이화여자대학교 일반대학원
2006 학년도
석사학위 청구논문

View-Dependent Simulation of Articulated Bodies with Haptic Feedback

컴 퓨 터 정 보 통 신 공 학 과

김 수 정

2007

View-Dependent Simulation of Articulated Bodies with Haptic Feedback

이 論文을 碩士 學位 論文으로 提出함

2007年 7月

梨花女子大學校　一般大學院

컴퓨터정보통신공학과　김 수 정

김 수 정의 碩士 學位 論文을 認准함

指導教授	김 영 준	
審査委員	김 명	
	용 환 승	
	김 영 준	

梨花女子大學校　一般大學院

Contents

Abstract x

I. Introduction 1

1.1 Dissertation Goals 3
1.2 Main Contributions 4
1.2.1 View-Dependent Dynamics 5
1.2.2 Continuous Collision Detection of Articulated Bodies 5
1.2.3 Interactive Character Control 6
1.3 Organization 7

II. Related Work 8

2.1 Physically-Based Character Simulation 8
2.1.1 Physically-Based Animation 8
2.1.2 Articulated-Body Dynamics 9
2.1.3 Continuous Collision Detection 9
2.2 Simplification of a Dynamics Simulation 10
2.2.1 Simulation Levels of Detail 10
2.2.2 View-Dependent Simplification 10
2.2.3 Perceptually-Based Simplification 11
2.3 Interactive Character Control 11
2.3.1 Interfaces for Controlling Animation 11
2.3.2 Haptic Interfaces 12
2.4 Featherstone's Divide and Conquer Algorithm 12

III. View-Dependent Dynamics of Articulated Bodies 14

3.1 View-Dependent Simulation 14
3.1.1 Defining an Error Measure 15
3.1.2 Simplifying the Dynamics 17
3.2 Adaptive Articulated-Body Dynamics 18
3.2.1 Hybrid Bodies 18
3.2.2 Active Region Determination 19

3.3 View-Dependent Metrics 20
3.3.1 Simplifying the Simplification Problem 20
3.3.2 Semi-Predictive Metrics 21
3.3.3 Calculation of $\mathscr{N}_{\mathbf{M}}$ using Bounding-Volume Hierarchies 22
3.3.4 Calculation of $\mathscr{N}_{\mathbf{M}}$ using GPUs-Based Occlusion Queries 23
3.3.5 Comparisons Between the Two Methods for Calculating $\mathscr{N}_{\mathbf{M}}$ 24

IV. Continuous Collision Detection for Adaptive Simulation of Articulated Bodies 25
4.1 Preliminaries 25
4.1.1 Continuous Collision Detection 26
4.1.2 Adaptive Dynamics 27
4.2 Kinematics 28
4.2.1 Definitions 29
4.2.2 Recursive Transformation Updates 30
4.2.3 Bounding Transformations 30
4.3 Continuous Collision Detection 31
4.3.1 AABB Culling 32
4.3.2 Computing Contact Information 32

V. Haptic Puppetry for Interactive Games 34
5.1 Virtual Marionette 34
5.1.1 Marionette Modeling 35
5.1.2 String Modeling 37
5.1.3 Modeling of the Control 38
5.2 Haptic Interfaces 39
5.2.1 Interface Design 40
5.2.2 Haptic Force Computation 40

VI. Results and Discussion 42
6.1 Implementation Platform 42
6.2 Benchmarks and Applications 43
6.2.1 View-Dependent Dynamics of Articulated Bodies 43
6.2.2 Continuous Collision Detection for Adaptive Simulation of Articulated Bodies 46
6.2.3 Haptic Puppetry 48
6.3 Discussions 51

VII. Conclusion 52

References **54**

논문 개요 **61**

Figures

2.1 **An example of a chain-like articulated body and its assembly tree.** ***Left:*** *Two sub-articulated bodies A and B are attached by a joint to form a new articulated body C.* ***Right:*** *The corresponding assembly tree for the articulated body C. Here, the root node represents the articulated body C; the internal nodes represent the sub-articulated bodies A and B; the leaf nodes represent all the rigid bodies in C.* . . 13

