Chapter 12
Haptic Software Design

Arsen Abdulali and Seokhee Jeon

Abstract This chapter reviews design concepts of haptic modeling and rendering software. The main focus lies in realistic kinesthetic and tactile haptic models for virtual and augmented reality based on the data collected from physical objects. We consider both data-driven algorithms providing a black-box action-response mapping and measurement-based approaches identifying parameters of physics-based models. To show the research landscape and highlight ongoing research challenges, we introduce a series of state-of-the-art methods including data-driven models with deterministic and stochastic responses, physics-based simulation using optimization-based FEM solver, and hybrid approaches of combining the concepts of both data-driven and physics-based methods. These examples also cover a wide range of haptic properties, i.e., modeling and rendering of elasticity and plasticity, tool deformation, and haptic textures.

12.1 Introduction

Computer haptics is a research discipline studying the science, art, and engineering of software design that synthesize and display haptic content. Depending upon the target content, haptic software can be generally classified into algorithms encoding the abstract information and simulating haptic interaction [17]. The abstract content in the former methods is usually represented in the form of tactile patterns allowing utilization of the haptic channel for communication [56], navigation [25], notifica-

A. Abdulali (✉)
Engineering Department, University of Cambridge, Trinity Lane, Cambridge CB2 1TN, UK
e-mail: aa2335@cam.ac.uk

S. Jeon
Kyung Hee University, Seocheon-dong, Giheung-gu, Yongin-si, Gyeonggi-do 446-701, South Korea
e-mail: s.jeon@hapticdevices.eu

tion and warning [33]. The latter approach, commonly known as haptic rendering, represents an interactive process of computing and displaying haptic stimuli with respect to the user's action. The essential role of the rendering is a simulation of the haptic interaction that enables a user to explore the haptic properties of a virtual entity like stiffness and roughness, as well as its physical attributes like shape and weight. The scope of this chapter is narrowed to rendering techniques of haptic interaction, which coincide with engineering aspects of haptic sensors and actuators presented in previous chapters.

The subject touching an object perceives its properties and attributes relying upon *kinesthetic* and *tactile* senses. Kinesthetic perception, also known as proprioception, provides spatial awareness of the body parts and joints, as well as the sense of external forces causing a limb load. This sense allows perceptions of the object's attributes like shape and weight and its internal material properties, which are usually perceived during deformation. The tactile, or cutaneous, sensation allows the subject to perceive surface properties of an object through a skin contact. Both kinesthetic and tactile feedback can be rendered either in the pure virtual configuration where the user perceives only synthetic feedback or by mixing the real physical feedback with a synthetic one.

12.1.1 Virtual Reality

The rendering environment where the user interacts only with virtual objects through haptic interfaces is referred to as *Virtual Reality* [52]. Virtual Reality (VR) is a non-physical computer-generated world that can be either newly created by a designer, e.g., computer game, fantasy world, or can mimic the scene from the real world. VR is typically expands to simulation of the other sensory modalities, e.g., visual and auditory.

The ultimate goal of the VR simulation is to enable the user to feel connected with or being a part of the virtual environment, which is referred to as immersion. Immersive haptic simulation, among other modalities, is considered to be more challenging as the user perceives the world through the prism of abundant haptic properties. Furthermore, sensing organs of haptic perception are ubiquitously distributed in user's body, which makes the design of haptic devices and software even more difficult. To render realistic haptic interaction, a wide range of haptic properties should be simultaneously considered. For instance, when we stroke our fingers over a wooden table, apart from high stiffness, we also feel the texture and friction. Likewise, we distinguish plastic and metal spoons not only due to their weights but also considering the temperature flux, where the metal spoon feels colder. To achieve a high level of realism, the research in haptic rendering strives to design a model that ultimately reproduce the haptic interaction, by gradually modeling individual properties and incorporating them into a unified framework.

In VR simulation, haptic properties can be broadly classified into surface properties, which we feel using our tactile perception, and material properties, which we feel in form of force feedback. In both cases, we perceive the feedback with respect to certain actions. To achieve a high level of realism, the research in haptic rendering strives to design a model that ultimately reproduce the feedback for all possible input actions. The essential idea of *data-driven* and *measurement-based* modeling is to build a virtual copy of an object from action-feedback data pairs collected during real interaction.

12.1.2 Mixed Reality

Mixed Reality (MR) is an interactive environment where the real-world feedback and computer-generated stimuli superimpose one another. Depending upon the amount of the real and virtual content in the resultant feedback, the MR can be classified as *Augmented Reality* (AR) or *Augmented Virtuality* (AV) [39]. There is no specific rule clearly defining a boundary between AR and AV. However, taking into account the richness of real-world haptic feedback, the Augmented Reality configuration has been found more practical for applications with real-virtual mixed haptic content.

There are several abstraction levels in the design of AR applications. First, when we want to recreate the virtual object and its all haptic properties in a physical world. For instance, let's imagine that you want to buy a lamp from an online store. The AR techniques can render it on top of your table so you can touch it, press on the switches, etc. The second level is when one wants to alter a particular haptic property of a target object. For example, the pointiest is trying to switch from conventional paper-pencil sketching to a digital one using a stylus and tablet [5]. The artist might want to recreate the paper-pencil experience while drawing on the digital canvas of the tablet. Another practical example is medical training, where the size and stiffness of a tumor within a phantom body can be modulated to simulate various possible cancer cases [40]. All in all, the main strength of the AR system is that the designer can take advantage of the realism of the physical world and focus only on target properties. In the VR system, on the other hand, the designer should take care of all haptic properties.

Haptic properties of the physical world in AR simulation are either modulated by direct overlaying the synthetic stimuli or occluded by a physical barrier to be completely recreated. The former configuration, commonly known as a *feel-through* strategy, can be applied when it is possible to estimate the stimuli correcting the physical feedback. For instance, three-dimension force feedback during interaction with an object can be modulated to change its stiffness and friction [40, 41]. When then correcting stimuli is too challenging to estimate, the latter approach is more suitable. For example, direct modulation of haptic texture feedback having a stochastic nature still remains impractical. It is more convenient to cancel the physical stimuli using a special tool and then to recreate the complete feedback in a similar manner as in VR simulation [22]. In this chapter, however, we focus mainly on a feel-through approach, as our main goal is to deliver a software-related conceptual difference between AR and VR systems. Designing the hardware that occludes physical stimuli is beyond the scope of the current study.

12.1.3 Touch User Interfaces

Recent trends in *Human Computer Interaction* (HCI) interfaces show the transition from conventional interfaces with physical buttons and switches to digital ones simulated on touch screens. These interfaces typically miss the physical feedback deteriorating the user experience. Haptic rendering techniques are getting popular for simulation of physical interfaces. For instance, real-like feedback can be simulated for virtual buttons, knobs, and switches displayed on smartphones and tablets [51].

It is important to note that we consider only rendering aspects of simulation of physical interaction in Touch User Interfaces (TUI). The TUIs can also be used for abstract information encoding and transfer, e.g., Braille display [38] and vibrotactile patterns of notification or warning [33], which is outside of the current study's interest.

12.1.4 Structure and Contents

The focus of the current chapter mainly lies in data-driven haptic modeling and rendering techniques used for Virtual Reality. We first introduce a generic definition of the haptic model that relates the user's action to response stimuli in Sect. 12.2. Then we provide a series of examples addressing various aspects of modern haptic modeling and rendering. In Sect. 12.3, the interpolation-based data-driven method is introduced with deterministic input-output mapping. The data-driven haptic model that governs a mapping of user's action to stochastic response is presented in Sect. 12.4 along with the example of texture modeling and rendering. The physics-based haptic simulation was introduced in Sect. 12.5, where the Finite Elements Method was used to compute the deformation of a hyper-elastic object and corresponding non-linear force feedback. Finally, to model the plastic deformation, we introduce a hybrid approach of physics-based simulation with a data-driven controller (Sect. 12.6). Altogether, this chapter covers most modeling and rendering techniques that a novice haptics researcher might encounter.

12.2 Haptic Rendering

Exploration and perception of haptic properties, as it has been already discussed in Chap. 2, involves a complex cognitive process incorporating both *action* and *feedback*. For instance, we perceive the object's stiffness by relating the amount of object deformation and sensed force feedback. Thus the key component in the haptic rendering is a mathematical *model* or *algorithm* governing the action-feedback mapping. The rendering pipeline can therefore be expressed in three steps: sensing action, estimating and displaying the haptic feedback.

12.2.1 Haptic Model

Haptic model is a numerical method employing the action-feedback mapping. Haptic models can be generally classified into two groups, i.e., the *parametric* and *data-driven* methods. In parametric methods, the model with a fixed number of parameters is usually designed based-on rules, intuition, and empirical observations of underlying physical processes. Each parameter in these models, usually correlates with a certain material or haptic property of an object. One widely used example of parametric model is the Hook's elasticity model. In data-driven methods, the underlying physical processes of the object are neglected and the model of action-feedback relation is discovered directly from observations of a physical interaction. This approach is advantages in cases where the action-feedback relation is too complex or unclear like in texture rendering. Sometimes, similar problem can be modeled using both ways. For instance, the interaction with a fluid can be model by considering dynamics of its particles [18], or instant action can be directly map to the feedback bypassing the physics [35].

Haptic models can be designed in a closed- or open-loop setting. In close-loop rendering, the model output is continuously computed and displayed to the user. In open-loop simulation, the feedback stimuli is independent of the input action during the simulation, but computed and triggered based-on an action dependent event (event-based rendering). For example, object's hardness can be simulated by rendering contact vibrations, the pattern on which depends on the impact velocity [44]. The vibrotactile pattern of the click can be similarly rendered for digital buttons or switches [51].

12.2.2 Action

The action representing a haptic contact is usually expressed in a form of a vector having a finite set of variables correlated with the target response. For example, the input action of an elastic stiffness model can be described by a displacement vector representing the local deformation at the contact point [52]. In a physics-based simulation, where the object deformation is simulated by an external numerical solver, the action can be represented in the form of boundary conditions [9]. Tactile feedback from haptic texture is correlated with the velocity and contact pressure at the contact during the stroke over a surface. Hence the haptic texture model can be designed with two-dimensional action space [23]. The dimension of input space represents the degree of freedom of the model. By increasing the number of input variables, the model captures new characteristics of the interaction. For instance, to model the anisotropic texture, the authors in [4] included movement direction as an additional action variable. However, it is important to mention that with every additional input variable, the design and fine-tuning of the haptic model becomes much more difficult [2].

12.2.3 Response

The dimension of the response vector usually corresponds to degree of freedom of the haptic device. For instance, the response of the virtual wall contains a single variable repressing a force in a normal to wall direction. In order to model interaction with a virtual sphere a three-dimensional force output is computed. If the user interact with an environment through a virtual tool having arbitrary shape, an additional three-dimensional torque vector is required [12].

Rendering system is considered to be under-actuated, when the device is incapable to deliver the feedback for all response variables. In this cases, the model can be adjusted trying to compensate missing actuation or simplified to eliminate unnecessary calculations. For instance, if haptic device supports only one dimensional force actuation (e.g., force feedback for a finger of a haptic glove), a three-dimensional force vector computed during interaction with virtual object can be project into actuation dimension. In haptic texture rendering, taking into the account that the user does not perceive the direction of vibrations, the model response output of the model can be simplified from three- down to one-dimensional signal.

Several models with the same kind of feedback can share a single device for the output if they compute independent haptic properties. For example, the model of stiffness producing a force feedback in a normal to contact direction can be accompanied by a friction model with a force response in lateral direction. To simulate a surface texture, stationary vibrations can be added to a force feedback [21].

Depending upon the nature of the response stimuli, the response stimuli of a haptic model can be deterministic or stochastic. Training a stochastic data-driven model, which produces a continuously changing output for a constant input, is more complicated since in the current state of the art methods, the data should be segmented into either piecewise constant action [4] or piecewise stationary output [23] segments. Each segment of a stochastic model represents a single model point of multidimensional input space.

12.2.4 Data-Driven Modeling

In data-driven haptic modeling, the model governing the action-response relation is identified or trained exclusively using the experimental data collected during physical interaction with a real object or environment. Data-driven approaches generally omit the meaning of the underlying physics of the interaction and do not require manual design and tuning of the mathematical models. The data-driven modeling is advantageous where the action-response is non-linear and too complex for the manual design. Even the elastic object can exhibit high complexity and non-linearity due to its irregular morphology (shape). For instance, the feedback while deformation of the fork or spoon largely varies depending on the contact position and orientation (Sect. 12.3). In the case of texture modeling, it gets even more complex, as the feedback depends on many factors related to its surface properties and the applied action (Sect. 12.4). It is very challenging to capture all these factors in a manual model design, which greatly influences realism.

12.2.5 Measurement-Based Modeling

In measurement-based haptic modeling, the action-response mapping is usually governed using a parametric model. The set of parameters is usually identified using the data collected during interaction with a real object or environment. For instance, the linear stiffness model can be approximated using Hook's law. The friction, on the other hand, can be simulated using a Dahl formulation [41]. Since the parametric models usually consider the underlying physics of the interaction, the rendering stability is usually guaranteed theoretically. Therefore, the parametric model is often utilized for complex problems, where it is challenging to collect sufficient data for interpolation-based approaches. For example, to render a large deformation of a hyper-elastic object, the FEM model can provide a reasonable level of approximation as we discussed in Sect. 12.5. The approximation quality of physics-based models, however, is limited as the many factors are often not included in the model. To increase realism, the hybrid approach of data-driven and physics-based simulation can be utilized. For instance, in Sect. 12.6, we overview the rendering of plastic deformation, where the non-linear forces are computed using the FEM framework, and the plastic flow of the deformation is handled by a data-driven controller.

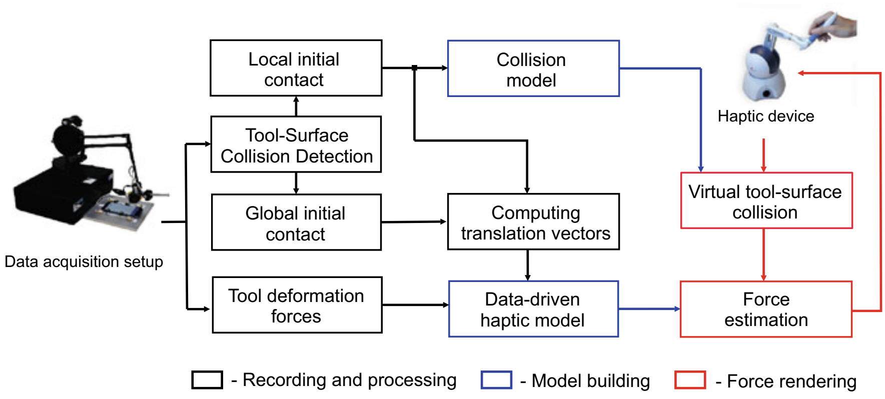


Fig. 12.1 General system overview

12.3 Deterministic Data-Driven Modeling

Deterministic models are generally utilized when the response stimuli of the haptic model remain the same for a given input action regardless of the time and history of previous actions. For instance, the deformation of an elastic object that exhibits a unique action-response correspondence can be approximated using a deterministic model [7]. In some cases, the short-term history can be used as a part of the model input. To model visco-elasticity, for example, the rate of deformation, i.e., the difference between immediate and previous deformation states, is required [35]. In other cases, when the current feedback depends on a long sequence of actions, e.g., in plastic deformation modeling, incorporating history into a model input is rather impractical and a physics-based modeling approach becomes inevitable [9].

Interpolation and regression techniques are considered to be the backbone of the deterministic models. *Interpolation methods* compute the feedback stimuli based on the neighboring data points collected during real interaction. The goal of *regression methods*, on the other hands, is to find the best fit of a parametric function for a given set of data points. The parametric function can be as simple as linear defined by a single parameter or as large and complex as deep neural network.

In this section, for input-output mapping, we utilize Radial Basis Functions Network (RBFN) interpolation method. This method has been found beneficial for haptic rendering due to its simplicity, efficiency and the ability to handle non-linear input-output mapping. As an example, we apply this approach for modeling a tool-deformation [7], that exhibits non-linearity due to its morphological complexity.

