even more degrees of freedom may be needed for realistic haptics and purposeful training. For example, quick horizontal translations overimposed to gait are needed to simulate walking on ice. A robot that can reproduce such haptic effects is the *mobility simulator* prototype seen in Figure 25.3(c) [Boian 05]. Similar to the Lokomat and the HapticWalker, this robot incorporates a BWS system. However, each foot sits on top of a Rutgers Mega Ankle Stewart Platform with direct-drive pneumatic actuators [Boian et al. 04]. Thus, each foot is moved in six degrees of freedom, which allows training for walking on even or uneven terrain (mud, gravel, ice). To provide gait training associated with ADLs, the patient faces a large (monoscopic) display showing a street crossing. The patient has to cross at the pedestrian stop light under various surface, time to cross, and visibility conditions. Distractions, in the form of street noises (honking) or impatient drivers pushing onto the street, are provided for additional training difficulty. Due to the fact that the bases of the two Rutgers Mega Ankle platforms are fixed and their dimensions are more compact than those of the HapticWalker, the step length is smaller that normal values, which is a drawback of the current design.

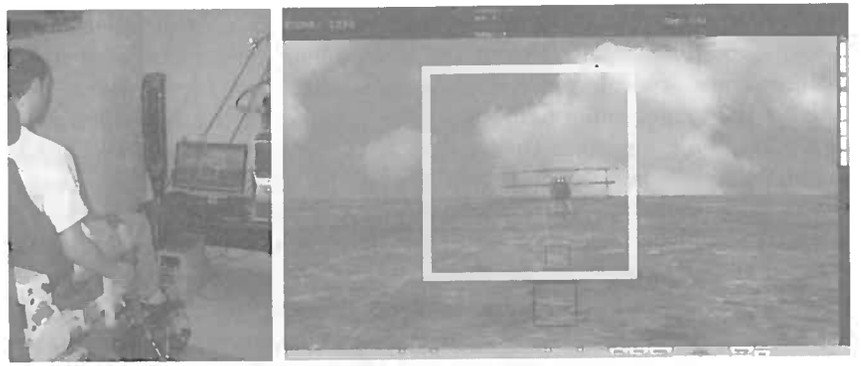
25.2 Specifics of Haptic Feedback for the Disabled

Haptic feedback used in physical therapy is different from that provided to able-bodied users due to the force and motor coordination deficits of the disabled. In domains not related to rehabilitation, haptic feedback is usually in the form of resistive forces which complement graphics and other simulation modalities. Such resistive forces are required to more realistically simulate object compliance, weight, inertia, and surface properties (roughness, stickiness, and friction).

Haptic feedback in physical therapy is more demanding, since it needs to adapt to each patient's functioning level and each therapy session. Furthermore, certain types of haptic feedback (such as vibrations) that adversely affect normal training can prove beneficial in physical therapy. The discussion here is focused on two aspects that play a central role in haptic feedback for physical therapy, namely *assistive haptics* and disturbances.

25.2.1 Assistive Haptics

Dues to the weakened upper or lower extremities of various patient populations, such as those with neurological disorders (stroke, spinal cord injury,



(a) (b)

Figure 25.4. Assistive haptics used to train ankle strength in children with cerebral palsy: (a) System view showing the Rutgers Ankle robot. (b) Screen image highlighting the ideal trajectory the robot is using to pilot the plane while patient is passive (© Rutgers University). Reprinted by permission.

cerebral palsy), the haptic interface needs to assist the patient in performing the simulated task. An example is the use of the Rutgers Ankle robot [Girone et al. 01] in the training of patients with cerebral palsy. Patients sit facing a PC display while their foot is strapped on the mobile platform of the Rutgers Ankle Stewart Platform-like robot. The simulation depicts an airplane that has to fly through a series of hoops while under patient control. In prior studies done with stroke patients, the robot provided purely resistive spring-like forces [Mirelman et al. 06]. This is not possible with children with CP, since at the start of each rehabilitation session their ankle needs to be stretched and moved over its range of motion, with the patient being passive. While in conventional therapy, this is done manually by the physical therapist: here the robot pilots the airplane over an ideal sinusoidal path (see Figure 25.4(a)–(b). During this time, the patient is completely passive. Subsequently, the patient is asked to progressively exert more torques to tilt the foot up/down while the robot creates a "haptic tunnel." Small corrective forces are applied to keep the airplane within an acceptable (threshold-determined) neighborhood of the ideal path. In subsequent rehabilitation sessions, while the patient's ankle exertion capability increases, the robot will switch off assistance and eventually apply resistive forces, which will challenge the patient more.