3.1 **View-dependent dynamics simplification of a toy-like dog model in an interactive application (offline rendering).** **a***: the complete environment. The user controls the model (sixteen rigid bodies) with a haptic interface.* **b***: the user places the dog behind the environment. Our algorithm automatically rigidifies the legs of the model, resulting in a total of eight rigid groups.* **c:** *this back view shows the rigid groups corresponding to the position shown in* **b** *(one color per rigid group).* 15

3.2 **Assembly tree of an articulated body.** *Adaptive articulated-body dynamics simulates only some of the joints in the articulated bodies, which form the* active region. . 18

3.3 **View-dependent motion metric.** *The view-dependent motion metric is obtained by combining a priori motion metrics with visibility estimations.* 19

3.4 **Visibility estimation from bounding volumes.** *The visibility estimation* $\widetilde{N}_{\mathbf{M}}(C)$ *is taken as the minimum of two projections.* **a***: Projected bounding volume of the entire articulated body.* **b***: Sum of the projected bounding volumes of individual rigid bodies.* 23

4.1 **The two-stage pipeline of our continuous collision-detection algorithm (adapted from [60]).** *The algorithm computes the time of collision and the contact location from two successive configurations of an articulated model.* 26

4.2 **The kinematic representation used in our continuous collision-detection algorithm.** *The figure shows the principal joint transformation* $\mathbf{T}_A^B$*, the principal-to-secondary handle transformations* $\mathbf{T}_1^A$, $\mathbf{T}_2^A$, $\mathbf{T}_1^B$, $\mathbf{T}_2^B$ *and* $\mathbf{T}_3^B$*, and the child-to-parent transformations* $\mathbf{T}_A^C$ *and* $\mathbf{T}_B^C$. 28

4.3 **Adaptive computation of rigid-body transformations using an assembly tree.** *Only the nodes above the dotted curve are being simulated at the current time step, which allows us to limit the transformation updates to a limited number of nodes: (***a***) updating the principal joint, the principal-to-secondary and child-to-parent transformations of the simulated nodes (green nodes); and (***b***) updating the world transformations for the potentially colliding rigid bodies (yellow nodes).* 32

5.1 **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}.* 35

5.2 **Different Joint Types.** *ball and socket joint, universal Joint, hinge joint (from left to right)* 36

5.3 **System Setup and Control Mapping.** *(a) shows the system setup and (b) shows of mapping the control to the haptic stylus.* 39

5.4 **System Diagram.** *The red block is an asynchronous process whereas the blue blocks are synchronized with each another.* 40

6.1 **View-dependent dynamics simplification of a falling character.** *Our algorithm automatically simplifies the dynamics of the falling character while preserving the overall visual aspect of the impact (*e.g.* legs motion, see also Fig. 6.2.2 for a close-up on the final frames.).* 44

6.2 **View-dependent dynamics simplification. Top***: Our algorithm automatically simplifies the dynamics of a falling character as its distance to the viewer increases.* **Bottom***: Corresponding rigidification at this time step (one color per rigid group). See also Fig. 6.1.1 for the corresponding motion strips.* 44

6.3 **View-dependent Simulation of 100 Swinging Toy Dogs.** *100 toy-like dogs consisting of 1600 rigid bodies are attached to virtual springs (not shown in the image) in space and simulate dynamics in a view-dependent manner. In this scene, as many dogs are occluded by other dogs, clipped against the viewport, or seen far from the viewer, the number of simulated, active joints is reduced from 1500 to 493 on average without incurring visual deterioration in the simulation. Each simulation frame takes 14 msec on average.* 45

6.4 **Performance of our algorithm depending on visual error thresholds.** *As the visual error threshold e increases, the number of active nodes is automatically decreased by the view-dependent algorithm (***left***), and the computational cost is reduced (***right***).* 45