12.3.1 Tool Deformation Modeling

Tool-deformation modeling is a challenging non-linear problem. The morphological complexity of a tool like a spoon or a fork makes the force-displacement relation highly non-linear and anisotropic. Bending a spoon in different directions, for instance, requires a different amount of force. Additionally, the physics-based simulation for tool deformation is less practical, as the shape with a relatively thin and long body requires a high resolution of FEM tessellation. Therefore, the interpolation-based data-driven model is a good candidate to model the deformation of a tool.

To model the deformation of a tool, the RBFN-based data-driven approach can be utilized. The main objective of the current example, on the other hand, is to learn how to define the action and response spaces. By considering the action and response spaces of the model, we also need to design the data-collections setup that captures corresponding action-response pairs during the deformation of physical tools.

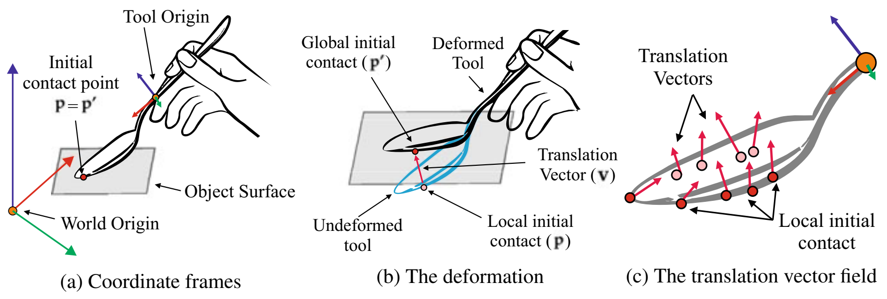

Tool Origin
Initial contact point $\mathbf{p} = \mathbf{p}'$
Object Surface
World Origin
(a) Coordinate frames
Global initial contact ($\mathbf{p}'$)
Deformed Tool
Translation Vector ($\mathbf{v}$)
Undeformed tool
Local initial contact ($\mathbf{p}$)
(b) The deformation
Translation Vectors
Local initial contact
(c) The translation vector field

Fig. 12.2 Descriptions of the model input space: **a** the input space is defined with respect to the origin of a tool; **b** a six-dimensional input vector consists the position of an initial contact **p** and a translation vector, **v** that describes the state of deformation; **c** A set of recorded input vectors are used in interpolation, and the force response is approximated at the tool's origin during rendering

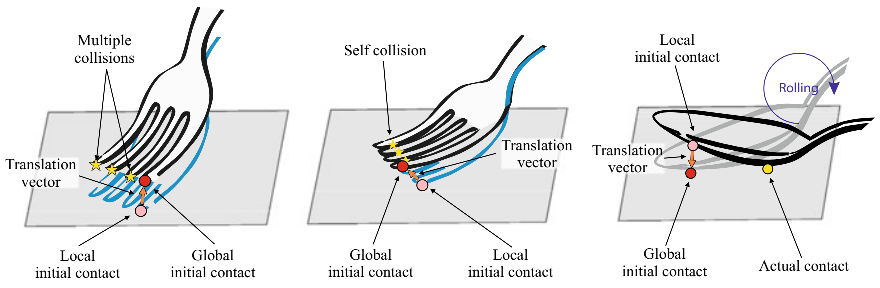


Fig. 12.3 Complex contact deformation: our data-driven model provides a non-linear input-output mapping that allows simulating the following deformations **a** multiple-contact; **b** self-collision; and **c** rolling-contact

12.3.2 Action and Response Spaces

To model contact deformation of the elastic tool, we define a six-dimensional input space. First three dimensions describe the position of the initial contact in tool's local coordinate frame, which is denoted as the *local initial contact* (**p** in Fig. 12.2a). **p** is determined at the moment of initial contact and remains constant in the local coordinate frame during a contact. The last three dimensions are related to the position of the initial contact that remains constant in the global coordinate frame, i.e., initial contact point on the object surface ($\tilde{\mathbf{p}}$ in Fig. 12.2b). We denote this as *global initial contact*. At the initial moment of the contact, both the local and global initial contact points represent the same point (Fig. 12.2a). However, when the tool begins deforming, **p** penetrates into the surface and moves away from $\tilde{\mathbf{p}}$ as illustrated in Fig. 12.2b. The difference between the two point can explain the state of deformation, and the difference vector $\mathbf{v} = \tilde{\mathbf{p}} - \mathbf{p}$ is referred to as *translation vector*. The translation vector becomes the last three input dimensions (Fig. 12.3).

The resultant input vector of the model is $\mathbf{u} = \langle \mathbf{p}, \mathbf{v} \rangle$. Taking into account that interaction happens in three-dimensional space, the final input space of the model can be expressed in six dimensions $\mathbf{u} = \left\langle p_x, p_y, p_z, v_x, v_y, v_z \right\rangle$.

It is important to notice that translation vector $\mathbf{v}$ differs from deformation vector used in the continuum mechanics (a vector representing the total movement of a particle on a deformed surface). In general, calculating this deformation vector needs the actual geometry of the deformed surface, which requires geometry recalculation of the tool deformation. This information is computationally expensive and is not generally available in a data-driven modeling scenario. Thus, we decided to avoid it. Instead, we utilize a vector representing a change of the initial contact point during deformation.

The model output is a three-dimensional force vector $\mathbf{f}$ at the tool's origin (Fig. 12.2a). The force response under a certain deformation should be explicitly determined by an initial contact point and the current position of the external force application, i.e., the encountered surface in our case. Thus, the initial contact point and the translation vector can fully explain the response force at the tool's origin. In our implementation, both of the inputs are presented in the local coordinate frame of the tool.

12.3.3 Data Acquisition and Preprocessing

We designed and assembled a manual recording setup that captures data from three sources (left side of Fig. 12.4). Three-dimensional force signal was captured by force/torque sensor (Nano17; ATI Industrial Automation, Inc., Apex, NC, USA) using NI DAQ acquisition board (PCI-6220; National Instruments, Austin, TX, USA) with a sampling rate of 1000 Hz. The position and orientation of the tool's origin were recorded by a haptic device (PHANToM Premium 1.5; Geomagic Inc., Rock Hill, SC, USA). In order to acquire the orientation of the tool, we design a custom gimbal encoder (right side of Fig. 12.4). The pitch and roll angles were measured by incremental encoders with angular resolution 0.045° (E2-2000; US Digital, Vancouver, WA, USA). The yaw angle was measured by a standard incremental encoder (OME-N; Nemicon, Tokyo, Japan) with angular resolution 0.18°, which was mounted by the manufacturer of the haptic device. The raw data from the gimbal encoder was acquired through the original 24-pin Mini Delta Ribbon (MDR) interface of the haptic device using the library (OpenHaptics 3.4; 3D Systems, Inc., Rock Hill, SC, USA). In order to compute the position and orientation of the tool's origin, we implemented the forward kinematics of the haptic device considering the angular resolution of the custom gimbal encoder.

The collision point between the tool and a flat surface was recorded using a capacitive touch screen of the smartphone (Galaxy S7; Samsung Electronics Co. Ltd., Suwon, Korea). In order to make the tool sensitive for the touch screen, we coated it with a liquid that is comprised of evenly dispersed ultra-fine conductive nano-particles (Nanotips Black; Nanotips Inc., Vancouver, BC, Canada). The position of the initial contact was recorded from the smartphone through the network. The

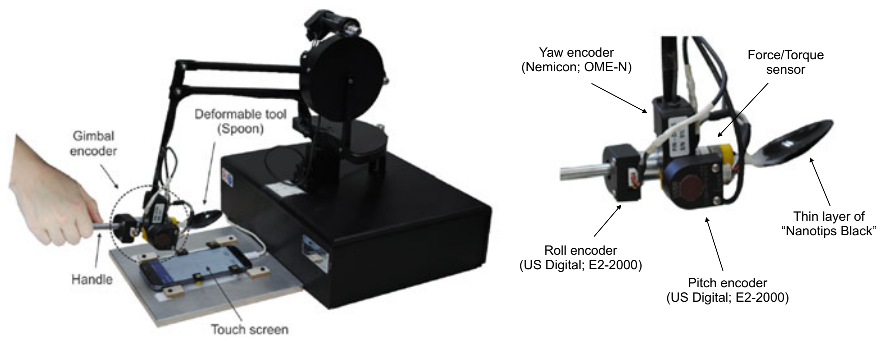

Gimbal encoder
Deformable tool (Spoon)
Handle
Touch screen
Yaw encoder (Nemicon; OME-N)
Force/Torque sensor
Thin layer of "Nanotips Black"
Roll encoder (US Digital; E2-2000)
Pitch encoder (US Digital; E2-2000)

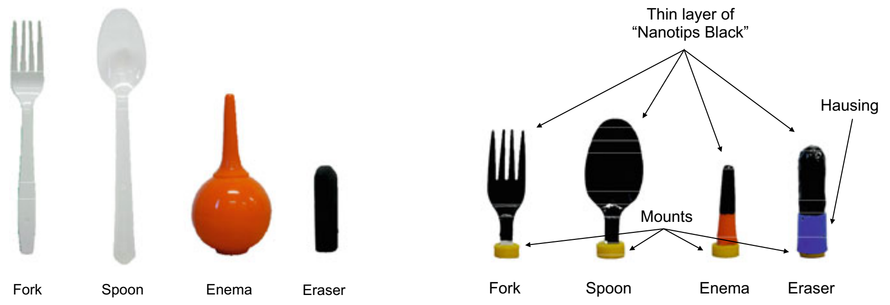
Fig. 12.4 Data recording setup. A user holds the handle of the device and presses the tool to surface of the touchscreen. Data from the tool deformation is recorded when the tool is in contact with the touchscreen. Left side - recording hardware; Right side - gimbal encoder

package delivery latency of the network was less than one millisecond. The position of the initial contact with respect to the world coordinate system (coordinate system of the haptic device) is stored and translated to the local coordinate system of the tool during the deformation. First translated initial contact point represents the *local initial contact* and each subsequent translated initial contact point represents the *global initial contact*. When the position and orientation of the tool's origin are changed, the global initial contact point moves away from the local initial contact. The vector pointing from the local to global initial contact points is a *translation vector*. The input vector of the proposed model $\mathbf{u}$ is a combination of the local initial contact and the translation vector.

To minimize the noise, force signals were filtered using a three pole Butterworth low-pass filter with a cut-off frequency 25 Hz. The cut-off frequency of the filter was selected accordingly to the human hand movement capability [11]. Similarly, the position data was smoothed using a third-order Butterworth low-pass filter with a with a cut-off corner frequency 25 Hz. Only the data points where the tool was in touch with the object were considered while other redundant data points were removed.

Each of four tools that we used for data collection (Fig. 12.5), was cut at the grip point, i.e., a point between index and thumb fingers while holding a tool. We refer to this point as tool's origin. Then, a 3D-printed adapter was attached to the edge of the cut for mounting the tools to the recording setup as shown in Fig. 12.5b. It is important to notice that the cut could cause mechanical changes in a tool's structure. However, the major contribution to the haptic feedback during tool-object interaction is provided by the deforming part of a tool. The upper part of a tool, which is on the other side of the grip point, is grasped in person's hand contributing negligible feedback.

In order to detect the contact with the touchscreen, each tool was coated with a thin layer of Nanotips Black (Fig. 12.5b). For the best performance, the coating layer was dried for 2 days. Hereafter, the modified versions of tool's presented in Fig. 12.5b are referred to as real tools.



(a) Original tools. (b) Prepared for recording.

Fig. 12.5 A set of real tools for evaluation: **a** illustration of original tools; and **b** modified versions of tools prepared for numerical and psychophysical experiments

12.3.4 Model Training

We develop a computational formulation that relates aforementioned input-output spaces. Data-driven model of the tool deformation can be understood as a function, whose parameters are optimized based on given observations, i.e., a set of input-output recordings from the real tool-surface interaction. The process of computing model parameters is referred to as *modeling*. During the simulation, the sequence of input vectors is fed into the resultant model while the model computes a continuous output of the force feedback.

In literature, there were several approaches proposed for input-output mapping. The most straightforward way is to utilize simplex-based methods [34] where the data are stored in a look-up table and approximated using weighted interpolation. The second way is to utilize feed-forward neural networks [1] that continuously compute a feedback output based on a given input during rendering. In this work, we adopted a radial basis functions network (RBFN), since it was successfully used in the majority of data-driven simulators of the object deformation [27, 35, 60].

An RBFN consists of three layers, i.e., input, hidden, and the output layer. The nodes of the hidden layer represent non-linear RBF activation functions. The input vector belongs to an n-dimensional Euclidean vector space, $\mathbf{u} \in \mathbf{R}^n$, and is transformed to a single scalar value of the output layer, $\phi : \mathbf{R}^n \rightarrow \mathbf{R}$, which can be described by

$$f_t(\mathbf{u}) = \sum_{j=1}^{N} w_{tj}\phi(\|\mathbf{u} - q_j\|) + \sum_{k=1}^{L} d_{tk} g_k(\mathbf{u}) \quad \mathbf{u} \in \mathbf{R}^n, \tag{12.1}$$

where w_{tj} is the weight constant, q_j is the center of the radial basis function, the function $g_k(u)$ $(k = 1, ..., L)$ forms a polynomial term, t is basis of the output space of the model, and $\phi(\| \cdot \|)$ is a radial basis activation function. Since cubic spline

$\phi(r) = r^3$ is chosen as the RBF kernel, the polynomial term is needed to ensure stability [37].

The weight constants w_t and polynomial coefficients d_t can be estimated by solving the following linear system:

$$\begin{pmatrix} \Phi & G \\ G^T & 0 \end{pmatrix} \begin{pmatrix} w_t \\ d_t \end{pmatrix} = \begin{pmatrix} f_t \\ 0 \end{pmatrix}, \tag{12.2}$$

where $\Phi_{ij} = \phi(\|u_i - u_j\|)$, and $G_{ik} = g_k(u_i)$ for $i, j = \{1, ..., N\}$, and $k = \{1, ..., L\}$, respectively. Since the RBFN provides only vector-scalar mapping, each basis of the force vector f_t is computed independently.

The desired weight vector w_t and polynomial coefficients d_t of the RBFN model can be calculated based on the inverse of the interpolations matrix, as follows

$$\begin{pmatrix} w_t \\ d_t \end{pmatrix} = \begin{pmatrix} \Phi & G \\ G^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} f_t \\ 0 \end{pmatrix}. \tag{12.3}$$

Since the size of the interpolation matrix is proportional to the square of the number of selected samples, it becomes computationally expensive to find the inverse of the interpolation matrix. It is important to notice that w_t and d_t are computed for three basis of the force vector independently. In order to compute force responses during rendering (Eq. (12.3)), three matrices should be provided, i.e., w, d, and q. A set of these matrices is referred to as *Haptic Model*.

During model building, the input vector can be directly derived from the sensor readings, i.e., data from the touch-contact sensor and the PHANToM's position encoder. However, during rendering, we do not employ the touch-contact sensor, so the initial contact positions and corresponding translation vector should be estimated.

In order to construct the input vector, the initial contact between virtual tool and object surface is required. This indicates that the shape of the object must also be provided to the rendering algorithm. One way is to use 3D mesh model of a tool for collision detection. However, this approach requires perfect reconstruction of the mesh model. Instead, we decided to build a *Collision Model* out of the local initial contacts from the training set. The collision model is a mesh model where its vertices are taken from the unique initial contact points measured for the haptic model. In order to build the mesh model out of arbitrary point, we utilize a 3D mesh processing software (MeshLab; ISTI-CNR, Pisa, Italy). The main benefit of such design is that the collision model perfectly matches the haptic model that ensures stability in rendering (Fig. 12.1).

12.4 Stochastic Data-Driven Models

Stochastic data-driven models approximate the feedback stimuli with a random nature. The distribution of the stochastic response signal is usually conditional on the user's action. The classic example of the stochastic data-driven methods is a haptic texture modeling, where the distribution of the vibrotactile response changes with respect to applied action, e.g., contact pressure, velocity and direction of the movement [2].

