Haptic Puppetry for Interactive Games

Sujeong Kim, Xinyu Zhang, and Young J. Kim*

Department of Computer Science and Engineering,
Ewha Womans University, Seoul, Korea
kimsujeong@ewha.ac.kr, {zhangxy, kimy}@ewha.ac.kr

Abstract. In interactive computer games and computer animation, intuitively controlling the motion of an articulated character is considered as a difficult task. One of the reasons is that, typically, an articulated model used in the field has a high degree-of-freedom (DOF) for joints so that it is challenging to devise an easy-to-use interface to control the individual DOF. In this paper, as an alternative to existing techniques for controlling articulated characters, we propose the traditional marionette control [1] as natural interfaces to control the characters, and explain how to implement a virtual marionette based on physically-based modelling and haptic paradigm. Using our virtual marionette system, we can rapidly but easily create sophisticated motions for a high-DOF articulated character. Moreover, our system relies on haptic interfaces to model the behavior of real-world marionette controls and provides to the puppeteer responsive forces as a result of the created motions. This results in the puppeteer having a better sense of control over the marionette that she or he manipulates. Our experimentations show that our system can create reasonably complicated motions for articulated characters in an easy and quick manner at highly interactive rates.

1 Introduction

In computer animation and interactive computer games, controlling the motion of an articulated character intuitively is considered as a difficult task. One of the reasons is that, typically, an articulated model used in the fields has a high degree-of-freedom (DOF) for joints so that it is challenging to devise an easy-to-use interface to control an individual DOF. For example, the human model used in typical gaming environments has more than 30 DOFs [2] and controlling each DOF in the model intuitively is very difficult. In order to address these issues, techniques based on motion capturing or manual motion composition have been proposed [3]. However, these techniques require huge motion database or tedious manual work to create sophisticated motions. Moreover, these methods are often computed as off-line process so that creating an interactive response of characters at run-time is very difficult.

As a completely new alternative, we propose the traditional marionette control as natural interfaces for articulated character control in computer games and computer animation. In our system, instead of controlling an individual joint parameter in an articulated character, we use a virtual marionette to create sophisticated motions very quickly. The virtual marionette is simulated using physically-based modelling paradigm. For the most of real-world marionettes, a puppeteer manipulates the control - typically shaped

* Corresponding author. Tel.:+82-2-3277-4068, Fax:+82-2-3277-2306.

as cross or bar - to create a swinging motion of strings, which in turn moves the marionette itself. Our system relies on haptic interfaces to accurately model the behavior of real marionette controls and provides to a puppeteer accurate responsive forces as a result of created marionette motions. This results in the puppeteer having a better sense of control over the marionette that she or he manipulates. Typically, two controllers are used for real-world marionette controls: one for primary control and the other one for secondary, delicate control. In our system, we use two commodity haptic interfaces such as Sensable's Omni^{TM} to model each of them.

Our virtual marionette system based on haptic interfaces provides the following benefits over other existing animation system for articulated characters:

- Using our system, we can quickly create complicated motions for an articulated character in games.
- Haptic interfaces in our system enable users to intuitively control the articulated character and make them interact with virtual environments in real-time.
- Our system is based on physically-based modelling paradigm so that the generated motions and their responses are similar to those experienced in the real world.
- Our puppetry can be used as a stand-alone game like [4].

2 Related Work

2.1 Interfaces for Controlling Animation

Recently, many interfaces have been introduced to control animation for virtual characters [5, 6]. Typically, 2D interfaces like mouse [7, 4], a pen [8, 9] or a combination of mouse and keyboards are considered as most popular and approachable choices for an animation control. Along this line of approaches, an intuitive mapping of limited device-dependent actions, for example mouse clicks or dragging, to animation controls has been major research issues [10]. However, for high DOF characters like human, it is very challenging to create intuitive interfaces to map device actions in 2D to complicated character motions in 3D. In order to alleviate the issues of mapping 2D actions to 3D motions, different types of interfaces have been considered. A particular strength of 3D interfaces is that, unlike 2D interfaces, one does not need to rely on complicated mapping or animation sketching scheme to interpret device outputs in terms of character animation [11, 12]. However, it is still challenging to create complicated character motions using these devices, and some researchers have developed specialized interfaces for particular animation characters [5]. However, this approach lacks a generality of application to other types of characters.