6.5 **Performance of our algorithm depending on the viewer's distance.** *Our algorithm automatically decreases the number of active joints when the distance d between the viewer and the pendulum increases (***left***), reducing the computational cost of the simulation (***right***).* 45

6.6 **Benchmarking examples.** *In these figures, the number of active joints is 15 and rigid bodies with identical colors belong to the same group of inactive links. Also, for each row of the figures, the third column is a zoomed-up version of the second column to show collision area.* **Top row:** *two wooden mannequin models consisting of 29 rigid bodies, 11K triangles for each model are pulled together by a spring-like string and collided with each other.* **Middle row:** *a pendulum consisting of 30 rigid bodies and 119K triangles swings because of gravity and collides with itself.* **Bottom row:** *a wooden mannequin falls under gravity and collides with static tableware. The static environment consists of 153K triangles in total.* 47

6.7 **CCD timings for three benchmarks and a comparison between them.** *Each graph shows a timing profile of CCD performance with AABB culling and exact contact computation steps for three benchmarking scenarios, and their comparison. The CCD time complexity gradually increases as a function of the number of active joints.* 48

6.8 **Animation sequences in our virtual marionette system.** *The virtual marionette is standing up from a chair, and kicking and chasing a ball.* 50

Tables

6.1 **Model complexities of benchmarking models.** *The third and fourth column show the triangle counts and number links of the benchmarking models, respectively.* . . . 46

Abstract

Recently, articulated-body simulation has been widely used in computer graphics as an efficient way of modelling and animating virtual characters. In particular, physically-based articulated-body simulation is able to realistically and naturally model the motion of articulated bodies as well as the interactions between different bodies, or between a body and the surrounding, virtual environment.

Some of the major challenges in articulated-body simulation include offering a fast simulation with a large number of characters and designing an effective and easy-to-use interface which allows users to interactively control the characters. This dissertation makes three contributions to address these challenges.

First of all, we propose a view-dependent dynamics simulation, generating an approximated simulation by simplifying the articulated-body dynamics based on visual criteria. This method enables an automatic trade-off between visual precision and computational efficiency. We also examine general problems related to the subject, and propose a semi-predictive simplification method based on motion with visibility estimations.

Secondly, we introduce a continuous collision detection (CCD) algorithm for adaptive articulated-body dynamics. We define a new hierarchical set of transforms that represent the kinematics of an articulated body and it is used in our algorithm to efficiently detect the first time of contact between moving bodies, whose motions are governed by the view-dependent dynamics.

Finally, we introduce a new user interface combining a traditional marionette control method with haptic interfaces to be able to generate a complex motion of an articulated body. It offers easy and intuitive manipulation with force feedback to interactively control articulated bodies in virtual environment.

We implement and evaluate each of these techniques with various benchmarking settings. The experimental results shows that these techniques provide fast and effective solutions for physically-based simulation of a large number of articulated-body and its interactive control.

Chapter I

Introduction

The goal of character animation is to generate a desired motion of characters in virtual environment. The simulation result, such as interactions between characters or between the characters and its surrounding environment, as well as the motion of the character, should be natural and realistic to the user (or viewer).

Physically-based simulation has been taking on an increasingly important role in numerous graphical applications where a realistic motion is desired, in such areas as computer animation, feature films, computer games, and virtual reality. In particular, articulated-body dynamics has been used to realistically simulate the motions of diverse forms of animating characters such as humans, hair, animals, plants, etc. Even though there has been extensive research in this field, but many interesting issues have yet to be addressed.

One of the fundamental problems in articulated-body dynamics is the forward dynamics problem, which computes the motion of an articulated-body when the given forces are exerted on the body. The use of forward dynamics is an effective way to control or simulate a large number of articulated bodies in computer graphics or robotics [4, 11, 22, 23, 24, 30, 44]. Linear-time, optimal solutions for forward dynamics are well-known (*e.g.* [23, 24]); however, current solutions prove to be expensive when simulating numerous or complex articulated bodies as is common in feature films or computer games.