To approximate stochastic signals conditional to a variable input, an interpolation or regression methods are commonly utilized to convert the applied action to a latent representation, which in turn, parameterizes a model of a stationary random process. For instance, Romano and Kuchenbecker proposed to employ the bilinear interpolation of the vibrotactile signals that are encoded in the form Linear Predictive Coding (LPC) coefficients and stored in a look-up table of force-velocity action space [50]. This model was further improved in [23], by encoding acceleration patterns into auto-regressive moving average (ARMA) coefficients and by using Delaunay triangulation for interpolation of vibrotactile patterns. In [4], the vibrotactile patterns was interpolated using RBFN network allowing the arbitrary dimension of the action space of a model.

Another challenge in data-driven modelling of stochastic response is the segmentation of a signal into stationary vibrotactile patterns with relatively constant applied action. In [29], the authors proposed to segment acceleration patterns into sections with decaying waveforms. They assumed that the acceleration signal consists of decaying waves, where magnitudes of the local extrema are decreasing. The main drawback of this method was over-segmentation of the stochastic signal and under-segmentation of the patterned signal. These limitations were partially resolved in [30] by including a deadband threshold into segmentation criteria that constraints a starting point of new segments. Culbertson et al. proposed to use an AutoPARM algorithm for acceleration signal segmentation [23]. The AutoPARM algorithm finds an optimal partition of the signal by applying the evolutionary algorithm and minimizing the minimum description length (MDL) of each fragment [24]. In a similar fashion, the other algorithms estimating structural breaks of time series signals can be used for acceleration signal partitioning, e.g., AutoSLEX [20]. Assuming that the stationary acceleration corresponds to the relatively constant input, the input variables were averaged. In [2], the segmentation was performed in the action space where the input signals are partitioned into piecewise constant fragments. The optimal segmentation was achieved using bottom-up agglomeration strategy.

In this section, we introduce a new method allowing modeling of both isotropic and anisotropic textures through unconstrained tool-based interaction. To incorporate the directionality of the texture, we developed an action-space segmentation concept allowing modeling haptic textures with an arbitrary number of input variables. To store and interpolate vibrotactile signals, we developed a Radial Basis Function Network (RBFN) based haptic texture model allowing more general and flexible data-driven modeling. The complete pipeline of the haptic texture modeling and rendering is provided in Fig. 12.6.

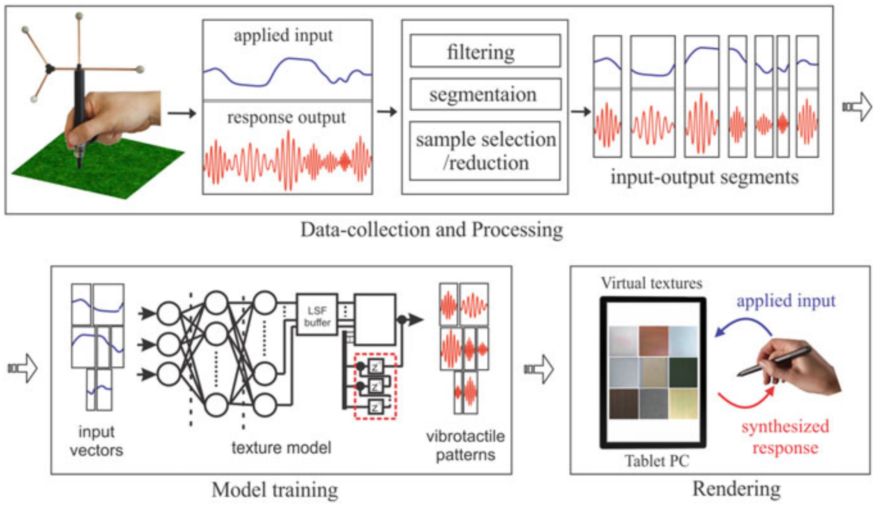


Fig. 12.6 Data-driven haptic texture modelling/rendering pipeline

12.4.1 Haptic Texture Modeling Pipeline

Surface haptic texture is one of the essential information for human to discriminate objects. While small-scale geometry variation is one of the main causes of haptic texture, human can effectively perceive fine details of the variation through not only a bare-hand interaction, but also tool-mediated stroking as high frequency vibrations. Sometimes, these small-scale geometry variations can be anisotropic: the characteristic of the vibration varies depending on the stroking direction. This direction-dependent haptic texture sometimes plays as a crucial cue for haptically identifying surfaces, e.g., identifying wooden surface based on directional grain, judging the quality of fabric using thread grain, etc.

Even though all haptic texture models have their own contributions, the conceptual representation of most models remain similar. The *model space* is an abstract coordinate system that describes the location of *model points*. Each model point is described by a location inside the model space and the feedback pattern $m_i = \{x_i, y_i\}$, where x_i denotes the $n-$dimensional vector of the model points location, and y_i represents the feedback pattern $y_i = \{a_1, a_2, ..., a_n\}$. Thus the general haptic texture model can be described by the set of model points $M = \{m_1, m_1, ..., m_p\}$. An example is provided in Fig. 12.7. Since the data-driven model in most cases is an interpolation model, a given minimal set of model points M_{min} is required for a stable output. In some case the M_{min} consists of the synthetic model points, which marks the interpolation boundaries.

The modeling algorithm of haptic textures consists of three stages: *data preprocessing and segmentation*; *model building*; and *rendering*.

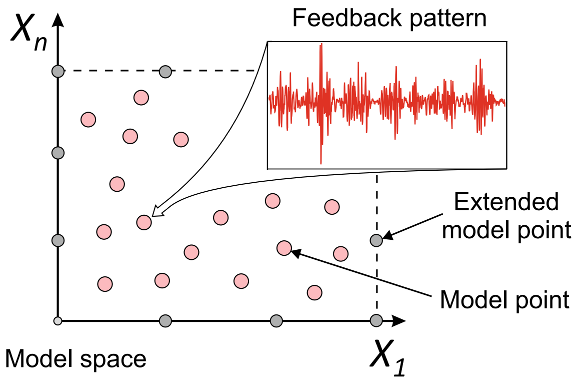


Fig. 12.7 General haptic texture model, ©2022, IEEE Reprinted, with permission, from [3]

12.4.2 Data Processing and Segmentation

In this section, we develop a generic segmentation algorithm that partitions a multivariate input signal. The segmentation algorithm searches the optimal partition, where the deviation of the input vectors within each segment is constrained by a set of constraint functions and corresponding thresholds. The set of constraint functions and thresholds is selected for a particular task, where a single pair of a constraint function and threshold can be applied for single or multiple input variables at the same time. To find the optimal partition of the input signal, we adopted a bottom-up agglomeration principle and employed it in three configurations, i.e., offline segmentation for single- and multi-trial data collection, and online segmentation of the streaming signal.

Problem Formulation

The main objective of the algorithm is to partition a multivariate signal X into a minimum number of segments $M = \{m_1, \dots, m_l\}$ having an arbitrary number of input vectors, such that the distribution of input vectors within each segment m_i must satisfy a given set of conditions $G = \{g_1(m_i) \leq \tau_1, \dots, g_p(m_i) \leq \tau_p\}$ where G is a set of inequality constraints. It is important to notice that each constraint can condition a single or multiple basis of the input space. Furthermore, any input variable can be conditioned by multiple constraints.

The preceding formulation imposes the restriction on distribution of each segment but can admit multiple solutions (several partitions can satisfy a given set of conditions). For example, the signal which initially satisfies the given set of constraints can be further partitioned until all segments become finest. Thus in order to find an optimal partition of the signal, the number of resultant segments should be minimized. In this manner, the signal segmentation task can be seen as an optimization problem.

$$\begin{aligned} \underset{n(M)\in\bar{Z}^+}{\text{minimize}} \quad & n(M) \\ \text{subject to} \quad & g_j(m_i) \le \tau_j \quad \text{for } j = 1,\dots,p \end{aligned} \tag{12.4}$$

where $n(\cdot)$ denotes cardinality of a set and $\bar{Z}^+ = \{1,\dots,n(X)\}$ is a finite set of positive integers bounded by the maximum number of samples in X. Below, we introduce the recursive constraint projection algorithm that segments a multivariate signal into a minimum number of segments satisfying a given set of constraints.

12.4.3 Bottom-Up Agglomerative Segmentation

The bottom-up algorithm breaks the input signal into a set of segments, where each segment contains at most δ_2 data points. Then, the merging cost e for each pair of neighboring segments is calculated using a cost function $\bar{f}$ and stored separately in set $E = \{e_1,\dots,e_{n-1}\}$. A pair of neighboring fragments with the lowest cost is merged, and the merging costs for the resultant and neighboring segments are updated. The agglomeration process is repeated until the lowest cost from the entire set E exceeds the predefined threshold $\bar{\tau}$. Detailed steps of the algorithm are provided in the form of pseudocode in Algorithm 1, where the function $initialPartition(\cdot,\cdot)$ partitions the input signal into a sequence of segments having δ_2 data points each

Algorithm 1 Bottom-up Agglomerative Segmentation

```
1: function SEGMENTDATA($\bar{M}$, $\bar{\tau}$, $\bar{f}(\cdot)$)
2:     $\bar{M}, n \leftarrow initialPartition(\bar{M}, \delta_2)$          ▷ $\bar{M} = \bigcup_{j=1}^{n} \bar{m}_j$
3:     for $j \leftarrow 1, n-1$ do
4:         $e_j \leftarrow \bar{f}(\bar{m}_j \cup \bar{m}_{j+1})$          ▷ Costs of pairwise merging
5:     end for
6:     while ($min(E) \le \bar{\tau}$ and $|E| \ne 0$ ) do
7:         $i = idx(min(E))$
8:         $\bar{m}_i \leftarrow \bar{m}_i \cup \bar{m}_{i+1}$
9:         $\bar{m} \leftarrow \bar{m} \setminus \bar{m}_{i+1}$
10:        $E \leftarrow E \setminus e_i$
11:        if ($i > 1$) then
12:            $e_{i-1} \leftarrow \bar{f}(\bar{m}_{i-1} \cup \bar{m}_i)$
13:        end if
14:        if ($i \le |E|$) then
15:            $e_i \leftarrow \bar{f}(\bar{m}_i \cup \bar{m}_{i+1})$
16:        end if
17:    end while
18:    return $\bar{M}$
19: end function
```

Generally, the optimization problem given by Eq. 12.4 is a Zero-one integer programming problem, which has been proven to be an NP-complete [42]. The Bottom-up Agglomerative Segmentation algorithm, on the other hand, belongs to the family of Branch-and-bound algorithms, which has been commonly used as the relaxation to the Zero-one integer programming problem [19]. Therefore, the proposed algorithm provides an approximation to Eq. 12.4. Although, the proposed approximation is computationally efficient which is crucial for real-time applications. Furthermore, due to the dynamic programming nature of the Bottom-up Agglomerative Segmentation algorithm, the resultant partition achieves the minimum number of segments balancing constraint function values across segments. This property of the Bottom-up Agglomerative Segmentation is very important as the resultant partition represents a set with evenly significant segments. Additionally, the bottom-up approach converges in finite number of steps. When the complete signal satisfies given constraints a single segment, the algorithm reaches the maximum number of operations (merging calls). The bottom-up approach also guarantees that all segments in the resultant partition satisfy a given set of constraints, as long as all finest segments from initial partition satisfy one.

Constraint Function Design

In order to evaluate the proposed algorithm, we developed two exemplary cases, i.e., data segmentation for modeling *isotropic haptic textures* and *anisotropic haptic textures*. Similarly, a user can define other task dependent constraints and apply them to an arbitrary multivariate signal. For instance, our approach can be used for other vibrotactile data-driven models, such as Virtual Reality bicycle [49], texture classification [54], and texture rendering on variable friction displays [47]. Furthermore, our segmentation algorithm can be applied for redundancy reduction in modelling non-linear force responses of visco-elastic object deformation [34], tool deformation [7], interaction with viscous fluids [35], where data segments with negligible variation can be substituted by representative vectors.

Isotropic Haptic Texture

In order to build the model of isotropic haptic texture, at least two input variables are required, i.e., normal force, and velocity magnitude [23]. The input stream $X(t)$ from the recording device generates two-dimensional input vectors $x_t = \langle f, v \rangle$ at each time step t. In order to build a reliable haptic texture, the input variables for each segment should be relatively constant.

The average deviation of the normal force within each segment m_i can be conditioned using the following constraint

$$f_1(m_i) = \sqrt{\frac{\Sigma_{i=1}^{N}(f_i - \mu_f)}{N}} \leq \tau_1, \tag{12.5}$$

where N denotes the size of the set, μ_f is a mean force for segment. The mean deviation of the velocity magnitude can be constrained as follows

$$f_2(m_i) = \sqrt{\frac{\Sigma_{i=1}^{N}(v_i - \mu_v)}{N}} \leq \tau_2, \tag{12.6}$$

where μ_v is a mean velocity magnitude per segment. Based on forgoing equations, the segmentation algorithm finds the optimal partition, where the average deviation of the normal force and velocity magnitude will be at most τ_1 and τ_2, respectively.

Anisotropic Haptic Texture

Anisotropic texture modeling requires partitioning of position data into segments with relatively straight movement trajectories. Thus, two additional input variables should be included into streaming signal $x'_t = \langle \mathbf{p}, f, v \rangle$ where $\mathbf{p}$ is a two-dimensional position vector. The maximum deviation of the position points from the line segment between starting and ending points can be computed as follows

$$g(m_i) = \left\| \frac{\|(\mathbf{p}_k - \mathbf{p}_1) \times (\mathbf{p}_k - \mathbf{p}_N)\|}{\|(\mathbf{p}_N - \mathbf{p}_1)\|} \right\|_\infty \tag{12.7}$$

where $k = \{2, \ldots, N-1\}$ and $\|\cdot\|$ means a vector norm. The foregoing equation can be used as a constraint function limiting the deviation of position points from the straight line (Fig. 12.8a). However, it does not prevent the change of direction close to the reverse movement, since the position points remain close to the line segment [4]. Therefore, an additional equation preventing loop-outs (change of the movement trajectory to reverse direction) is required.

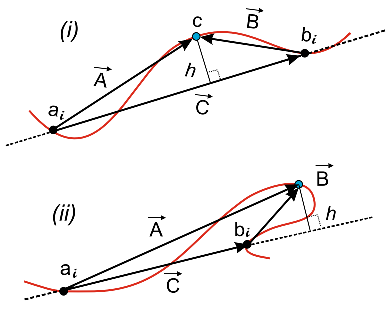
$$h(m_i) = \frac{\sqrt{\|\mathbf{p}_k - \mathbf{p}_1\|^2 - b^2} + \sqrt{\|\mathbf{p}_k - \mathbf{p}_N\|^2 - b^2}}{\|\mathbf{p}_N - \mathbf{p}_1\|}, \tag{12.8}$$

where b is a distance between point $\mathbf{p}_k$ and the line segment $(\mathbf{p}_N - \mathbf{p}_1)$. The function $h(\cdot)$ equals to unity while a segment of the movement trajectory does not contain loops, and exceeds one otherwise. Combining Eqs. 12.7 and 12.8, we can create a constraint function allowing segmentation of position data into relatively straight line segments.

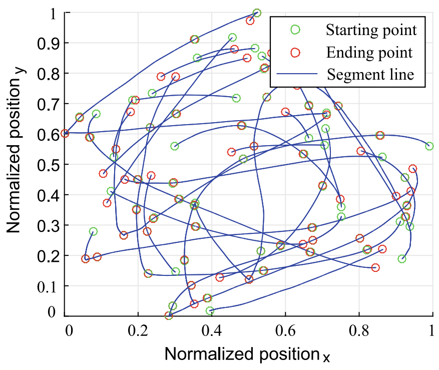
$$f_3(m_i) = \begin{cases} g(m_i), & \text{if } h(m_i) = 1 \\ inf., & \text{Otherwise} \end{cases} \leq \tau_3 \tag{12.9}$$

By using three constraints, the algorithm can find the optimal partition, where movement trajectories are approximately straight lines, whereas normal forces and velocity magnitudes are maintained relatively constant.