2.2 Haptic Interfaces

Thanks to the recent advancement in hardware and software of haptic technology, haptic rendering techniques have been improved from a simple, point-based method [13, 14], generating only translational forces, to an object-based 6DOF haptic rendering method [15], creating both translational and rotational (i.e., torque) forces. Unlike other 3D interfaces, haptic interfaces are a promising tool for creating complicated character motions in that users can get an immediate response (i.e., force feedback) from what they control. However, most of work in haptically-inspired character control has been centered on controlling an individual joint in an articulated, animating character [11, 12], instead of creating the character's full body motion.

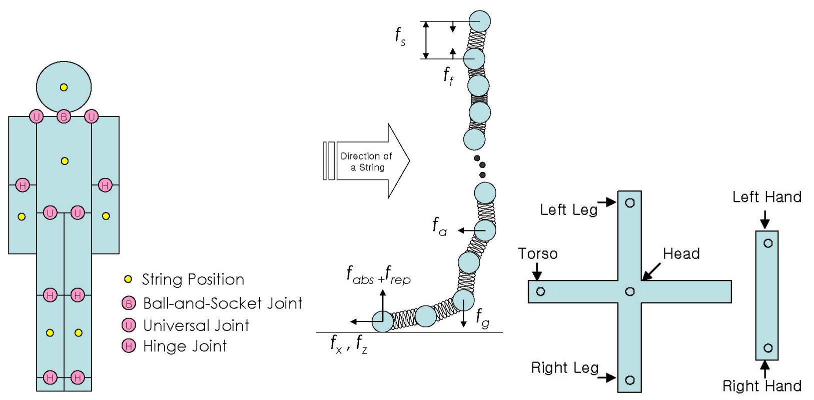
2.3 Physically-Based Animation

Compared to other animation techniques like traditional, cell-based animation [16] and motion capture-based animation [3], the physically-based animation technique has a particular strength in creating a realistic animation responsive to the surrounding environments. At a broad level, physically-based animation techniques can be classified into rigid body and deformable body simulation [17]. In our work, we are interested in rigid body simulation techniques for simulating the marionette itself [18, 19] as well as deformable body simulation techniques for simulating strings attached to the marionette [17].

3 Virtual Marionette

A real-world marionette is mainly composed of three parts: a control, strings, and a marionette itself [1]. The control is a tool with which one can actually manipulate the marionette. It is normally constructed by combining several bars, where strings are attached to each bar. The strings link the control to the marionette body, and deliver the manipulated result of the control to the body while they also deliver the forces generated by the marionette movement back to the control - actually to the human hand holding the control. Strings pull each part of the body, making a variety of body motions.

Since the goal of our system is to mimic controlling a real marionette, we model the essential components of a real marionette in a physically-correct manner. Thus, the



(a) Marionette (b) String (c) Control

Fig. 1. Modeling. A string is modelled as a chain of particles undergoing different forces like spring force f_s, inner friction f_f, gravity f_g, air friction f_a, ground friction f_x, f_z, ground absorption f_{abs} and ground repulsion f_{rep}.

interactions among these components are simulated entirely based on physically-based animation techniques. There are many types of marionettes, but we only consider a human-like marionette in our work.

3.1 Marionette Modeling

The marionette is made up of twelve rigid bodies (one sphere-like body and eleven box-like bodies) and eleven joints that connect the bodies together. Each body represents, respectively, the head, torso, two upper arms, two lower arms, two upper legs, two lower legs and two feet of a marionette.

The eleven joints are, respectively, the neck, shoulder, elbows, hips, knees, and ankles and they play an important role in generating a proper body motion. These joints are dotted as pink circles in Fig. 1.