Another example of graded assistance by a robot is the upper extremity training provided by the MIT-MANUS system. As seen in Figure 25.1(a), the patient is asked to move the robot handle in a plane, such that a cor-

responding cursor on an associated display moves to a highlighted dot out of eight possible targets [Hogan and Krebs 04]. The robot implements an impedance control, which calculates a point that moves on an ideal path to the target while monitoring the position of the end effector. A spring-like force attempts to minimize the distance between the handle position and the moving ideal location on the ideal path. Tests showed this therapy to be useful; however, it did not adapt sufficiently to each patient's condition. This lack of adaptation was due to the fact that the speed of the ideal point on the nominal path was kept constant. A subsequent improvement in the haptic feedback provided by the MIT-MANUS was an adaptive impedance controller which implements a "virtual slot" running between the ideal position and the target position. The walls of the virtual slot are "springy" to provide assistance in case of inappropriate movements away from the ideal path. Furthermore, the back wall of the virtual slot moves to the target with a velocity that assures a fixed duration for a minimum-jerk trajectory. This back wall assists the patient if he or she lags behind the ideal position on the path. However, if the patient can move faster than the virtual slot back wall, he or she is free to do so (while getting no assistance from the robot). The duration of the ideal movement is set automatically based on the patient's past performance. If the patient was able to consistently move faster than the back wall of the virtual slot, then the simulation is made faster, requiring faster arm movements to stay ahead of the robot. Tests showed that this improved therapeutic haptic feedback which was between four to ten times more efficacious than the fixed impedance controller initially used.

25.2.2 Haptic Disturbances to Help Motor Control and Recovery

Haptic disturbances are effects overlaid in the simulation in order to increase therapy difficulty or induce desired after effects. Air turbulence was simulated when piloting the airplane during a storm by oscillating the Rutgers Ankle in the horizontal plane [Boian et al. 03]. Progressively more turbulence determined gradually faster swaying of the robot, while the amplitude of the vibrations was kept fixed. Tests showed that patients gradually learned to cope with these haptic disturbances, eventually being able to clear 100% of the target hoops. This is indicative of improved ankle control, which results in diminished reinjury due to accidents or falls.

Another type of haptic disturbance is illustrated by the graphs in Figure 25.5 [Patton et al. 04]. The curves represent planar arm-reaching movements towards one of six targets while holding a robot arm. Initial undisturbed "baseline" reach movements for a healthy user are plotted in Figure 25.5(a), followed by subject's movements when first confronted with a steady lateral force. Gradually the subject learns to cope with these forces, such that by the end of training (Figure 25.5(c)), the arm moves in straight lines again despite the presence of disturbances. Figure 25.5(d) illustrates the aftereffects of haptic disturbances, as soon as the lateral forces are removed. It can be seen that the arm moves over trajectories, which curve in the opposite direction to the previously applied lateral forces. With continuing repetitions, the trajectory straightens out again, such that aftereffects disappear (or "wash out"). While washing out of learned movements is common with able-bodied users, this is not the case for the disabled [Matsuoka et al. 04]. For the disabled, the effects induced by haptic disturbances do not wash out, because the training leads the patient to activate different sets of muscles. Once the distorting haptic effects disappear at the end of training, the disabled continue to use the new coordinated movements that they learned, using the muscles that had previously been unused.