The overall segmentation procedure is culminated with elimination of segments that are shorter than 75 samples, which is assumed to be data in transition period with very high curvature. Note also that we skip merging of neighboring segments if a mutual mean of velocity or normal force magnitudes is equal to near zero. Number of data in these subsegments is normally very small, and they are removed from the set $\bar{M}$. It is reasonable since subsegments with near zero mean velocity or normal



(a) Segment constraints. (*i*) - Score estimation; (*ii*) - Loop-out prevention.



(b) Segmentation results of 10 seconds data.

Fig. 12.8 Bottom-up Agglomerative Segmentation and resultant partition

force can be assumed to be data from very beginning or very end of the contact, which normally has very small vibration. The segmentation result of 10 second data is shown in Fig. 12.8b.

12.4.4 Multi-trial Data Collection

In this section, we propose a novel algorithm for representative sample selection across multiple recording trials. The main aim of this algorithm is to populate the input space by significant model points from multiple trials while reducing the number of outliers. Furthermore, a generic haptic texture model is also provided. This generic model provides the necessary platform to other haptic texture modeling algorithms to benefit from the aforementioned sample selection algorithm.

Despite the fact that none of available sample selection algorithms can be directly applied to model point selection for data-driven haptic texture modeling, the idea of several sample selection algorithms can be generalized and extended for this task. For example, Edited and Condensed Nearest Neighbor (ENN [58] and CNN [31]) were initially designed for classification task based on k-Nearest Neighbors (k-NN) classifier. The former algorithm is usually used for outlier reduction, whereas the later one eliminates redundant samples from the given set. Recently, Arnaiz-Gonzalez et al. adopted the idea of CNN and ENN for regression tasks [10].

Inspired by the work in [10], we extended the idea of ENN and CNN for the representative model point selection for data-driven haptic texture modeling. Instead of using the k-nn classifier, the general haptic texture model can be used in our approach.

The pseudo code of the proposed method is depicted in Algorithm 2. The algorithm starts with the outlier reduction procedure (lines 1–11), which is followed by

Algorithm 2 Sample Selection Algorithm

```
Input: $M = \{(x_1, y_1), ...(x_n, y_n)\}, k, l, \tau_1, \tau_2$
Output: $\widehat{M} \subseteq M$
1: Removing outliers:
2: $\rho \leftarrow getAverageSparsity(M)$
3: for $i = k + 1$ to $|M|$ do
4:     $model \leftarrow train(M \setminus \{x_i, y_i\})$
5:     $\widehat{y_i} \leftarrow model.simulate(x_i)$
6:     $d \leftarrow getDistance(y_i, \widehat{y_i})$
7:     $\widehat{\rho} \leftarrow getLocalSparsity(m_i, M)$
8:     $\theta \leftarrow \tau_1 + \alpha * (\widehat{\rho}/\rho - 1)$
9:     if $(d > \theta)$ then
10:        $M \leftarrow M \setminus \{x_i, y_i\}$
11:    end if
12: end for
13: Removing redundant patterns:
14: $\widehat{M} \leftarrow \{(x_1, y_1), ...(x_k, y_k)\}$
15: for $j = k + 1$ to $|M|$ do
16:     $model \leftarrow train(\widehat{M} \cup \{x_j, y_j\})$
17:     $\widehat{y_j} \leftarrow model.simulate(x_j)$
18:     $d \leftarrow getDistance(y_j, \widehat{y_j})$
19:     if $(d > \tau_2)$ then
20:        $\widehat{M} \leftarrow \widehat{M} \cup \{x_j, y_j\}$
21:     end if
22: end for
```

redundant sample elimination (lines 12–21). The input of the algorithm consists of an initial set of model points $M = \{\{x_1, y_1\}, ...\{x_n, y_n\}\}$, where the first k elements form the minimal set of model points. Threshold values τ_1 and τ_2 are used to control the reduction rate of outliers and redundant model points, respectively.

Outlier Reduction

Outlier Reduction is an iterative process over the initial set M, where each model point $m_i = \{x_i, y_i\}$ is examined one at a time, starting from the $(k+1)^{th}$ element of the set. In each iteration, one model point is temporarily removed from the initial set $M \setminus (x_i, y_i)$. The resultant set is used for the model training. Following this, the feedback pattern $\widehat{y}$ is estimated by feeding the input vector x_i to the model. If the estimated $\widehat{y_i}$ and original y_i feedback patterns are considerably different, the probability that i^{th} sample is an outlier increases. This dissimilarity means that the contribution from the feedback pattern y_i contradicts to contributions of the neighboring ones. The dissimilarity between two feedback patterns is calculated by a dissimilarity metric, which is explained at the end of this section. The threshold value τ_1 denotes the level of dissimilarity, at which the model point is permanently removed from the set M.

This outlier detection strategy works well for dense regions, where the model point resembles to the neighboring ones. However, it can be misleading for sparse regions. The neighboring model points in sparse regions are usually different, since they are far from each other. Thus the threshold τ_1 should be adaptive to the local density of the model space. In order to solve this problem, the regularization term

$\alpha * (\widehat{\rho}/\rho_i - 1)$ is introduced, where $\widehat{\rho}$ and ρ_i are average and local sparsity of the model space respectively. When the local sparsity equals to the average one, the regularization term turns to zero. Similarly, when the local sparsity is higher then the global one, the adaptive threshold value θ is increased, and the other way around. The parameter α represents the sensitivity of the algorithm to the local density. It is recommended to estimate the α by using the following equation.

$$\alpha = \tau_1 * \frac{\widehat{\sigma}}{\widehat{\rho}}, \tag{12.10}$$

where the $\widehat{\sigma}$ denotes the mean deviation of local sparsity at each model point from the average sparsity $\widehat{\rho}$. In order to estimate the local sparsity ρ_i of each model point m_i for a two-dimensional model space, we built the Delaunay triangulation by excluding the target model point m_i, and computed the average distance from m_i to three enclosing neighbors. Similarly, the average distance to four surrounding model points of the tetrahedron represented the local density for a three-dimensional model space.

Redundant Sample Elimination

Unlike the previous stage, the process of redundant sample elimination starts with the minimal set, which contains only k model points. In every iteration, the haptic texture model is trained by using the set $\widehat{M}$. The set $\widehat{M}$ is extended by the candidate model point m_i, if the difference between the original and simulated feedback patterns exceeds the threshold τ_2. This iterative process finishes when all samples from M are assessed.

Error Metric

The error metric used for comparing the acceleration patterns is the spectral rms error. It is the difference between the approximated $\widehat{a}[n]$ and the recorded acceleration pattern $a[n]$. The equation for spectral rms error is given as:

$$E_n = e_n(\widehat{a}[n]) = \frac{RMS(F(\widehat{a}[n]) - F(a[n]))}{RMS(F(a[n]))}, \tag{12.11}$$

where RMS is the root mean square operator in the frequency domain, and $F(.)$ is the discrete Fourier transform. This error metric is preferred since it provides a better account of the perceptual differences as compared to the time domain error metrics.

12.4.5 Online Segmentation of Motion Primitives

In order to perform online segmentation, we introduced two contributions. First, to apply the given set of constraints, we developed a Recursive Constraint Projection algorithm. The segmentation process starts with the first pair of constraint function and threshold. The resultant segments from the first round are sub-segmented accordingly to the second constraint function, and the process continues until all constraints

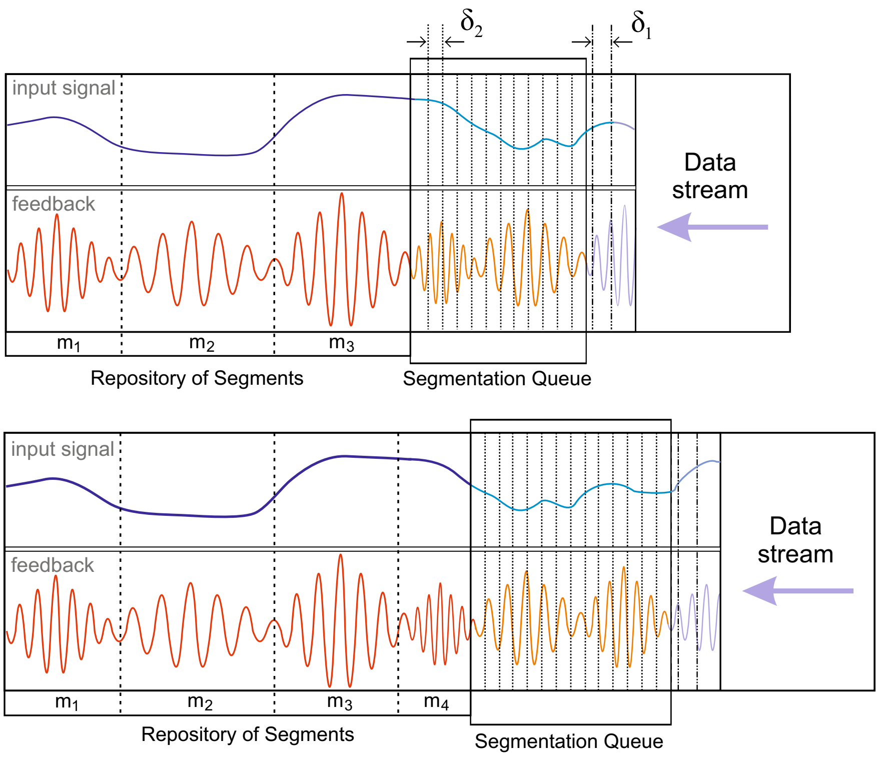


Fig. 12.9 Basic principle of the online segmentation schema. The example illustrates how the repository having three segments in (a) is extended by the fourth one sometime later © 2022, IEEE Reprinted, with permission, from [2]

are applied. The second improvement is an online segmentation schema. Inspired by the sliding window approach in [43], we developed a novel concept, where the streaming data is partitioned inside the segmentation queue (refer to Sect. 12.4.5 and Fig. 12.9).

Recursive Constraint Projection

The algorithm segments a multivariate signal X by recursively projecting it onto constraints from set G. The first constraint is applied to partition the complete signal, whereas each resultant segment is recursively sub-segmented utilizing the remaining set of constraints individually. The algorithm consists of two parts, i.e., a recursive function projecting multivariate signal onto the constraints and a bottom-up agglomerative data segmentation algorithm that finds an optimal segmentation for a given constraint.

The algorithm starts with a single element in the set of segments M where the only segment is represented by a complete multivariate signal $M = \{m_1 | m_1 \sim X\}$. Every call of the recursive function commences with pulling a single constraint $\bar{g} = \{\bar{\tau}, \bar{f}\}$

Algorithm 3 Recursive Constraint Projection

```
Input: $X = \{x_1, \dots, x_N\}$
    $G = \{(\tau_1, g_1(\cdot)), ...(\tau_p, g_p(\cdot))\}$
Output: $M = \{m_1, \dots, m_l\}$
 1: $M \leftarrow X$
 2: $M \leftarrow RecConstProj(M, G)$
 3: return $M$
 4:
 5: function RecConstProj($M, G$)
 6:     $\bar{\tau}, \bar{f} \leftarrow SelectConstraint(M, G)$
 7:     $G \leftarrow G \setminus \{\bar{\tau}, \bar{f}\}$
 8:     $\bar{M} \leftarrow \emptyset$
 9:     for all $m_i \in M$ do
10:         $\bar{M} \leftarrow \bar{M} \cup SegmentData(m_i, \bar{\tau}, \bar{f}(\cdot))$
11:         if $G \neq \emptyset$ then
12:             $M \leftarrow RecConstProj(\bar{M}, G)$          ▷ Recursive call
13:         else
14:             $M \leftarrow \bar{M}$
15:         end if
16:     end for
17: end function
```

from set G. Afterwards, each segment in M is partitioned satisfying the selected constraint $\bar{g}$, and resultant segments are stored in $\bar{M}$. If the set G is nonempty, the recursive function is called passing $\bar{M}$ and G. Thus, the depth of the recursion equals the initial number of constraints, and the signal is sub-segmented in every call such that the resultant segments satisfy a selected constraint. Regardless of which constraint is selected first, the signal eventually will be partitioned into segments satisfying all constraints. However, in order to reduce computational complexity, we introduce a constraint selection criteria (Algorithm 4).

Due to the recursive nature of the proposed algorithm, the number of calls on each next call equals the number of segments in the previous. It is reasonable to arrange the order of constraints in such a way that the constraints producing a lower number of segments are applied earlier. For instance, the same signal is partitioned using two different constraints producing two and four segments, respectively. Next round, for the former case, will require only two calls, whereas in the latter case the recursion will be called four times. Thus, for a case with greater number of constraints, this constraint selection strategy can considerably reduce the computational complexity. On the other hand, it is also computationally expensive to apply every constraint and select the one with the least number of segments. Therefore, we introduce an alternative measure $\psi_i = \bar{c}/\tau_i$ representing a merging cost of complete signal normalized by the threshold value of the constraint. The ψ_i is correlated with a number of resultant segments, meaning that lower value produces fewer segments and vice versa. The value ψ_i less or equal than one indicates that the signal on average satisfies the given constraint, nevertheless should be further partitioned to meet the condition locally throughout the complete signal.

Algorithm 4 Constraint Selection

```
1: function SELECTCONSTRAINT($M$, $G$)
2:     $C \leftarrow \emptyset$
3:     for all $g_i \in G$ do
4:         $\bar{c} = g_i(M)$                          ▷ Merging cost for complete $M$
5:         $\psi_i = \frac{\bar{c}}{\tau_i}$              ▷ $C = \bigcup_{i=1}^{|G|} \psi_i$, and $\tau_i > 0$
6:     end for
7:     $j = idx(min(C))$
8:     $\bar{\tau}, \bar{f} \leftarrow g_j$                  ▷ $g_j = \{\tau_j, f_j\}$
9:     return $\bar{\tau}, \bar{f}$
10: end function
```

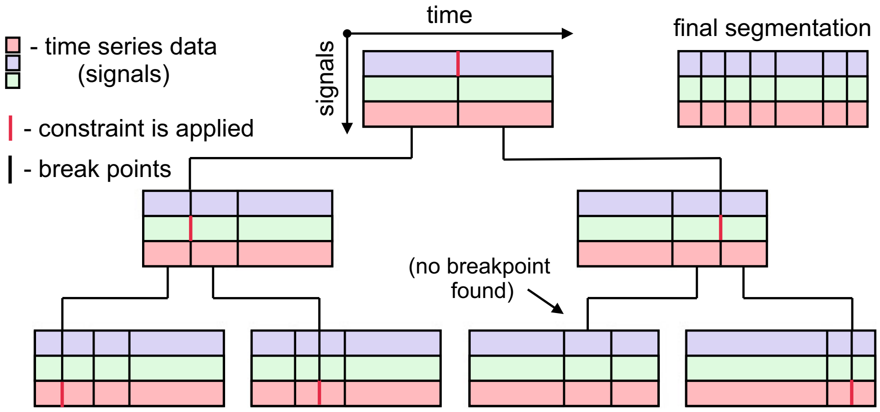


Fig. 12.10 Example illustrating the recursive segmentation. For simplicity, we assume that constraints were selected in order and the segments are always bisected (partitioned into two sub-segments with equal length) ©2022, IEEE Reprinted, with permission, from [2]

This selection criterion can be applied only if the threshold values of all constraints are strictly positive. Furthermore, the most of the commonly used energy and distance cost functions are non-negative and the thresholds are strictly positive. However, if the task requires to use thresholds that are equal or less than zero, it is recommended to follow the rules which are commonly used in coordinate descend optimization, i.e., selecting constraints one by one or randomly.

A simple example illustrating the recursive process of the segmentation is depicted in Fig. 12.10. Suppose that we have a three-dimensional signal where each variable has an individual constraint. First, the RCP algorithm selects an optimal constraint using Algorithm 4. Then, the complete signal is segmented by Algorithm 1 using the selected constraint. The similar process is repeated for two resultant segments using the remaining set of constraints. This recursive process continues until all constraints are applied. The depth of the recursion in this example equals three, which is the number of initially available constraints.