We use three types of joints to connect the bodies as shown in Fig. 2 (terms borrowed from [20]): ball-and-socket, hinge and universal joints. The ball-and-socket joint simply makes two bodies always move together. The hinge joint makes two bodies only rotate around a certain axis. The universal joint acts like two hinge joints are combined at an anchor position. The universal joint provides relatively limited movement compared to other joint types, but it is more flexible than the hinge joint. However, restricting motion is not always a desirable way.

The constraint dynamics equation for each joint is expressed as follows [20]:

$$\begin{aligned} J_1 v_1 + \Omega_1 \omega_1 + J_2 v_2 + \Omega_2 \omega_2 &= c + C\lambda \\ \lambda &\geq l \\ \lambda &\leq h \end{aligned} \tag{1}$$

J_i and Ω_i are the Jacobian matrices. Different joint types have different constraints (i.e., Jacobian matrices). The linear and angular velocity vectors for the first body participated in the joint are v_1 and ω_1. Similarly v_2 and ω_2 correspond to the second body. c is a joint-dependent constraint vector. λ is a constraint force that is applied to the bodies to ensure that Eq. 3.1 is satisfied. λ has a lower (l) and upper (h) bound. C is a diagonal matrix, called the constraint force mixing (CFM) matrix. It allows the constraint force λ to be part of the constraint equation. C can be manipulated to get certain interesting effects [20].

The yellow circles in Fig. 1 are the locations in which the strings are attached to the marionette bodies: center of the head, torso, lower arms, and lower legs. Only six

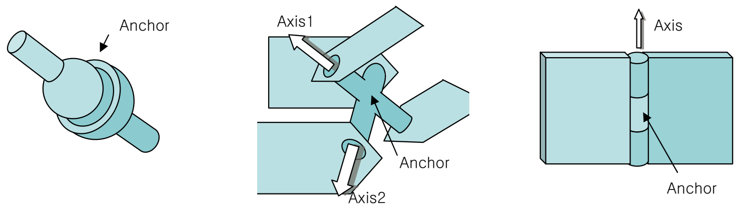


Fig. 2. Different Joint Types: ball and socket joint, universal Joint, hinge joint (from left to right)

parts of the body can be moved by strings, but from our experience these are enough to generate the whole body motion. The body can also return responsive forces back to the strings as a result of body-string interaction.

The forces of a string to pull each part of the body is calculated as follows:

$$f_{body} = -k(x_{body} - x_{string}) - k_d(\dot{x}_{body} - \dot{x}_{string}), \tag{2}$$

where x_{string} is the position of the last mass particle in the string, x_{body} is the position of the part of body that the string is attached to, k is a stiffness constant, and k_d is a damping constant. The force that is returned to the string is simply $f = -f_{body}$.

3.2 String Modeling

The strings provides a mean to deliver user's intention to the marionette. The user manipulates strings by using the control, and the strings pull each part of the marionette body so that the body finally creates some pose.

We model the string as a deformable body. There exist many methods to model a string-type deformable body, but we value the speed rather than the accuracy of simulation. In this regard, we select the spring-mass method, mainly because it is fast and, at the same time, it can provide a reasonable accuracy of the system. This method models a string as a set of mass particles inter-linked by a spring. The shape and behavior of a string are approximated by the particles' motion. However, we do not consider twisted motion of a string since real-world marionette strings are rarely twisted.

In our system, six strings hold the body: head string, hip string, two hand strings, two leg strings. The position of the first particle in each string is updated at every simulation time step and follows the position of the control. The motions of the rest of particles are governed by particle dynamics based on the following forces:

1. Spring force between two adjacent particles: $f_s = -k_s(x_i - d)$ where k_s is a stiffness constant of a spring, x_i is the distance between two particles, and d is the string length that we want to maintain.
2. Inner friction force: $f_f = -k_f \times (\dot{x}_i - \dot{x}_{i+1})$ where k_f is a friction constant
3. Gravitational force: $f_g = mg$ where m is the mass of a particle and g is the gravitational acceleration.
4. Air friction: $f_a = -k_{air}\dot{x}_i$ where k_{air} is an air friction constant
5. Ground friction (when particles fall on to the ground): The ground friction force is applied only to the x and z direction; $f_x = -v_i^x k_{ground}$ and $f_z = -v_i^z k_{ground}$ where v_i^x, v_i^z is the velocity of a particle i along x and z directions and k_{ground} is a ground friction constant.
6. Ground repulsion (when particles continue to remain on the ground): Apply ground absorption force $f_{abs} = v_i^y k_{abs}$ and repulsion force $f_{rep} = v_i^y(h_{ground} - p_i^y)$ where k_{abs} is a ground absorption constant, p_i^y is the y position of a particle i and h_{ground} is the ground height