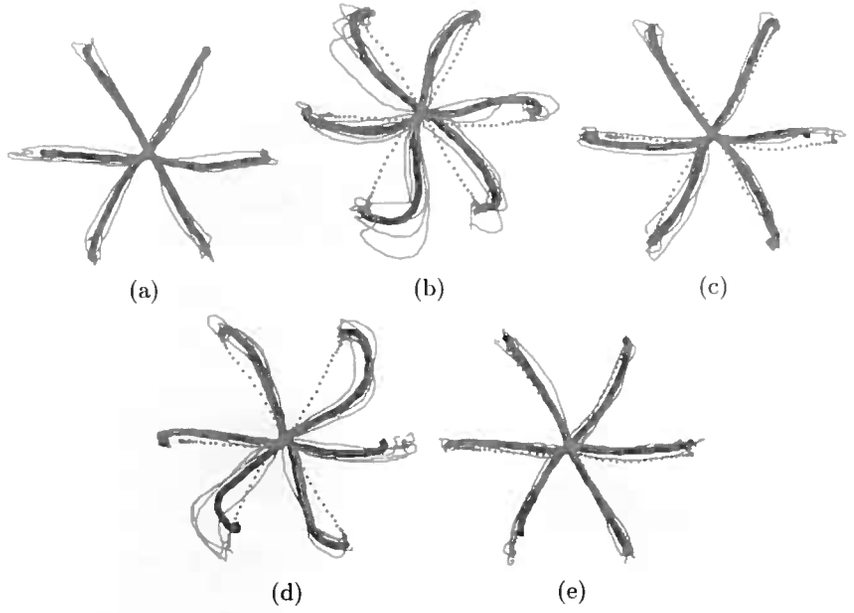


Figure 25.5. Hand trajectories in horizontal plane illustrating aftereffects of systematic haptic disturbances: (a) unperturbed baseline; (b) early training with disturbance; (c) final training; (d) aftereffects when disturbance was removed; (e) final washout. Dotted lines are the initial baseline; bold lines represent average movements [Patton et al. 04] (© IEEE). Reprinted by permission.

25.3 Safety Issues in Haptics for Rehabilitation

While the haptic interface mediates interactions with virtual environments, the forces applied on the user are real. Robots designed for industrial applications, capable of high output forces and large accelerations, pose a real risk when used as haptic interfaces. Even robots designed from the start for physical rehabilitation applications may be dangerous to the patient, since they need to apply large enough forces and torques to make therapy meaningful.

The start of this chapter pointed out that the user's safety is even more important for the disabled. Their slower defensive reflexes, diminished awareness of surroundings, diminished sensory capability (blurred vision, degraded proprioception), and diminished cognitive capacity put the disabled at increased risk when involved in haptics-assisted rehabilitation. It is thus important to look at ways to design computerized physical rehabilitation systems that address the patient's safety concerns mentioned here.

The first line of defense, commonly used in industrial applications, is the provision of safety switches that disable the robot in case of danger. In rehabilitation settings, there should be several such manual switches, one for the patient and one for the attending therapist, who can stop the simulation in case of danger.

Manual switches, however, are not sufficient in a rehabilitation application, due to the slow human response. Additional measures are the integration of sensors and limit switches in the haptic interface itself. This is the approach taken in the design of the HapticWalker patient's foot attachment, as seen in Figure 25.6(a) [Schmidt et al. 04]. The patient wears a shank strap connected to an ankle goniometer through a lever. If the ankle dorsiflexion angle exceeds a prescribed limit, the controller monitoring the goniometer executes an emergency shutdown. Additional safety measures are the thrust pieces that snap in holes that incorporate emergency stop switches. These are built in the supporting plate under the foot, both front and back, and excessive forces detach the thrust pieces and thus trigger a shutdown of the robot.

The above example illustrates the redundancy principle used in good safety design. Several layers of safety measures are necessary in case one layer fails, and designers have to foresee such sensor failures. [Roderick and Carignan 05] describe how they improved the exoskeletons designed for shoulder therapy in order to incorporate redundant layers of safety. Their preliminary analysis identified hazards related to the movement of the patient's arms outside safe position ranges with excessive velocity, or hazards due to excessive torques applied to the patient. Their initial hardware design used an incremental encoder to measure joint values and provide feedback to the servo controller for that joint haptic feedback motor. This design would not prevent motion outside safe ranges if the encoder failed. Thus, the improved design added a second position sensor (an absolute encoder) at each joint. The divergence between the values reported by the two position sensors is monitored to detect failure. The same hardware is used in joint velocity monitoring; thus redundancy is assured in order to prevent excessive joint velocities. In order to build redundancy in force control, the design adds a power amplifier thus senses the power draw of the feedback actuator motor. A motor power divergence check is done in software to detect when the requested output set by the servo controller does not correspond to the motor actual current draw.