Online Segmentation Schema

At this point, we will explain the architecture of the online segmentation algorithm that finds the optimal partition of the streaming signal (Fig. 12.9). The algorithm performs segmentation over the segmentation queue. The incoming input vectors are buffered into fragments with length δ_1. Then, the buffered fragment is fed into the segmentation queue, which triggers the process of segmentation by passing the data from the segmentation queue and the set of constraints to recursive constraint projection algorithm. If there is more than one segment in the segmentation queue after partitioning, all except the last segment are stored into the repository. The last segment remains in the queue and used for segmentation in the next iteration since it can be a part of the future segment. Thus, the segmentation queue is always non-empty having a variable length, which makes our technique different from conventional sliding windows approaches [43].

The parameters δ_1 and δ_2 are required to reduce the processing time of the segmentation. When δ_1 is set to one, the segmentation process will be triggered at the sampling frequency of the input signal. Usually, in haptic modeling, the streaming frequency is very high and the change of the input state for one tick is negligible. Thus, it is reasonable to invoke the segmentation process with the step of δ_1 samples. Similarly, if the δ_2 is set to one, at the beginning of segmentation the signal will be broken into fragments with one sample. In such a case, the agglomeration process will take longer.

12.4.6 Interpolation Model

The goal of the interpolation model is to estimate the vibration output under a given input data sample based on interpolating captured data. We denote the input data sample as a 3D vector, $\mathbf{u} = \langle v_x, v_y, f_n \rangle$, where v_x and v_y are 2D tool velocity vector, and f_n is a normal response force. Since output of the interpolation model is a time-series high frequency vibration, it is more convenient to express it using a time-varying parametric model. For this, auto regression model (AR) are commonly used in data-driven haptic texture rendering, which we also use.

However, the coefficients of the AR model cannot be directly interpolated due to stability problem, which happens when poles of the transfer function H in Fig. 12.11 are not within the unit circle in the complex plane [26]. Therfore we convert AR coefficients into line spectrum frequency (LSF) coefficients for storing in the interpolation model as introduced in [23]. For rendering, we restore the AR coefficient from LSF model.

Another contribution of this dissertation is the use of a radial basis function network (RBFN) as an interpolation model for texture modeling. The RBFN interpolation model outperforms simplex based in two aspects. First, the output is computed using basic mathematical operations that makes it fast whilst the interpolation result remains good. Second, the input space can be easily increased. For example it is

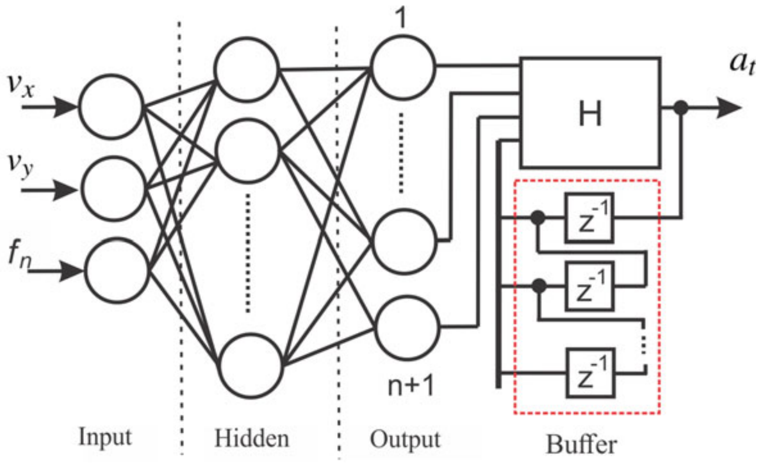


Fig. 12.11 RBFN architecture for model storage and interpolation

possible to store several different models inside of the network, switch them or even interpolate using additional input during rendering.

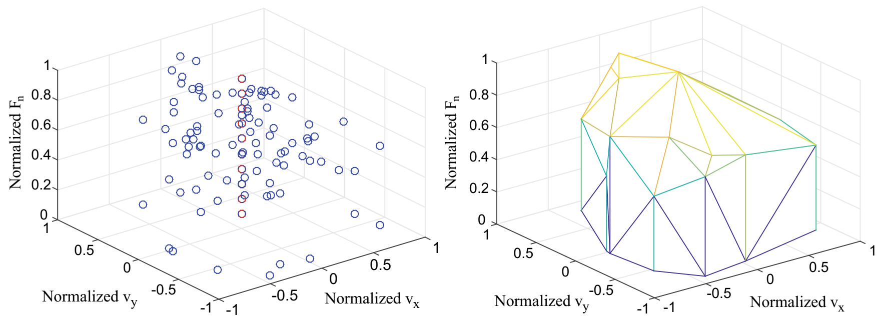
RBFN architecture that we used in this work consists of three layers (Fig. 12.11). The input of the network is a vector $\mathbf{u} \in \mathbf{R}^n$ of the n-dimensional Euclidean vector space. The nodes of the hidden one are non-linear RBF activation functions. The output of RBFN is a scalar function of the input vector, $\phi : \mathbf{R}^n \to \mathbf{R}$, which is described as

$$f(u) = \sum_{j=1}^{N} w_j \phi(\|u - q_j\|) + \sum_{k=1}^{L} d_k g_k(u), \quad \mathbf{u} \in \mathbf{R}^n \tag{12.12}$$

where w_j is the weight constant and q_j is the center of the radial basis function. The functions $g_k(u)$ $(k = 1, ..., L)$ form a basis of the space $\mathbf{P}_m^n$ of polynomials with degree at most m of n variables. Since we use the first order polyharmonic splines $\phi(r) = r$ as the kernel for RBF, the polynomial term is necessary. Otherwise, the interpolation results might be not as accurate as we want [37]. Using Eq. (12.12), a linear system can be obtained to estimate the weight constant vector $\mathbf{w}$ of radial basis functions as well as the polynomial coefficient vector $\mathbf{d}$, such that

$$\begin{pmatrix} \Phi & G \\ G^T & 0 \end{pmatrix} \begin{pmatrix} w \\ d \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix} \tag{12.13}$$

where $\Phi_{ij} = \phi(\|u_i - u_j\|)$, and $G_{ik} = g_k(u_i)$ of the range $i, j = \{1, ..., N\}$, $k = \{1, ..., L\}$.



(a) Model points. (b) A convex hull that limiting feature space.

Fig. 12.12 Input feature space of the haptic texture model

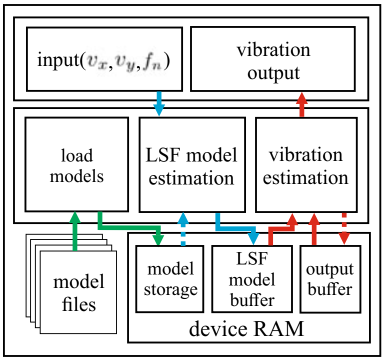
The input vector **u** is fed into three nodes of input layer. There are n outputs each of which corresponds to each LSF coefficient. One extra output is for interpolation of the variance provided by the Yule-Walker algorithm [32].

Once a set of LSF coefficients is obtained, the output vibration can be estimated in two steps. First, the estimated coefficients are converted to AR. Second, the vibration value is calculated applying Direct Form II digital filter along with n previous outputs. It is common way to use digital filter for AR signal estimation. The digital filter we used in this work can be replaced by any of a kind.

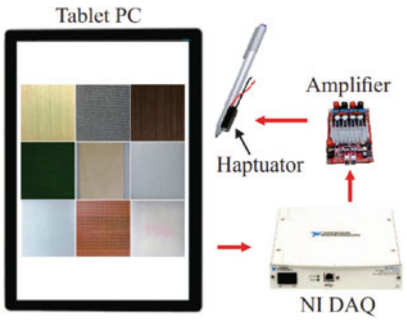
The RBFN is trained as follows. First, the representative input points are calculated from each segment by averaging data points in a segments after the zero-mean/unit variance normalization along each axis of **u** (Fig. 12.12a as an example). Second, in order to cover zero normal force area, we select points that lies on the convex hull of existing points and are facing the $\langle v_x, v_y \rangle$ plane. Then we project them onto the $\langle v_x, v_y \rangle$ plane (Fig. 12.12b). For the new points at the zero normal force, the LSF coefficients is copied from that of the original point, while the variances are set to zero. In case of zero velocity, new model points are uniformly created and scattered along f_n axis, whose variance is set to zero, and the LSF coefficients are copied from the closest model points. Lastly, using the model points, we trained RBFN applying a SpaRSA algorithm [59] that identifies sparse approximate solutions to the undetermined system Eq. (12.13) using an extended set of features from the previous step.

12.4.7 Real-Time Texture Rendering

In this section we will describe a setup for anisotropic texture rendering algorithm. The setup consist of software and hardware components. The software component is implemented in form of a computing library to make it independent from the hardware. The software architecture of the computing library is depicted in



(a) Software architecture of the computing library.



(b) Hardware setup for algorithm demonstration.

Fig. 12.13 Rendering setup and architecture

Fig. 12.13a. The hardware setup that will be used for rendering is shown in Fig. 12.13b.

The architecture of the rendering software consists of three layers. The upper layer is referred to as interactive layer. This layer computes the input vector **u** based on readings from the input device. Additionally, this layer displays response vibrations back to the user. The business logic of the anisotropic haptic texture library is represented by the second layer. This layer is developed in form of a platform independent computing library. The computing library consist of three functional blocks. First block loads the set of haptic texture models into device memory. The second block estimates the LSF (line spectrum frequency) coefficients and the variance by feeding the input vector **u** to the RBFN haptic texture model. The LSF coefficients and the variance are updated inside the buffer in 100 Hz. Meanwhile, the output vibration signal is generated by the third block, which runs on the other computing thread having frequency 2 kHz. The output vibration signal is produced based on buffered LSF coefficients, the variance and m buffered vibration outputs, where m is a number of LSF coefficients. Note also that all functional blocks work in separate computing threads. The frequency of each computing thread can be reset in accordance with user needs.

Vibration Estimation

The set of LSF coefficients and the variance describe the contact vibration pattern for a given vector **u** inside the input space of the RBFN haptic texture model. Therefore, the main task of the RBFN haptic texture model is to provide the mapping of three-dimensional input vector **u** with corresponding $(m + 1)$-dimensional output vector (m LSF coefficients and the variance). This output vector can be calculated using following equation

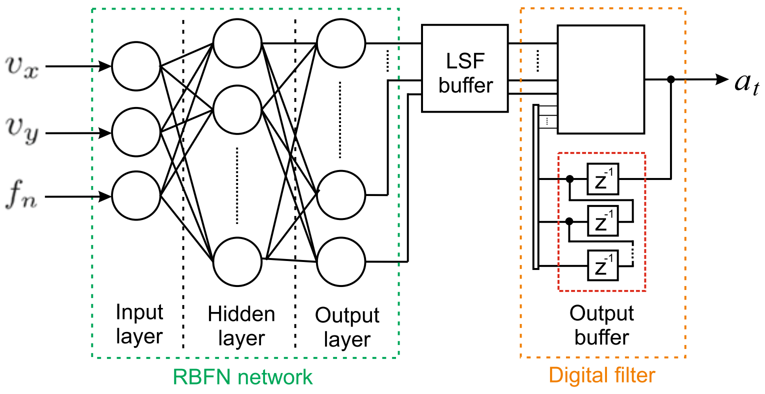


Fig. 12.14 Model architecture for action-dependent vibrotactile signal synthesis

$$f_i(u) = \sum_{j=1}^{N} w_{ij}\phi(\|u - q_j\|) + \sum_{k=1}^{L} d_{ik}g_{ik}(u), \quad \mathbf{u} \in \mathbf{R}^n \tag{12.14}$$

where $i = \{1, ..., m + 1\}$ denotes the index of LSF coefficients and the variance, w_{ij} is a weight constant and q_j is a center of the radial basis function. The functions $g_k(u)$ $(k = 1, ..., L)$ form a basis of the space $\mathbf{P}_p^n$ of polynomials with degree at most p of n variables.

The output vibration values are calculated using an approach similar to [23]. First, the LSF coefficients are converted to AR ones. Second, the AR coefficients, variance, and m buffered outputs are fed to the transfer function of the Direct Form II digital filter

$$H(z) = \frac{\varepsilon_t}{1 - \sum_{k=1}^{p} w_k z^{-k}}, \tag{12.15}$$

where w_k are AR coefficients, ε_t is a random sample from a normal distribution. The output value of the transfer function (Eq. 12.15) is the output acceleration value. Therefore the overall rendering algorithm of the stochastic vibrotactile signal can be decomposed into two computing threads as in Fig. 12.14.

Rendering setup

In order to demonstrate the quality of the modeling and rendering algorithms, we designed a tablet-PC-based hardware setup (Fig. 12.13b). The tablet PC (Surface Pro 4; Microsoft) was selected as a rendering device. The contact velocity is calculated based on the contact position data from the touch screen of the tablet PC. The normal force of the contact is calculated based on readings from active digital pen (Surface Pen; Microsoft) with a sensing capability of 1024 pressure levels.

The output vibrations are displayed using NI DAQ data acquisition device (USB-6251; National Instruments). This output signal is amplified by an analogue amplifier and is displayed using a voice coil actuator (Haptuator Mark II; Tactile Labs).

12.5 Physics-Based Modeling

In the physics-based simulation, the haptic properties of an object or environment are approximated by an external mathematical model of the underlying physical process of the interaction. In haptic rendering, physics-based modeling is commonly used to simulate the global deformation of an object. The volume of an object is usually discretized into a finite set of mass points, the dynamics of which is approximated by Newton's laws of motion. The relation among neighboring mass points is modeled by constitutive models governing strain-stress relation. This relation can be approximated by the mass-spring-damper system or Finite Element Method (FEM). The solution of the former approach is estimated by a system of ordinary differential equations (ODEs). The latter approach is based on partial differential equations (PDEs) is considered to be physically more accurate. In the physics-based simulation, the haptic properties of an object or environment are approximated by an external mathematical model of the underlying physical process of the interaction.

In this section, our goal is to model a hyper-elastic object deformation using FEM based approach, where the stress-strain relationship derives from a strain energy density function. This approach allows modeling large deformation of relatively soft objects, which is challenging to approximate by other simulation methods. A complete set of methods covering the whole process of the measurement-based modeling/rendering paradigm is newly designed and implemented deformation phenomenons, with a special emphasis on haptic feedback realism.

12.5.1 Hyper-Elastic Material Modeling

In this section, we establish an easy and standard procedure for identifying material parameters of hyper-elastic object and corresponding real-time rendering algorithm. While real-time simulation and digitization of large deformation for visual feedback have been a mature research topic [15, 48, 57], those for haptic feedback are still in their early stage. This is mainly due to that global deformation usually involves changes of geometry throughout the whole object body which is very expensive to simulate in so-called 1 kHz "haptic-real-time" and has nearly infinitely large input and output space for measurement-based approach. What is worse is that for haptic simulation, changes of particles inside the object matter. Due to these difficulties and high realism requirement of these applications, the Finite Element Method (FEM)-based approach is considered as the most suitable direction, which this paper follows. In the FEM-based approach, physical deformation of an object's continuum media is approximated using discretization methods and is commonly used for deformation modeling, where the stress-strain relationship of the deformation is governed by a constitutive model and material parameters. However, there are yet two critical hurdles to overcome to apply the FEM-based approach to the haptic digital copy and rendering scenario. First, there is no well-defined way to tune the FEM parameters rendering the behavior of the virtual copy to be exactly the same as the one of an existing real object. Second, it is still not quite feasible to use FEM in haptic rendering due to rather slow update rate of even the state-of-the-art algorithms. Our goal is to tackle these two problems.

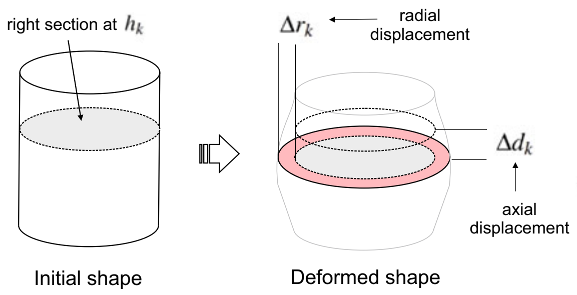
We focus on identifying a single set of FEM parameters through the palpation of an object with homogeneous and isotropic material where a single constitutive model and the same material parameters can describe the whole elements within the object. This identification procedure can be repeated for multiple objects, yielding a material library, which can be used to design a heterogeneous or composite object with multiple different materials. We put building material library as our future work.