After accumulating all the aforementioned forces (F), we numerically solve the Newtonian second order ordinary differential equation (ODE) for a particle system (i.e., $\ddot{\mathbf{x}} = \frac{\mathbf{F}}{m}$) using the implicit Euler's method [18].

We perform collision detection and collision response between strings and the ground. If the position of a particle falls below the ground, we apply the ground repulsion and absorption force to the particles (steps 5 and 6). However, we do not consider collision detection and response between a string and a string and between strings and bodies. The reasons are as follows:

- Our system requires to be highly interactive so that we can not afford such costly collision detection and response time.
- The goal of our system is a character motion control, not string simulation.
- In real-world marionette control, there is no technique using twisted strings requiring collision detection between them.

3.3 Modeling of the Control

The control itself is not a target object to be physically simulated in our system. It is just a tool or an interface that takes the user's inputs (position and orientation) and delivers them to the system. In other words, it just drives other physical objects (e.g., strings and marionette body) in our system to move accordingly. Each part in the marionette body is manipulated by a string, and the string is operated by the control. Users can generate various marionette motions by moving the control and give it different positions and orientations. The control is directly manipulated by the user through haptic interface. In our system, we have two controls, a main control and a hand bar. The main control is for controlling the main body including the head, torso, legs; the hand bar is for the hands. The geometric structure of the control in our system is very similar to the real marionette controls.

The real-world control can be classified into two types, vertical and horizontal controls. Ours resembles the latter. The strings are attached to a marionette as shown in Fig. 1 (the white circles).

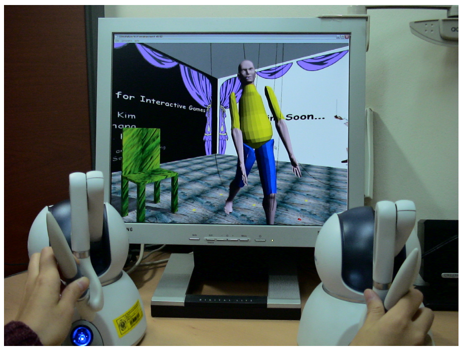
4 Haptic Interfaces

A notable aspect in our puppetry system is that we use haptic interfaces to manipulate the virtual marionette. In fact, the haptic interfaces are directly mapped to control bars as shown in Fig. 3. As a result, we can provide a more intuitive, easier way to manipulate the marionette, instead of using complicated key combinations.

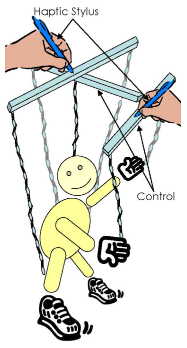
4.1 Interface Design

We use two stylus-type, commodity haptic devices, Omni developed by Sensable, to model the main control and hand bar in our system, as shown in Fig. 3.

Each device has six degree-of-freedom (DOF) inputs (position and orientation of haptic stylus) and three DOF outputs (translational forces). Its position and orientation are updated at haptic update rates (i.e., $1KHz$). The tips of the haptic stylus in the two haptic interfaces are mapped to the center of mass of the main control and the hand bar, respectively as shown in Fig. 3. Furthermore, since each haptic device has its



(a) System Setup



(b) Control

Fig. 3. System Setup and Control Mapping. (a) shows the system setup and (b) shows of mapping the control to the haptic stylus.

own device coordinate system, we need to get its proper position in world coordinate system.

4.2 Haptic Force Computation

In order to take into account tension forces from strings, we distinguish the state of a string into *tight* and *loose*. We simply calculate the Euclidean distance between the first and last particles comprising the string, and if it is greater than a certain threshold, we call the state of a string tight; otherwise call it loose.