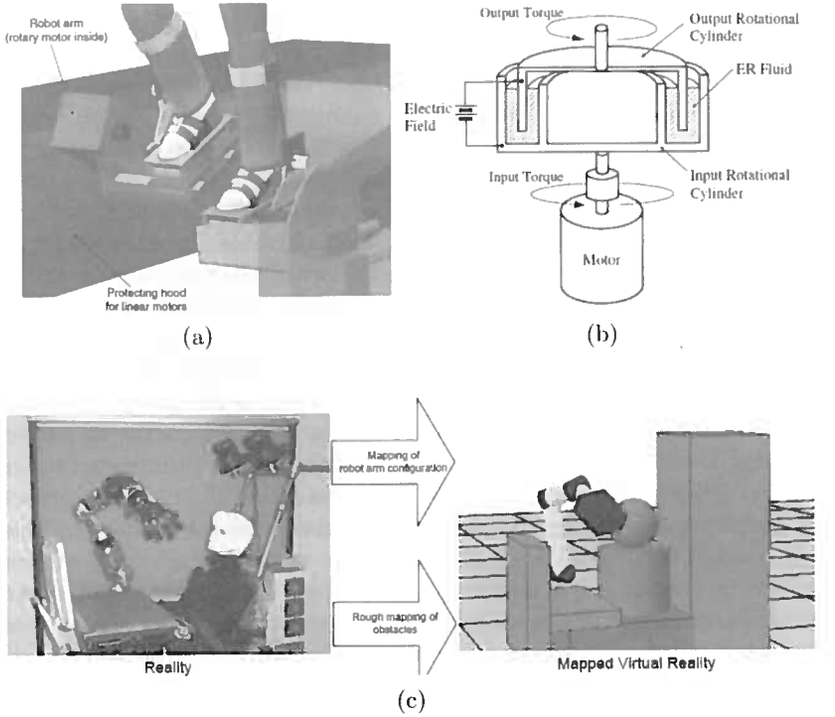


Figure 25.6. Safety methods used when applying haptics in physical rehabilitation: (a) Sensors and mechanical limit switches incorporated in the foot support of the HapticWalker [Schmidt et al. 04] (© IEEE 2004). Reprinted by permission. (b) Electro-rheologic actuator couplings incorporated in a haptic interface for arm rehabilitation [Furusho et al. 05] (© IEEE 2005). Reprinted by permission. (c) Predictive real-time modeling used to prevent patient-robot collisions [Feuser et al. 05] (© IEEE 2005). Reprinted by permission.

Figure 25.6(b) illustrates another approach to increase the safety of a robot used in arm rehabilitation [Furusho et al. 05]. Instead of connecting the actuator directly to the robot joint, the designers use an *electro-rheologic* (ER) coupling. The ER fluid changes its viscosity in proportion to the electrical field applied, which in turn is controlled by the robot controller. Hence it is possible to modulate slippage, thus limiting the potentially dangerous output torques. In case of power loss, the link is decoupled and the robot arm becomes completely back-drivable. In order to further improve safety, haptic interface arm inertia (which does not disappear even when power is lost) is minimized by placing the actuators at the base of the robot and passively counterbalancing the robot arm with weights.

A departure from the previous designs, which relied on robot actuators and internal sensors to improve the patient's safety, is the system illustrated in Figure 25.6(c) [Feuser et al. 05]. It uses a pair of cameras to create a simplified model of the environment consisting of 3D primitives (sphere, cylinder, prism). The robot is modeled as a series of linked 3D objects, and obstacles (including the patient) are also modeled with primitives. Such a simplified model facilitates real-time updates that are performed any time a new object is added or the patient moves. The robot control software performs collision detection using vertex-to-vertex distance calculation (it is thus necessary to convert the primitives to a sparse vertex lattice) [Gilbert et al. 88]. Once the real-time collision detection determines that distances in the updated virtual model fall below a threshold, the real robot is stopped before colliding with the patient.

25.4 Looking at the Future

It is expected that haptics will play an increasing role in physical rehabilitation in the years to come. Based on initial study data, it is expected that the technology will prove efficacious, especially when robotics is coupled with game-like virtual reality training. The penetration of the technology into widespread clinical use will benefit from lower cost hardware, such as game consoles and cheaper haptic interfaces.

Another direction of future growth is the nascent area of telerehabilitation, where therapy is provided at a distance (eventually in the patient's home). It is common in today's rehabilitation practice for the physical therapist to manually manipulate (move, stretch, warm up) the patient's affected limbs. Doing so at a distance will make at-home exercises more meaningful for the patient, without requiring the physical therapist to be co-located. Innovative approaches are clearly required to overcome the problems due to current network limited quality of service (jitter, time delays) in order to implement remote touch.

Acknowledgments

The author's research reported here was supported by grants from the National Science Foundation (BES-9708020 and BES-0201687), from the National Institutes of Health (5R21EB00653302), and the New Jersey Commission on Science and Technology (R&D Excellence Grant).