Our approach of identification follows the conventional procedure; the parameters are estimated by observing the object's shape change in responses to well-defined external force application. In order to facilitate the capturing procedure, our framework assumes an object, from which the material parameters are extracted, having a cylindrical shape. Unlike other volumetric primitives, the cylindrical object has a beneficial property that simplifies deformation capturing. By fixing the bottom of the cylinder and applying the orthogonal force to its top, the shape of the cylinder expands away from its central axis symmetrically. This property allows capturing the deformation at a particular level of the cylinder using several tracking markers. Additionally, the simplicity of the cylindrical shape allows a user to prepare material samples for the identification (Fig. 12.16).

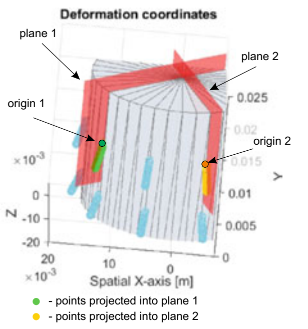
12.5.2 Deformation Features

During compressive deformation, the shape of a cylindrical object expands outwards, whereas the symmetry of deformed shape about cylinder axis is maintained. Thus the right section of a cylinder (Fig. 12.15a) travels downwards and the area of the right section increases. Relatively to its initial shape, the deformation of cylinder at h_k height level can be represented by axial Δd_k and radial Δr_k displacements (Fig. 12.15b).

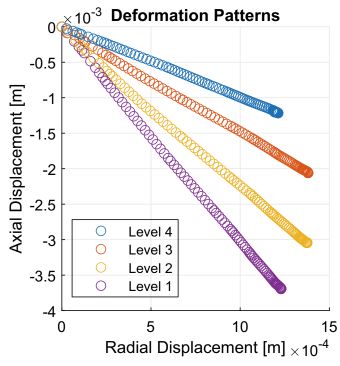
At every time step, the data collection setup recorded a state of the deformation. A complete session of the deformation consists of n states, where each i-th state is represented by compressing displacement Δs_i (a distance that top end of cylinder traveled downwards during deformation), normal force f_i, and sixteen three-dimensional position points of markers $\mathbf{p}_{ij}$. The absolute marker positions p defined in the world coordinate system are variant to translations and rotations. Therefore, to use the $\mathbf{p}_{ij}$ position points in material identification, the coordinate system of the data collection device and that of the FEM simulation should be aligned. As the miss-alignment of



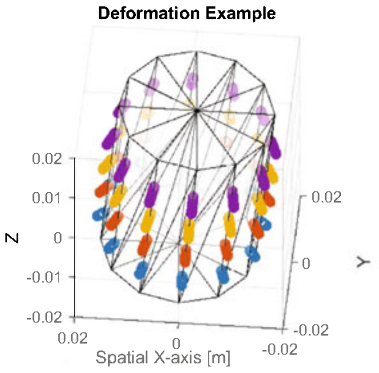
(a) Deformation parameters.



(b) Coordinate frame of axial and radial displacements.

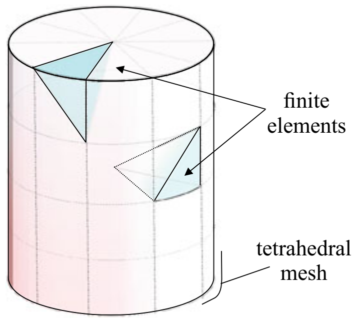


(c) Estimated deformation patters at four height levels h_k.

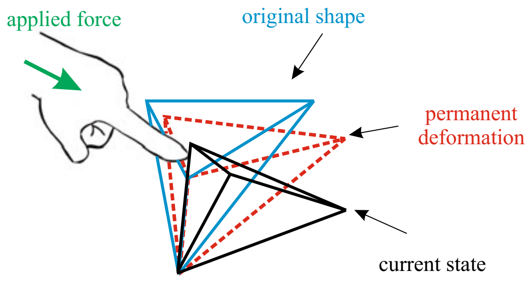


(d) Reconstructed deformation using estimated deformation patterns.

Fig. 12.15 Deformation parameters definition, deformation feature construction, and deformation pattern extraction ©2022, with permission from Elsevier [9], all rights reserved



(a) Tetrahedral mesh.



(b) Plastic deformation of the finite element.

Fig. 12.16 Elasto-plastic deformation of the tetrahedral mesh

the coordinate systems might degrade the model identification quality, using absolute positions is inappropriate. In order to build translation and rotation invariant deformation features, the positions $\mathbf{p}_{ij}$ are transformed into more convenient coordinate frames, describing relative displacements of each marker. The origin point p_{0j} and the plane passing through the central axis of the cylinder and fitted to the position points $\mathbf{p}_{ij}$ describe the coordinate frame of the displacements for the marker j (Fig. 12.15). The axial Δd_{ij} and radial Δr_{ij} displacements are coordinates of position points $\mathbf{p}_{ij}$ projected on to the plane of the coordinate frame of the displacements. At every height level h_k of the cylinder, the radial and axial displacements are averaged and stored into matrices $\mathbf{D}_{i\times k}$ and $\mathbf{R}_{i\times k}$, which define our deformation features. The deformation patterns of four levels of the cylinder can be seen in Fig. 12.12c. The deformation of the cylinder at any height level h_k can be reconstructed using deformation patterns as shown in Fig. 12.15d. Thus, the deformation of a cylinder can be represented by two matrices. Note that the deformation features do not describe the whole deformation dynamics. To describe the whole deformation dynamics, along with deformation features, the normal forces $\mathbf{f}$ and the compressing displacements $\mathbf{d}$ of all states are needed as well.

12.5.3 Model Identification

Our goal is to estimate model parameters based on observations collected during deformation of a real object. In order to optimize material parameters, i.e., Young's modulus k and Poisson ratio v, we define a FEM solver as a function $\Gamma(\cdot)$ mapping vector of compressing displacement $\mathbf{d}$ to the synthesized normal forces $\tilde{\mathbf{f}}$ and shape deformation patterns, i.e., $\tilde{\mathbf{D}}$ and $\tilde{\mathbf{R}}$.

$$\{\tilde{\mathbf{f}}, \tilde{\mathbf{D}}, \tilde{\mathbf{R}}\} = \Gamma(k, v, \mathbf{d}) \tag{12.16}$$

Then the model identification can be seen as a nonlinear optimization problem with a following objective function

$$\min_{\mathbf{k},\mathbf{v}}(||\mathbf{f} - \tilde{\mathbf{f}}||_2^2 + \alpha(\frac{1}{r_c}||\mathbf{D} - \tilde{\mathbf{D}}||_F^2 + \frac{1}{h_c}||\mathbf{R} - \tilde{\mathbf{R}}||_F^2)), \tag{12.17}$$

where h_c and r_c are reference height and radius of the cylinder, respectively. The h_c and r_c are required to make objective function invariant to the physical dimensions of the cylinder. The parameter α is used to balance the force error with the error of deformation. In our case we selected α equals 0.1.

Note that any FEM solver being able to describe deformation of the target object can be utilized in a function $\Gamma(\cdot)$ for model identification. However, it is recommended to use the same solver for both modeling and rendering. In this work, we adopted the FEM solver with implicit integration schema based on Alternating Direction Method of Multipliers (ADMM) [48]. This solver having the generic formulation allows to use of any constitutive model by defining a proximal operator (Sect. 12.5.4). We employed two commonly used nonlinear hyper-elastic material models, i.e., St. Venant-Kirchhoff and Neo-Hookean models.

In order to solve the nonlinear optimization problem (equation Eq. 12.17), we utilized a single objective Genetic Algorithm (GA). We first created an initial population of 50 two-dimensional genes. Then we ran optimization with 0.7 crossover and 0.1 mutation rates. Other gradient-free optimization algorithms can also be used for a given objective.

12.5.4 Finite Elements Method Solver

The second goal of the work is to make FEM simulation run fast enough for haptic rendering. In this section, we integrated the optimization-based FEM solver from [48] into a haptic rendering environment. Here, we first provide a brief background required for the contact forces computations. Next we explain how the actual contact forces are computed and rendered in our setup.

In FEM modeling, a deformable object is understood to be a set of material points having individual masses m_i, which are interconnected with each other forming a tetrahedral mesh. Each tetrahedron of a mesh can be treated as a generic spring that keeps mass points at the equilibrium state by raising conservative forces. External forces applied to the deformable object cause a motion of mass points, which in terms obeys the Newton's second law. Thus, in order to approximate the motion of mass points, one can perform explicit or implicit time integration. The implicit method has been found to be more practical for real-time applications providing a stable approximation for relatively large time steps, whereas the explicit methods tend to overshoot the equilibrium point and explode. The implicit time integration method, the backward Euler method, is computationally intractable for real-time haptics since it requires to solve a large non linear system of equations. However, for the conservative system $f = -\nabla U$, the backward Euler method can be formulated as an optimization problem [28]. The main advantage of this formulation is that the parallelizable solvers can be utilized. Here, we provide a brief summary of the framework. For detailed explanation, refer [48].

The optimization problem for the state of a deformable object at Δt time step later can be formulated as,

$$\mathbf{x} = \arg\min_{\mathbf{x}}\left(\frac{1}{2\Delta t^2}||\mathbf{M}^{\frac{1}{2}}(\mathbf{x} - \tilde{\mathbf{x}})||_F^2 + U(\mathbf{x})\right) \tag{12.18}$$

where $\mathbf{M}$ is inertia matrix, $\tilde{\mathbf{x}}$ is the predicted state of the deformable object in the absence of implicit forces, and $U(\mathbf{x})$ is elastic potential energy of a deformable object at Δt time step later.

The first and second terms of the objective function (Eq. 12.18) represent the momentum and the elastic potentials respectively. The optimization problem itself can be seen as finding the equilibrium between the momentum and the elastic potentials. In order to solve the optimization problem, ADMM-based solver splits equation (12.18) into two objective functions by introducing a dual variable and a constraint function, which relates these objective functions. In this way, each objective function is optimized separately satisfying constraint relating both. As a result, the solution satisfying the both of new objective functions converges to the solution of Eq. (12.18) in iterations. The dual variable $\mathbf{u}$ introduced by ADMM is also updated at each iteration. As the deformable object is modeled as FEM, elastic potentials can be calculated locally for each mesh element. Based on [48], the dual variable $\mathbf{u}$ correlates with implicit conservative forces caused by elastic potentials, and can also be updated locally for each tetrahedral element.

Elastic Potential (local-step) During the collision with a haptic probe, the internal deformation undergoes in elastic object raising conservative forces. The deformation gradient $\mathbf{F}(\mathbf{x})$ of whole mesh object can be calculated by its current state $\mathbf{x}$. The strain of each tetrahedral element can be defined separately by its deformation gradient $\mathbf{F}_i$. Integration of elastic potential energies of all tetrahedral elements constitutes the elastic potential energy of full mesh.

$$U = \sum V_i \Psi(\mathbf{F}_i). \tag{12.19}$$

Here, $\Psi(\mathbf{F})$ is strain energy density function which can be calculated from deformation gradient and material parameters, V_i is the volume of the tetrahedron element.

In local step, the ADMM-based solver searches updated deformation gradient values minimizing the elastic potentials and approaching optimal gradient deformation which is $\mathbf{F}_i + \mathbf{u}_i$.

$$\mathbf{F}_i^{n+1} = \arg\min_{\mathbf{F}}(\Psi(\mathbf{F}) + \frac{\tau}{2}||\mathbf{F} - (\mathbf{F}_i^n + \mathbf{u}_i)||_F^2) \tag{12.20}$$

Note that the dual variable $\mathbf{u}_i$ is also updated in this local-step.

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^n + \mathbf{F}_i^n - \mathbf{F}_i^{n+1}. \tag{12.21}$$

Momentum Potential (global-step). At global step the solver iteratively updates the state of whole mesh. ADMM algorithm introduces the following optimization problem that searches solution for minimizing momentum potentials satisfying the solution of optimal elastic potentials:

$$\mathbf{x}_{n+1} = \arg\min_{\mathbf{x}}(\frac{1}{2\Delta t^2}||\mathbf{M}^{\frac{1}{2}}(\mathbf{x} - \tilde{x})||_F^2 + \frac{1}{2}||\mathbf{W}(\mathbf{F}(x) - \mathbf{F}^n + \mathbf{u}^n)||^2) \tag{12.22}$$

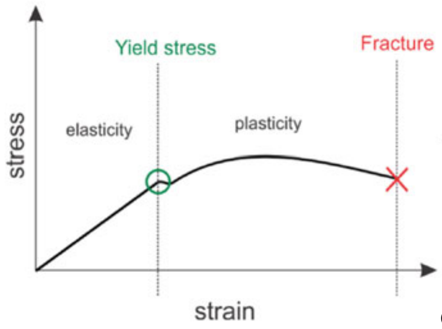
Here, $\mathbf{W}$ is a weight matrix which affects the convergence rate. Linearized constraints $\mathbf{Cx} = \mathbf{d}$ about the current state are also incorporated into 12.22 to handle the collisions. The solution satisfying the constraints and above optimization problem is found by introducing Lagrange multipliers λ in a saddle point system and solved using Uzawa Conjugate Gradient for General Constraints [55]. The self-collision can be handled in the similar manner as in [48]. However, in the current study, the self-collision was not considered to reduce the computational load. Note that the algorithm is optimized globally for the whole mesh, so we call this update rule global-step. The whole algorithm converges to the optimal solution of the problem (12.18) by updating the local- and global-steps alternatively in iterations. The interaction forces can be computed using Lagrange multipliers, which explain in Sect. 12.6.7.

12.6 Combination of Physics-Based and Data-Driven Models

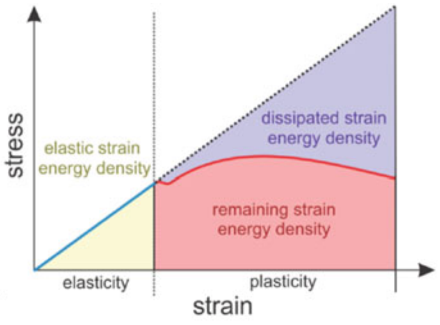
As we discussed earlier, the simulation approach, whether it is physics-based or data-driven, is usually determined by considering the nature and complexity of the target stimuli. The physics-based simulation is usually preferred when the global deformation is in priority. For instance, the physics simulation is beneficial when the object undergoes large or plastic deformation, when the visually plausible rendering is also required, and when the interaction is complex with self and multi-point collisions. The data-driven methods, on the other hand, are usually used when realistic modeling of the local contact is in priority with a highly non-linear action-feedback relation. Many haptic modeling problems, however, fall on neither side of this trade-off. In such cases the hybrid approach of physics-based and data-driven can designed. For instance, to simulate FEM, the authors in [46] computed deformation forces at eight points around the haptic probe positioned on a virtual grid and performed a real-time interpolation at haptic update rate. Bickel et al. [14] proposed to adjust the strain of each element of the linear FEM model with non-linear function. In this section, we explain the hybrid approach of modeling the elastoplastic deformation, where the non-linear forces are computed using a hyper-elastic model from Sect. 12.5 and the plastic flow is approximated by a neural network-based controller. The goal of this section is to demonstrate the joint optimization of physics-based and black-box data-driven models.

12.6.1 Plasticity Modeling

The ability to portray three-dimensional shapes by sculpting malleable materials played an important role in human evolution. Plastic modeling has been actively used in pottery, molding, architecture, and sculpture. Modeling the desired shape from a pliable material like clay or dough requires a special set of sensory-motor skills, which can be developed only through haptic interaction. In order to achieve a target deformation, the artist fine-tunes the contact manipulations relying upon kinesthetic



(a) Yield point.



(b) Strain energy of elastic and plastic components ©2022, IEEE Reprinted, with permission, from [10].

Fig. 12.17 Elasto-plastic deformation curve and strain energy density

perception of the plastic flow and force feedback. Due to the lack of realistic haptic feedback, learning the plastic and pastry arts in online or by using virtual reality (VR) simulation remains rather impractical. The realistic haptic feedback might also be beneficial for the digital sculpting in modern computer-aided design (CAD), which in turn shares common manipulation operations as in physical sculpting. In this dissertation, we aim to develop an end-to-end framework that captures material properties from plastic objects, builds the corresponding digital copy, and renders it in a haptic-enabled simulation.