Another challenge in articulated-body dynamics is the modelling of interactions between bodies through collision detection and response. Collision detection (CD) is the problem of testing for interference between geometric models moving in space. Many applications in such areas as computer graphics, robotics and geometric modeling require fast and reliable CD to simulate the interaction between virtual objects. As a result, CD has been extensively studied and many efficient algorithms are known. At a broad level, CD algorithms can be classified into two categories: discrete CD algorithms which check for interferences between static instances of moving objects and continuous CD (CCD) algorithms which must explicitly account for the object's motion, and report the first time of contact (TOC) if a collision occurs.

Recently, CCD has drawn much attention from different communities because of the need for correctly dealing with dynamic nature in applications. The major advantage of CCD is that it always maintains the non-penetration constraints for moving objects so that no collisions are ever missing between simulation time steps, allowing the accurate modeling of rigid-body dynamics [5, 59]. In haptic rendering, CCD can be used to compute the god-object of the haptic probe, which should not penetrate the objects that the user is touching [49]. In robot motion planning, CCD plays an important role in finding a continuous, collision-free path between two configurations of a moving robot [66, 63].

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 [10] and intuitive controlling each DOF in the model is very difficult. In order to address these issues, techniques based on motion capturing or manual motion composition have been proposed [37]. 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.

1.1 Dissertation Goals

The goal of this dissertation is performing a view-dependent simulation of articulated bodies which then can be controlled by haptic interfaces. To achieve this goal, we introduce three relevant solutions that can be combined together to produce a realistic simulation of articulated bodies.

The view-dependent dynamics algorithm provides a fast and natural simulation of articulated bodies. It improves simulation performance by simplifying the forward dynamics based on visual criteria, more specifically, the visibility. For example, when a body is occluded by an obstacle or by another body, or when the body is clipped against the viewing window, we simplify the simulation for those bodies which are not visible. Moreover, when the body is visible but its extent on the screen space is very small, we also simplify this motion. Toward this end, our view-dependent dynamics algorithm automatically chooses relevant joint motions from the expected size of motion on the screen, and determine the level of simulation.

Closely coupled with the view-dependent forward dynamics algorithm, we introduce a CCD algorithm. Our algorithm efficiently detects the first time of contact by taking advantage of the simplification in adaptive dynamics simulation, and it guarantees non-penetration between objects which is necessary to model proper interactions during the simulation.

While these two algorithms are related to the simulation of articulated bodies in virtual environment, we also propose an interface which allows users to interact with the virtual environment. Here, haptic devices are used as a connecting metaphor between the virtual environment and the user. It delivers user's manipulation to the virtual environment, and returns feedback to the user based on the results of the simulation.

In our approach, we combine a traditional marionette control method with haptic devices to generate the complex motion of an articulated body. The haptic device is capable of naturally controlling 3D characters, since it receives inputs in 3D space. It makes the interface simpler and more intuitive than the one with a mouse or a keyboard, and it also provides an intuitive manipulation method. Moreover, it enables more realistic and detailed control for the user with the force feedback delivered to the users.

These three solutions are integrated into a system of interactive, articulated-body simulation. The sequence of procedures in this system can proceed as follows:

1. The user manipulates haptic devices by moving or tilting the tip of haptic devices.
2. The information from the haptic device is mapped to a control stick in the simulation environment, which holds a stringed character. The forces generated by the movement of haptic device are computed and are applied to an articulated character.
3. The motion of the character is computed by our view-dependent dynamics algorithm. The simplification can be applied based on the metrics using visual criteria.
4. During the simulation, our CCD algorithm detects collision between the bodies or between the body and its environment. An appropriate collision response such as colliding contact or sliding contact is computed in this step to model proper interactions.
5. As a result of simulation, a desired motion is generated, and the force feedback, which maps the string tension of the control stick, is delivered to the user.
6. These steps are repeated.