When a string i is tight, its haptic feedback F_i is delivered to the haptic device based on the following equation:

$$F_i = -k(x_{proxy} - x_{string}) - mg - k_d(\dot{x}_{proxy} - \dot{x}_{string}) \tag{3}$$

where k is a stiffness constant, m is the total mass of a marionette, k_d is a device-dependent damping factor, g is a gravitational acceleration, x_{string} is the position of the last particle in the string, and x_{proxy} is the position of haptic device. If the string is loose, no force is calculated. The force is accumulated from each string ($F = \sum F_i$) and it is finally delivered to the haptic device.

In practice, however, when a string is almost tight, a slight perturbation in the underlying dynamic simulation can unstably change the state of a string from tight to loose and vice versa. This can introduce unstable force jump in haptic force computations. Worse yet, the unstable jump in force computation can also cause haptic stylus to vibrate, which makes strings and marionette also move unstably following the device. This sequence of instabilities in force computation can make the entire simulation very unstable.

As a remedy to this problem, we do not allow a string to switch its state very often when the distance between the first and last particles comprising the string does not change much.More specifically, when the string state once enters a tight state, the string can not easily get out of a loose state. In our case, we enforce a threshold for distance difference to allow for state changes.

5 Results and Analysis

5.1 Implementation Issues

We have implemented our puppeteering system using Visual C++ and OpenGL graphics library. We have chosen Sensable's Omni haptic devices and OpenHaptics library [21] as haptic APIs. To perform articulated body dynamics for a marionette at interactive rates, we bound each body part of a marionette with a simple bounding volume such as a box, a cylinder, and a sphere, and perform dynamics based on them. However, when we render the marionette, we display the actual geometry contained in the bounding volume. The Open Dynamics Engine (ODE) [20] has been adopted as a basic physics engine for articulated body dynamics and slightly modified to be better suited for our purpose. We use ODE's ball-and-socket, hinge and universal joints to model Marionette's joints.

In our system, we maintain four independent processes that need to be synchronized: string simulation, marionette body dynamics, haptic rendering, and graphical rendering as shown in Fig. 4. Different processes are synchronized in the following manner:

1. The haptic simulation loops around asynchronously at haptic update rates.
2. The string simulation reads the position of the tip in the haptic device, and deforms corresponding strings.
3. The result of deformable string simulation acts as external forces to the articulated body dynamics engine of the marionette.
4. The created motion of the marionette is graphically rendered.
5. The marionette motion applies forces to the last particles of strings, and the entire strings are deformed responsively.
6. The simulation results (i.e., forces) of strings and marionette motions are sent back to the haptic device.

Each process is updated at different rates; $1KHz$ for string simulation and haptic rendering, $30Hz$ for marionette body dynamics and graphical rendering. Typically we maintain 50 particles to simulate string dynamics and one can adaptively simulate the string dynamics using the technique like [22].

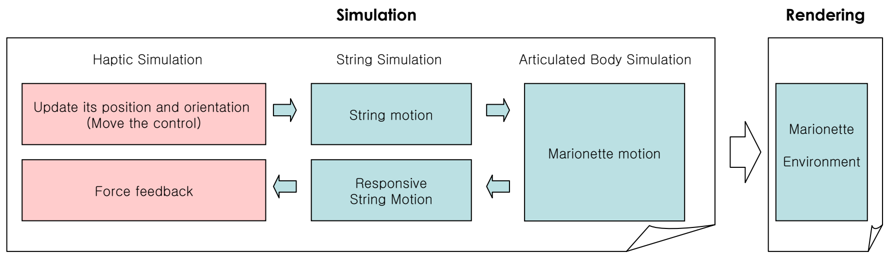


Fig. 4. System Diagram. The red block is an asynchronous process whereas the blue blocks are synchronized with each another.