Plasticity is a physical property of a material undergoing permanent deformation due to external forces. The permanent changes generally occur when the applied stress exceeds the material-specific yield point, i.e., transition point when the material behavior changes from elastic to the plastic regime (Fig. 12.16). The total deformation at the plastic regime, however, consists of both elastic and plastic components, as the material partially recovers after load removal. The plastic part of the deformation can be determined by the additive or multiplicative decomposition model. The multiplicative decomposition rule has been proven to be more appropriate since it preserves the physical meaning while keeping the object's volume persistently [36]. In computer graphics, the multiplicative decomposition is usually governed with a parametric model with manual tuning.

The multiplicative decomposition rule has been successfully used in computer graphics rendering. The decomposition is generally governed by a flow model with several parameters, e.g., yield point, flow rate, hardening parameter. In computer graphics, this parameters are generally manually tuned to achieve photo-realistic rendering. However, in haptic rendering, to achieve a realism, the accurate force feedback can be computed using multiplicative decomposition decomposition models allowing modeling plastic flow with arbitrary complexity.

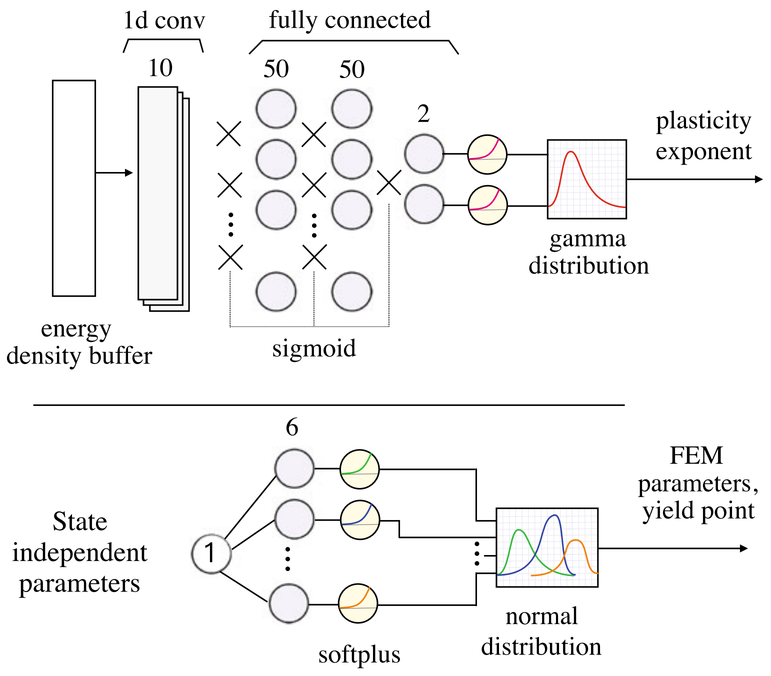


Fig. 12.18 Plasticity (top) and physics (bottom) policy models ©2022, IEEE Reprinted, with permission, from [6]

12.6.2 Elasto-Plastic Decomposition

The object undergoing a plastic deformation typically passes through four stages, i.e., elastic regime, yielding, plastic regime, and fracturing (Fig. 12.17a). Before reaching the yield point, the object recovers to its original shape after removal of all external forces. The system before the yield point remains conservative and can be approximated by optimization-based numerical integration that we already introduced in Sect. 12.5.4. When the deformation reaches the yield point, some strain energy start dissipating into the other forms of energy, e.g., turns to a heat during particle dislocation. The remaining energy is elastic causing partial recovery after the load removal and producing the force feedback during the contact (Fig. 12.17b). In nature, there is no any perfectly plastic solid material, as the minimal elastic potential is needed to withstand the shape against the gravity. Another important criterial is rate of dissipation of the strain energy in the plastic regime, as the underlying physical processes of turning one energy into the others require different time. Therefore, we should consider at least two factors to approximate the plastic flow. Note that in this study, we don't consider factors related to environment, e.g., temperature, humidity, etc. As long as these factors do not change rapidly, we can build models for different environmental conditions.

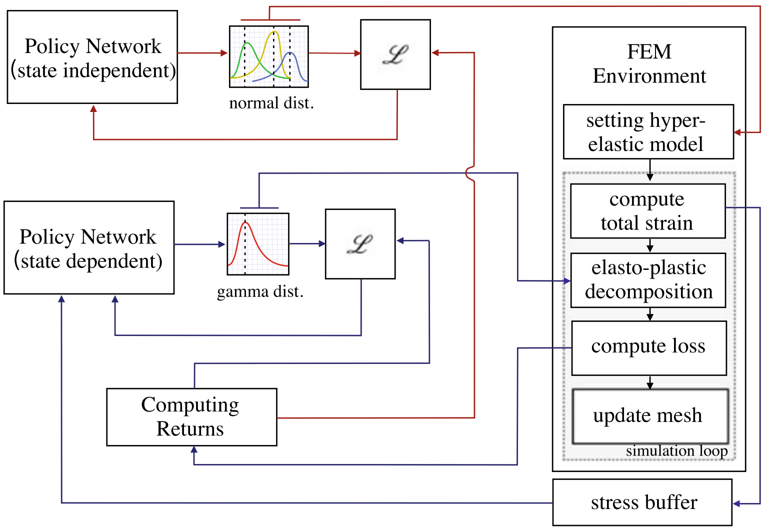


Fig. 12.19 Model identification pipeline. The blue and red lines of the pipeline represent the data flow for the update of state-dependent and -independent parameters, respectively ©2022, IEEE Reprinted, with permission, from [6]

The elasto-plastic decomposition is a continuous process where the total strain $\mathbf{F}_i$ from (12.18), is split into elastic and plastic components. In multiplicative decomposition, the amount of plastic component can be determined relying upon the following rule

$$\mathbf{F}_i = \mathbf{F}_i^e \, \mathbf{F}_i^p. \tag{12.23}$$

To find the strain of the permanent deformation, we first diagonalize the total strain, $\mathbf{F}_i = \mathbf{V}\Sigma\mathbf{V}^T$, where Σ is a diagonal matrix with eigenvalues, as in [16]. To prevent changes of the tetrahedral volume, we constrain the determinant of the diagonalized strain Λ to 1, as follows

$$\hat{F}_i^p = (det(\Lambda))^{-1/3}\Lambda \tag{12.24}$$

Then, the plastic strain can be estimated using plastic flow constitutive model

$$\hat{F}_i^p = (\hat{F}_i^p)^\gamma, \tag{12.25}$$

where the exponent γ is usually a function relating the current stress with the ratio of the plastic component in the original strain $\mathbf{F}_i$. For instance, if γ always equals zero, then the material is elastic. Likewise, if constant γ equals one, the plastic component occupies the complete strain. We approximate the γ for each finite element as follows

$$\gamma = min\left(\nu\frac{(\Psi(\mathbf{F}_i) - e_y)}{e_y}, 1\right), \tag{12.26}$$

where $\Psi(\mathbf{F}_i)$ is the energy density function, e_y denotes the yield point, and ν controls the rate of plasticity.

The rest configuration of the mesh model $\mathbf{X}$ can then be updated using the plastic strain that we obtained in Eq. (12.25).

$$\mathbf{X} \leftarrow \mathbf{X}V(F_i^p)^{-1}V^T \tag{12.27}$$

As it can be seen in equation Eq. 12.26, we compute the plastic flow γ directly using the strain energy density. This is advantages for the haptic rendering as the energy density is already computed in ADMM routines and we do not have to compute the the second Piola-Kirchhoff stress as in [13]. However, since we update the mesh object after ADMM iterations, we have to perform Cholesky factorization of a matrix used in a global step of ADMM solver [48]. This requires some additional computation, which can also be done in parallel using GPU.

12.6.3 Data-Driven Modeling of Plastic Flow

Identification of the physically-based dynamic models is a non-linear problem that is commonly solved by meta-heuristic methods, e.g., the genetic algorithm [8, 45]. The objective used in optimization usually requires running a complete simulation for each candidate set of parameters. Thus the training of the black-box controller with a relatively large number of parameters becomes computationally intractable using these methods. To tackle this problem, we propose a novel approach based on inverse reinforcement learning, that optimizes a complex controller by taking advantage of the intermediate steps of simulation.

Reinforcement learning (RL) is a machine learning technique optimizing the control model while interacting with a dynamic environment. The controller in RL changes the state of the environment by executing an action and receives the reward. The main goal in RL is to optimize the state-action mapping function that maximizes cumulative reward. In inverse RL, the reward is derived from observations of an expert acting in an environment. In the case of plasticity modeling, we derive the reward function based on data collected from real deformation and identify a control model that tries to mimic the plastic flow.

The main requirement for modeling the homogeneous object is that the same controller should be able to approximate the plastic flow for any finite elements of the mesh no matter its size and location. The main difficulty is that the deformation measurement (force-displacement field) inside the physical object is infeasible and there is no practical way to compute the reward for each finite element. To address this issue, we propose a multi-agent single-policy reinforcement framework, where each finite element is individually represented by an *Agent*. The agent observes a deformation *State* of the corresponding finite element, which we represent in the form of a vector with recent energy densities. For a given deformation state, the agent executes the action, which is the exponent of the multiplicative elasto-plastic decomposition γ_i. The common policy model is optimized by all Agents in a Markov Game. The execution of simultaneous actions by multiple agents, however, interferes and becomes non-stationary since the next state that each agent observes is also conditional to previous actions of others. To mitigate this problem, we design the inter-agent cooperation in a relaxed form, where the stochastic action is executed by a single agent for a complete run of the simulation (trajectory). The idle agents apply the action from the fixed policy. In this way, we limit the variation of the reward at each time step to an action executed by a particular agent.

Considering the symmetric property of the cylindrical specimen, finite elements can be classified into several groups having similar topology and experiencing similar stress during deformation. The tetrahedral mesh of 63 elements that we used in material identification, thereby can be partitioned into nine groups. At each iteration, one out of nine groups can be randomly selected to sample the next game trajectory. The system reaches the Nash equilibrium point when the agents don't have to move towards improving the policy.

12.6.4 Policy Model

The RL algorithm is commonly classified into Value- and Policy-based methods. In our model, we employ the policy-based concept that performs a direct search over the policy space. The policy function samples continuous actions from distribution conditional to the observed state and parametrized by function approximators like deep neural networks. The main advantages of the policy-based methods are that they directly learn stochastic processes and allow using a gradient-based optimizer. This is beneficial for training deep neural networks using the backpropagation technique.

The material model in our framework consists of state-independent and -dependent learn-able parameters (Fig. 12.19). The state-independent parameters represent material-specific properties that do not change during the deformation. In this model, the state-independent parameters were encoding the normal distributions of Young's modulus and Poisson ratio used for total strain energy computation, as well as the yield point denoting the elastic limit. The state-dependent parameters are used to control the plastic flow in elasto-plastic decomposition by establishing the relation between the total density and the ratio of the plastic component in current deformation. We approximate the plastic flow using a deep neural network depicted in Fig. 12.18. The input of the model is a vector accommodating recent strain energy densities of a particular finite element. The 1D convolutional layers help to compute rate dependant features encoding the possible viscosity, as well as for filtering undesired oscillations. The fully connected layers map incoming feature vectors to the parameters of the gamma distribution. The plastic flow exponent can be sampled from the resultant gamma distribution in the training process, or represented by its mean in rendering.

12.6.5 Model Training

The model identification pipeline is depicted in Fig. 12.19. We identify the material model in iterations by alternating the optimization of physics and plastic flow parameters. In the first step, we sample a number of vectors with FEM parameters and run the simulation for each vector. In this step, the plastic flow was taken as a mean of distribution characterized by a fixed policy model. Likewise, in the second step, we randomly select one out of nine groups and sample state-dependent trajectories using the plasticity model, while keeping state-independent parameters constant. After each step, we update corresponding model parameters using Proximal Policy Optimization (PPO) algorithm [53]. The PPO, like other policy-based algorithms, tries to increase the probability of the action producing a higher return, and penalize the probabilities of actions leading to a lower return,

$$\tilde{p} = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} R^{\pi_\theta}(s_t, a_t), \tag{12.28}$$

where $R^{\pi_\theta}(s_t, a_t)$ is the infinite-horizon discounted return function computed at time step t, s_t is a vector of recent strain energy densities of a finite element representing its state, and $a_t = \{\gamma_i, k, v, e_p\}$ is an action. The PPO, however, additionally penalizes large update steps by clipping the probability ratio $\tilde{p}$, as follows

$$\mathcal{L} = \min\left(\tilde{p},\ clip\left(\tilde{p}, 1-\varepsilon, 1+\varepsilon\right)\right). \tag{12.29}$$

To estimate the ongoing reward, we adopted the objective function that we used for identification of hyper-elastic parameters in Sect. 12.5.1.

$$r = -||\mathbf{f} - \tilde{\mathbf{f}}||_2^2 + \alpha(\frac{1}{r_c}||\mathbf{X}_a - \tilde{\mathbf{X}}_a||_F^2 + \frac{1}{h_c}||\mathbf{X}_r - \tilde{\mathbf{X}}_r||_F^2), \tag{12.30}$$

where $\mathbf{f}$ denotes contact forces, $\mathbf{X}_a$ and $\mathbf{X}_r$ represent axial and radial projections of cylinder deformation, respectively Fig. 12.20.

Note that we don't apply the actor-critic schema used in the original PPO algorithm. Since each agent in our environment has a different influence on the reward, the expected return should be computed for each agent by an individual critic network. Instead, to reduce the variance, from ongoing reward we subtracted the expected reward computed for a mean action of current policy distribution.

12.6.6 Recording Setup and Sample Set

To capture the non-linear force response during the deformation of an elastoplastic sample, we build a motorized data collection setup (Figs. 12.21 and 12.22). The

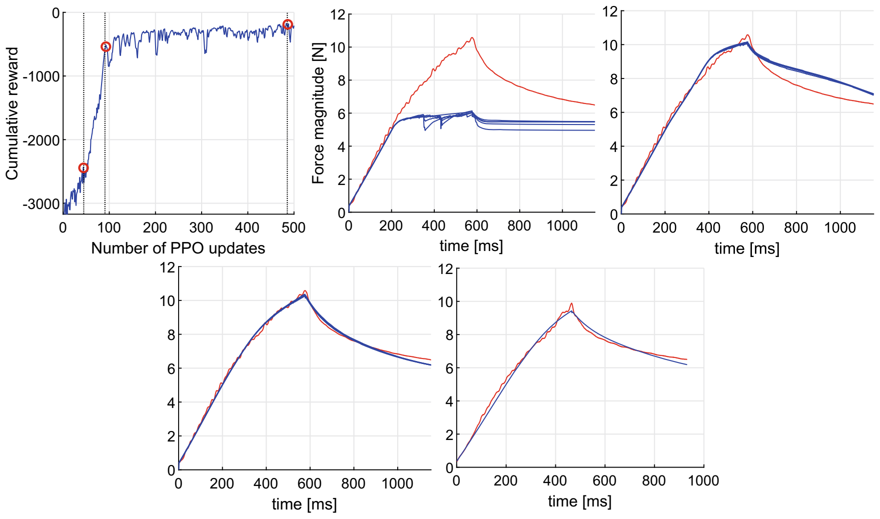


Fig. 12.20 Model identification progress: cumulative reward for 500 steps of PPO update; three snapshots of during training; force response for testing data ©2022, IEEE Reprinted, with permission, from [6]

device enables a uni-directional position and velocity control of the carriage for the compressive deformation of a cylindrical material sample. The carriage compressing the sample was equipped with a force sensor (Nano17; ATI Technologies) and was aligned in a normal direction by two rail rod sliders. The movement of the carriage was actuated by a stepper motor (17HS8401; NEMA 17), which was managed by the TMC-2130 controller in 1/16- microstepping mode. The motion of the carriage was smoothed by *stealthChop* algorithm with 1/256-microstepping interpolation (configured in the controller). The revolving resolution of the motor was 3200 steps per complete cycle, which corresponds to 0.025 mm resolution of the carriage's linear motion. In order to capture the shape deformation, we attached a grid of 15 IR markers to target samples and utilized four IR cameras (Flex 13; OptiTrack) for tracking. The force sensor was connected to the same data-acquisition board, which recorded force responses with 1000 Hz update frequency. For the evaluation, we prepared three elastoplastic material samples, i.e., dough, clay, and chewing gum. Two mounts were attached to both ends of each sample allowing fixating it in the device.