The resulting system provides an efficient, articulated-body simulation, which also allows the users to interactively manipulate the bodies with high DOFs using haptic devices. The marionette control interface with haptic devices enables the users to quickly and easily generate various complex motions, which can be delivered via artistic or technical endeavours.

1.2 Main Contributions

We propose new methods for an effective view-dependent simulation of articulated bodies with haptic feedback. Specifically, we address the following problems.

1.2.1 View-Dependent Dynamics

We proposes a method for *view-dependent articulated-body dynamics*, which simplifies the forward dynamics simulation of articulated bodies based on visual criteria. We propose a *semi-predictive* approach, which relies on a combination of exact, a priori error computations and visibility estimations. Our method is able to simplify the dynamics of an articulated body not only based on *visibility* criteria (*i.e.* the visible portion of the articulated body on the screen), but also based on the relative importance that the articulated-body motion has to the viewer. We demonstrate our approach on several benchmarks and show how our view-dependent articulated-body dynamics method allows an animator (or a physics engine) to finely tune the visual quality of a simulation, and obtain potentially significant speed-ups during interactive or off-line simulations. As will be shown in the benchmarking results, without incurring visual deterioration (*e.g.* popping), the view-dependent dynamics gracefully simplifies the level of details in the simulation and thus provide visually-plausible simulation to the viewer.

1.2.2 Continuous Collision Detection of Articulated Bodies

To model the interactions during the simulation, we present a method of CCD for articulated bodies whose motion is governed by such an adaptive simulation. We demonstrate how a new hierarchical set of transforms can describe the kinematics of an articulated body, and how it can be selectively and recursively updated during an adaptive simulation. This new approach to CCD matches the reduced complexity of adaptive simulation, resulting in a potentially significant increase in the speed with which articulated-body dynamics can be modelled.

Our CCD algorithm has the following characteristics:

- It extends the framework of adaptive articulated body dynamics to handle contacts.
- It includes a novel representation of hierarchical transforms which enables continuous collision detection to be performed adaptively with a number of degrees of freedom specified by the user.

1.2.3 Interactive Character Control

As an efficient control interface for animation characters, we propose the traditional marionette control as a natural interface 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 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. The system translates input from marionette controls into marionette motions and the resulting responsive forces are fed back into the haptic device. 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 [29].

1.3 Organization

The organization of this dissertation is as follows: chapter 3 discusses the problem of view-dependent simulation and proposes semi-predictive methods to address them. Chapter 4 presents a CCD algorithm for adaptive, articulated-body dynamics simulation. Chapter 5 introduces a haptic puppetry system, and explains the overall procedure of physically based simulation combined with haptic interface. Finally, chapter 6 concludes the dissertation.

Chapter II

Related Work

This chapter provides a brief survey of previous work related to physically-based simulation of articulated bodies and its simplification, as well as interactive control of animation characters.

2.1 Physically-Based Character Simulation

2.1.1 Physically-Based Animation

Compared to other animation techniques like traditional, cell-based animation [38] and motion capture-based animation [37], 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 [20]. In our work, we are interested in rigid body simulation techniques for simulating the marionette itself [72, 22] as well as deformable body simulation techniques for simulating strings attached to the marionette [20].

More precisely, the rigid body simulator employed in our system is, in fact, an articulated body simulator where many rigid bodies form links or chains. In computer graphics and robotics, articulated body simulation is often implemented using constraints-based techniques [72] based on divide-and-conquer paradigm [22].

Deformable body simulation also has been widely studied in computer graphics, computer animation, computer games, medical imaging, etc. The techniques to realize deformable body simulation can be classified into spring-mass system, finite element methods, and free form deformation [20]. Each method has its own advantage, but in our work, we are specifically interested in spring-mass system, since our system requires high update rates for interactivity and our deformable body is a simple string that can be easily modelled as a chain of mass-particles, inter-linked by a spring.