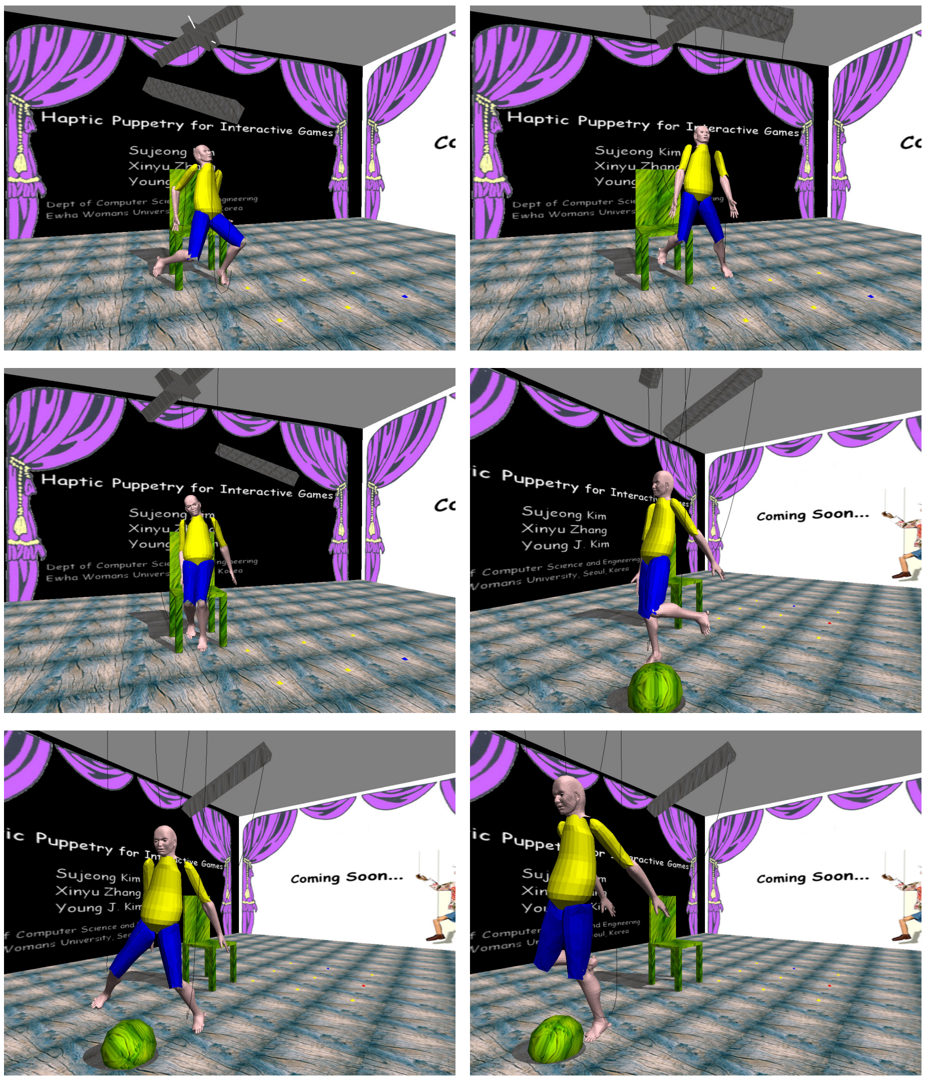

Haptic Puppetry for Interactive Games
Sujeong Kim
Xinyu Zhang
Young J Kim
Dept of Computer Science and Engineering
Ewha Womans University, Seoul, Korea
Coming Soon...

Fig. 5. Animation Sequences In Our Virtual Marionette System. The virtual marionette is standing up from a chair, and kicking and chasing a ball.

5.2 Results

With our system, an animator or a puppeteer can easily sketch motions for an articulated character as shown in Fig. 5. Our system is highly interactive (running at more than 1 KHz) and can create physically-plausible responses of a marionette to the environments. In Fig. 5, a virtual marionette is manipulated to stand up from a chair, kick a ball and chase it.