To identify material parameters, we performed the relaxation test with a constant strain rate. This test consists of two steps, i.e., loading and relaxation phases. In the loading step, the material sample was compressed with a constant velocity, 10 mm/s. In the relaxation step, the position of the carriage is fixed, and the decaying force feedback was collected for the same duration as in the loading step. For the training and testing datasets, the samples were compressed 5 mm and 6 mm, respectively.

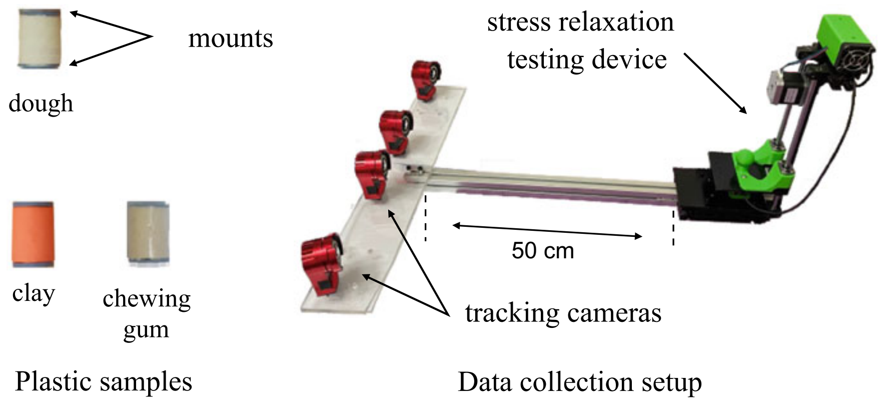


Fig. 12.21 Data-collection setup and sample set of plastic objects ©2022, IEEE Reprinted, with permission, from [6]

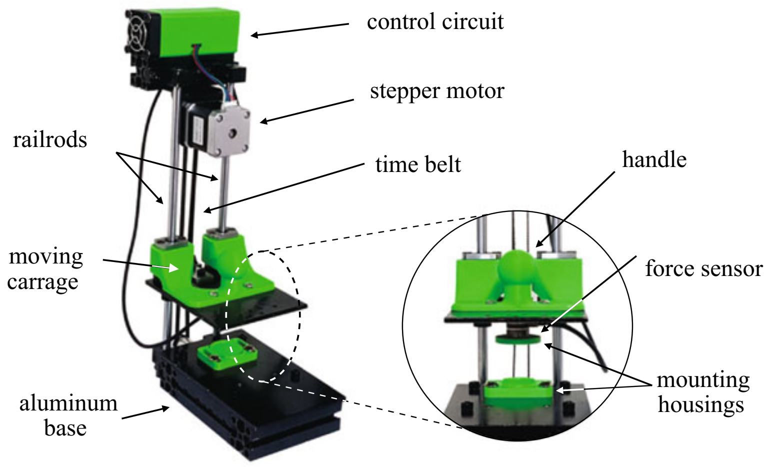


Fig. 12.22 Force-displacement measurement device of vertical deformation of cylindrical samples

12.6.7 Rendering Collision Forces

During the contact, the global deformation raises conservative forces in the object's media, such that the sum of all internal and external forces equals zero. In the virtual environment, a virtual tool coupled to a haptic manipulator applies external forces to contacting vertices of an object. However, impedance type haptic devices having a closed-loop control provide the position and orientation of the end-effector and accepts the force to be rendered. The contact deformation, in this case, can be described by a set of boundary constraints for contacting vertices $\mathbf{C_j}$. The response

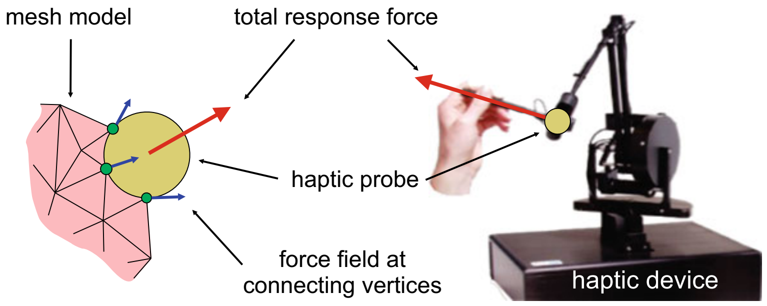


Fig. 12.23 Illustration of the force field at the contacting vertices and resultant force vector during contact © 2022, with permission from Elsevier [9], all rights reserved

force due to the contact can be computed as a sum of normal forces raised at the *m* contacting vertices (Fig. 12.23) due to boundary constraints as follows

$$f_r = -\frac{1}{\Delta t^2}\sum_{j=1}^{m} \mathbf{C_j}\lambda_\mathbf{j}, \tag{12.31}$$

where λ is Lagrange multipliers from the saddle point system.

To set the boundary constraints, all object's vertices are tested for a collision with a virtual tool using Axis-Aligned Bounding Box (AABB) collision detection algorithm. In our case, the virtual tool was in the form of a sphere. Depending upon the application, the shape of a virtual tool can be selected arbitrarily. The global positions of the colliding vertices are moved to the boundary of a virtual tool by forming equality constraints. The equality constraint is generally not recommended for haptic rendering. During a shallow and sliding contact, some vertices can provide oscillation by getting in and out of the contact. To solve this issue, we set a small dead-band allowing vertices to travel out of the object without the loss of the contact and employed the virtual coupling compensating the small contact oscillations.

12.7 Conclusion

The development of haptic software is a complex process that requires considering various engineering aspects of sensing and actuation additionally to the design of algorithms and mathematical models. In this chapter, with a special emphasis on realism, we discussed measurement-based and data-driven approaches that are optimized using data collected during real haptic interaction. The main goal was to deliver a fundamental knowledge of haptic modeling and rendering that can help a reader to formulate haptic models and implement realistic VR and MR simulators. To show the research landscape on haptic modeling and rendering, we provided a series of state-of-the-art methods, i.e., optimization-based FEM simulation, data-driven models with deterministic and stochastic response spaces, and the hybrid approach of the physics-based and data-driven models. The presented examples also provide an introduction to ongoing challenges in object deformation and haptic texture rendering.

References

1. Abdelrahman W et al (2011) A comparative study of supervised learning techniques for data-driven haptic simulation. In: Systems, man, and cybernetics (SMC), 2011 IEEE international conference on. IEEE, pp 2842–2846
2. Abdulali A, Atadjanov IR, Jeon S (2020) Visually guided acquisition of contact dynamics and case study in data-driven haptic texture modeling. IEEE Trans Haptics 13(3):611–627
3. Abdulali A, Hassan W, Jeon S (2017) Sample selection of multi-trial data for data-driven haptic texture modeling. In: 2017 IEEE world haptics conference (WHC), pp 66–71. https://doi.org/10.1109/WHC.2017.7989878
4. Abdulali A, Jeon S (2016) Data-driven modeling of anisotropic haptic textures: data segmentation and interpolation. In: International Conference on human haptic sensing and touch enabled computer applications. Springer, pp 228–239
5. Abdulali A, Jeon S (2016) Data-driven rendering of anisotropic haptic textures. In: International AsiaHaptics conference. Springer, pp 401–407
6. Abdulali A, Jeon S (2021) Data-driven haptic modeling of plastic flow via inverse reinforcement learning. In: 2021 IEEE world haptics conference (WHC), pp 115–120. https://doi.org/10.1109/WHC49131.2021.9517181
7. Abdulali A et al (2018) Data-driven modeling and rendering of force responses from elastic tool deformation. Sensors 18(1):237
8. Abdulali A et al (2019) Measurement-based hyper-elastic material identification and real-time FEM simulation for haptic rendering. In: 25th ACM VRST. New York, USA: ACM
9. Abdulali A et al (2020) Realistic haptic rendering of hyper-elastic material via measurement-based FEM model identification and real-time simulation. Comput Graph 89:38–49
10. Arnaiz-Gonzàlez Á et al (2016) Fusion of instance selection methods in regression tasks. Inform Fusion 30:69–79
11. Bachmann ER (2000) Inertial and magnetic tracking of limb segment orientation for inserting humans into synthetic environments. Tech. rep, Naval Postgraduate School Monterey CA
12. Barbič J, James DL (2008) Six-dof haptic rendering of contact between geometrically complex reduced deformable models. IEEE Trans Haptics 1(1):39–52
13. Bargteil AW et al (2007) A finite element method for animating large viscoplastic flow. ACM Trans Graph (TOG) 26(3):16-es
14. Bickel B et al (2009) Capture and modeling of non-linear heterogeneous soft tissue. ACM Trans Graph (TOG) 28(3):89. ACM
15. Chen D et al (2015) Data-driven finite elements for geometry and material design. ACM Trans Graph (TOG) 34(4):74
16. Choi MG (2014) Real-time simulation of ductile fracture with oriented particles. Comput Animation Virt Worlds 25(3–4):455–463
17. Choi S, Kuchenbecker KJ (2012) Vibrotactile display: Perception, technology, and applications. Proc IEEE 101(9):2093–2104

18. Cirio G et al (2010) Six degrees-of-freedom haptic interaction with fluids. IEEE Trans Vis Comput Graph 17(11):1714–1727
19. Clausen J (1999) Branch and bound algorithms-principles and examples. In: Department of computer science, University of Copenhagen, pp 1–30
20. Cranstoun SD et al (2002) Time-frequency spectral estimation of multichannel EEG using the auto-SLEX method. IEEE Trans Biomed Eng 49(9):988–996
21. Culbertson H, Delgado JJL, Kuchenbecker KJ (2014) One hundred data-driven haptic texture models and open-source methods for rendering on 3D objects. In: IEEE haptics symposium (HAPTICS). IEEE, pp 319–325
22. Culbertson H, KuchenbeckerKJ (2017) Ungrounded haptic augmented reality system for displaying roughness and friction. IEEE/ASME Trans Mechatron 22(4):1839–1849
23. Culbertson H, Unwin J, Kuchenbecker KJ (2014) Modeling and rendering realistic textures from unconstrained tool-surface interactions. IEEE Trans Haptics 7(3):381–393
24. Davis RA, Lee TCM, Rodriguez-Yam GA (2006) Structural break estimation for nonstationary time series models. J Amer Stat Assoc 101(473):223–239
25. Elliott LR et al (2010) Field-based validation of a tactile navigation device. IEEE Trans Haptics 3(2):78–87
26. Erkelens JS (1996) Autoregressive modelling for speech coding: estimation, interpolation and quantisation. Citeseer
27. Fong P (2009) Sensing, acquisition, and interactive playback of data-based models for elastic deformable objects. Int J Robot Res 28(5):630–655
28. Gast TF et al (2015) Optimization integrator for large time steps. IEEE Trans Vis Comput Graph 21(10):1103–1115
29. Guruswamy VL, Lang J, Lee W-S (2009) Modelling of haptic vibration textures with infiniteimpulse-response filters. In: IEEE international workshop on haptic audio visual environments and games. IEEE, pp 105–110
30. Guruswamy VL, Lang J, Lee W-S (2010) IIR filter models of haptic vibration textures. IEEE Trans Instrum Measur 60(1):93–103
31. Hart P (1968) The condensed nearest neighbor rule. IEEE Trans Inform Theory 14:515–516
32. Hayes MH (2009) Statistical digital signal processing and modeling. Wiley
33. Hogema JH et al (2009) A tactile seat for direction coding in car driving: field evaluation. IEEE Trans Haptics 2(4):181–188
34. Hover R, Harders M, Székely G(2008) Data-driven haptic rendering of visco-elastic effects. In: Haptic interfaces for virtual environment and teleoperator systems, 2008. haptics 2008. Symposium on. IEEE, pp 201–208
35. Hover R et al (2009) Data-driven haptic rendering-from viscous fluids to visco-elastic solids. IEEE Trans Haptics 2(1):15–27
36. Irving G, Teran J, Fedkiw R (2004) Invertible finite elements for robust simulation of large deformation. In: Proceedings of the 2004 ACM SIGGRAPH/Eurographics SCA, pp 131–140
37. Iske A (2004) Multiresolution methods in scattered data modelling, Vol 37. Springer Science & Business Media
38. Jayant C et al (2010) V-braille: haptic braille perception using a touch-screen and vibration on mobile phones. In: Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility, pp 295–296
39. Jeon S, Choi S (2009) Haptic augmented reality: taxonomy and an example of stiffness modulation. Presence: Teleoper Virt Environ 18(5):387–408
40. Jeon S, Choi S, Harders M (2011) Rendering virtual tumors in real tissue mock-ups using haptic augmented reality. IEEE Trans Haptics 5(1):77–84
41. Jeon S (2011) Extensions to haptic augmented reality: modulating friction and weight. In: IEEE World haptics conference. IEEE, pp 227–232
42. Karp RM (1972) Reducibility among combinatorial problems. In: Complexity of computer computations. Springer, pp 85–103
43. Keogh E et al (2004) Segmenting time series: a survey and novel approach. Data Mining Series Databases 57:1–22

44. Kuchenbecker KJ, Fiene J, Niemeyer G (2006) Improving contact realism through event-based haptic feedback. IEEE Trans Visu Comput Graph 12(2):219–230
45. Lloyd B, Székely G, Harders M (2007) Identification of spring parameters for deformable object simulation. IEEE Trans Vis Comput Graph 13(5):1081–1094
46. Mazzella F, Montgomery K, Latombe J-C (2002) The forcegrid: a buffer structure for haptic interaction with virtual elastic objects. In: Proceedings 2002 IEEE international conference on robotics and automation (Cat. No. 02CH37292), vol 1. IEEE, pp 939–946
47. Meyer DJ, Peshkin MA, Colgate JE (2016) Tactile Paintbrush: a procedural method for generating spatial haptic texture. In: IEEE Haptics symposium (HAPTICS). IEEE, pp 259–264
48. Overby M et al (2017) ADMM projective dynamics: fast simulation of hyperelastic models with dynamic constraints. IEEE Trans Vis Comput Graph 23(10)::2222–2234. ISSN: 1077-2626. https://doi.org/10.1109/TVCG.2017.2730875
49. Rakhmatov R (2018) Virtual reality bicycle with data-driven vibrotactile responses from road surface textures. In: IEEE games, entertainment, media conference (GEM). IEEE, pp 1–9
50. Romano JM, Kuchenbecker KJ (2011) Creating realistic virtual textures from contact acceleration data. IEEE Trans Haptics 5(2):109–119
51. Sadia B et al (2020) Data-driven vibrotactile rendering of digital buttons on touchscreens. Int J Human-Comput Stud 135:102363
52. Salisbury K, Conti F, Barbagli F (2004) Haptic rendering: introductory concepts. IEEE Comput Graph Appl 24(2):24–32
53. Schulman J et al (2017) Proximal policy optimization algorithms. In: arXiv preprint arXiv:1707.06347
54. Strese M, Boeck Y, Steinbach E (2017) Content-based surface material retrieval. In: IEEE world haptics conference (WHC). IEEE, pp 352–357
55. Uzawa H (1958) Iterative methods for concave programming. Stud Linear Nonlinear Program 6:154–165
56. Wada C, Shoji H, Ifukube T (1999) Development and evaluation of a tactile display for a tactile vocoder. Technol Disab 11(3):151–159
57. Wang B et al (2015) Deformation capture and modeling of soft objects. ACM Trans Graph (TOG) 34(4):94
58. Wilson DL (1972) Asymptotic properties of nearest neighbor rules using edited data. IEEE Trans Syst Man Cybern 3:408–421
59. Wright SJ, Nowak RD, Figueiredo MA (2009) Sparse reconstruction by separable approximation. IEEE Trans Signal Process 57(7):2479–2493
60. Yim S, Jeon S, Choi S (2016) Data-driven haptic modeling and rendering of viscoelastic and frictional responses of deformable objects. IEEE Trans Haptics 9(4)