2.1.2 Articulated-Body Dynamics

The use of forward dynamics to simulate the motion of articulated bodies has been extensively studied [25]. Some of the best-known linear-time methods rely on a recursive formulation of the motion equations [4, 11, 22, 30, 44], while others involve simplifying the motion equations including the use of specialized spatial notations [23, 24].

Several techniques have been designed to simplify and reduce the cost of dynamic simulations. Faure [21] proposed an iterative method to refine forward dynamics by correcting bilateral constraint errors. Chenney et al. [14] used a view-dependent dynamics simplification. More recently, Redon and Lin have introduced algorithms for adaptive simplification of forward quasi-statics [62] and dynamics [56] of articulated bodies.

2.1.3 Continuous Collision Detection

There are six different approaches to CCD for a single body: algebraic equation solving [12, 16, 33, 57], swept volumes [2], adaptive bisection [58, 66], the use of kinetic data structures (KDS) [3, 34, 36], approached based on Minkowski sums [8], and conservative advancement [73]. However, these approaches only deal with a single body, and very few [58, 66, 73] is a more recent development running at interactive rates. CCD for articulated models have been proposed with applications to simple, capsule-shaped avatar models in VR environments at interactive rates [61] and to more fully articulated models with somewhat slower performance [60].

2.2 Simplification of a Dynamics Simulation

2.2.1 Simulation Levels of Detail

A number of approaches exist for adaptive simulation of a number of complex systems, including deformable bodies [19, 27], cloth [41], fluid and smoke [42], hair [71, 9], or objects with a finite but potentially large number of degrees of freedom such as particle systems [47] or articulated bodies [54]. Often, these approaches resort to some type of *adaptivity* to refine the simulation where the current level of discretized simulation cannot appropriately emulate the full dynamics of the system, independently of visual criteria (but possibly with application to view-dependent simplification).

2.2.2 View-Dependent Simplification

Some authors have specifically addressed the problem of simplifying a simulation based on visual criteria. Carlson and Hodgins [13] use three levels of sophistication to animate creatures in a virtual environment partly based on the distance of the creature to the camera (with arbitrary thresholds, however). Perbet and Cani [53] animate prairies using three levels of detail based on the viewer's position (classified as near, medium and far). O'Brien *et al.* [47] simplifies the dynamics of particle systems by clustering particles into groups, partly based on visual criteria. Chenney and Forsyth [14] discuss a view-dependent culling of dynamic systems and introduce two important criteria that should be satisfied by a dynamics simplification method, namely *consistency* (object re-entering the view satisfy viewers' expectations) and *completeness* (objects that should re-enter the view do so). In particular, they classify the objects based on the viewers expectations, and focus computing resources on the objects for which the viewers have certain expectations. However, they do not discuss how to simplify the dynamics of visible objects. Chenney *et al.* [15] focuses on the consistency problem when the objects motions can be tightly bound. Their method only applies to continuously evolving systems with few degrees of freedom and no external influence, which forbids interactive simulations and collision handling. Beaudoin and Keyser [7] present a method to simplify plant motion, and propose rigorous methods to compute the errors caused by the approximations. The levels of details are pre-computed and a generalization to other objects and external forces does not seem straightforward.

2.2.3 Perceptually-Based Simplification

How an animation or a simulation is being perceived by humans has recently received research attention. For example, O'Sullivan and Dingliana [52] discuss how viewers perceive collisions in a dynamics simulation of rigid bodies. Harrison *et al.* [28] study how noticeable changes in the lengths of articulated-body links are, depending on viewers attention. O'Sullivan [51] discusses the effect of collisions on attention. One goal of this research is often to define *perceptual metrics*, which would help simplify an animation or a simulation based on perceptual criteria.