6 Conclusions and Future Work

We have presented an interactive system that simulates a marionette in a physically correct way. Moreover, our system provides an intuitive interface based on haptic devices to a puppeteer. The puppeteer can control a marionette using two haptic interfaces and perform a variety of interesting motions including interactions with environments that would be very difficult to be performed with a real-world puppet. As future work, we will like to apply our technique to other types of puppets (e.g., non-string type puppet). In order to improve the performance of our system, we want to apply an adaptive technique like [22] to both articulated body dynamics and string simulation. One limitation of our system is that it can not handle inter-string collisions and body-string collisions. However, this may be required for other types of more sophisticated puppets. We will like to incorporate such collision cases into our future puppeteering system. Finally, we want to apply our technique to higher DOF haptic interfaces to gain a more intuitive control over a marionette.

Acknowledgements

This work is sponsored in part by the grant R08- 2004-000-10406-0 of the KRF funded by the Korean government, the Ewha SMBA consortium, the ITRC program and the MOST STAR program.

References

1. Currell, D.: Making and Manipulating Marionettes. The Crowood Press (2004)
2. Boulic, R., Fua, P., Herda, L., Silaghi, M., Monzani, J., Nedel, L., Thalmann, D.: An anatomic human body for motion capture. In: EMMSEC. (1998)
3. Kovar, L., Gleicher, M., Pighin, F.: Motion graphs. In: ACM SIGGRAPH. (2002)
4. Healey, M.: (Ragdoll Kungfu) *http://www.ragdollkungfu.com/*.
5. A. Bar-Lev, A.M.B., Elber, G.: Virtual marionettes: A system and paradigm for real-time 3D animation. Technical report, Technion, I.I.T., Israel (2004)
6. Davis, J., Agrawal, M., Chuang, E., Popovi, Z., Salesin, D.: A sketching interface for articulated figure animation. In: SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation. (2003)
7. Vodislav, D.: A visual programming model for user interface animation. In: Visual Languages. (1997) 348–355
8. Laszlo, J., Panne, M., Fiume, E.: Interactive control for physically-based animation. In: Proceedings of SIGGRAPH 2000. (2000) 201–209
9. Oshita, M.: Pen-to-mime: A pen-based interface for interactive control of a human figure. In: Sketch-Based Interfaces and Modelling. (2004) 43–52
10. Thorne, M., Burke, D., Panne, M.: Motion doodles: An interface for sketching character motion. In: Proc. of ACM SIGGRAPH. (2004)
11. Oore, S., Terzopoulos, D., Hinton, G.: A Desktop Input Device and Interface for Interactive 3D Character Animation. In: Proc. Graphics Interface. (2002) 133–140
12. Jorissen, P., Wijinants, M., Lamotte, W.: Dynamic interactions in physically realistic collaborative virtual environments. IEEE Transactions on Visualization and Computer Graphics **11** (2005) 649–660

13. Basdogan, C., Srinivasan, M.: Haptic rendering in virtual environments. In: Virtual Environments HandBook. (2001)
14. Zilles, C., Salisbury, J.: A constraint based god-object method for haptic display. In: IEE/RSJ International Conference on Intelligent Robots and Systems, Human Robot Interaction, and Cooperative Robots. (1995)
15. Kim, Y.J., Otaduy, M.A., Lin, M.C., Manocha, D.: Six-degree-of-freedom haptic display using incremental and localized computations. Presence **12** (2003)
16. Lasseter, J.: Principles of traditional animation applied to 3D computer animation. In: Proc. of ACM SIGGRAPH. (1987)
17. Erleben, K., Sporring, J., Henriksen, K., Dohlmann, H.: Physics Based Animation. Charles River Media (2005)
18. Witkin, A., Baraff, D.: Physically based modeling: Principles and practice. In: SIGGRAPH Course Note. (1997)
19. Featherstone, R.: Robot Dynamics Algorithms. Kluwer (1987)
20. Smith, R.: Open Dynamics Engine user guide (2004)
21. SensAble: (3D Touch SDK OpenHaptics toolkit version 1.02 API reference) http://www.sensable.com.
22. Redon, S., Galoppo, N., Lin, M.C.: Adaptive dynamics of articulated bodies. In: Proceedings of SIGGRAPH 2005. (2005)