2.3 Interactive Character Control

2.3.1 Interfaces for Controlling Animation

Recently, many interfaces have been introduced to control animation for virtual characters [1, 18]. Typically, 2D interfaces like mouse [70, 29], a pen [39, 50] 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 [69]. 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 [48, 32].

However, it is still challenging to create complicated character motions using these devices, and some researchers have developed specialized interfaces for particular animation characters [1]. However, this approach lacks a generality of application to other types of characters.

2.3.2 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 [6, 74], generating only translational forces, to an object-based 6DOF haptic rendering method [35], 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 [48, 32], instead of creating the character's full body motion.

2.4 Featherstone's Divide and Conquer Algorithm

In our work, the dynamics of object is governed by the view-dependent dynamics algorithm or the adaptive dynamics algorithm. These can be seen as a generalizations of the divide-and-conquer algorithm (DCA) proposed by Featherstone [23, 24]. In DCA, an articulated body is recursively defined in terms of articulated bodies connected with joints. Furthermore, a handle is defined for an articulated body, specifying an interfacing location within the articulated body to which internal forces between rigid bodies as well as external forces from the world can be applied [23]. Finally, the structure of the articulated body is represented as a binary assembly tree (c.f. Figure 2.1). The assembly tree for an articulated body is defined recursively using the following rules:

1. The root node represents the whole articulated body.
2. Each leaf node represents the rigid body contained in the articulated body.

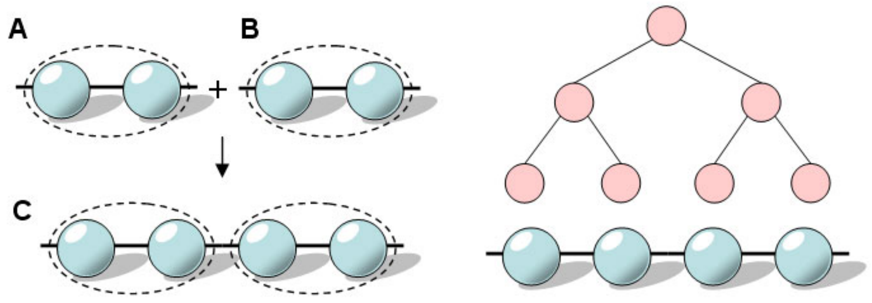


Fig. 2.1: **An example of a chain-like articulated body and its assembly tree.** ***Left:*** *Two sub-articulated bodies A and B are attached by a joint to form a new articulated body C.* ***Right:*** *The corresponding assembly tree for the articulated body C. Here, the root node represents the articulated body C; the internal nodes represent the sub-articulated bodies A and B; the leaf nodes represent all the rigid bodies in C.*

3. An internal node to represent a new articulated body is created by having the roots of two sub-assembly trees as children nodes that represent two sub-articulated bodies.

Featherstone [23, 24] shows that the dynamics of an articulated body can be described by the following *articulated-body equation*:

$$\mathbf{a} = \Phi\mathbf{f} + \mathbf{b}, \tag{2.1}$$

which is similar to the Newton-Euler equation describing the motion of a rigid body. Here, $\mathbf{a}$ is the composite acceleration of the articulated body (a vector which concatenates the bodies accelerations), Φ is the composite inverse inertia of the articulated body, $\mathbf{f}$ is a composite kinematic constraint force (which holds the articulated body together), and $\mathbf{b}$ is a composite bias acceleration, due to external forces and torques (inertial effects are zero under the quasi-statics assumption).

Featherstone's DCA essentially consists in two passes over the complete assembly tree. The *main pass* is a bottom-up traversal, in which the DCA determines the inverse inertias Φ and bias accelerations $\mathbf{b}$ for each node in the assembly tree from those of its children, and the external forces and torques applied on the articulated body. The top-down *back-substitution pass* computes, for each internal node starting from the root node, the kinematic constraint forces $\mathbf{f}$ (which enforce the kinematic constraint described by the node) and the acceleration $\ddot{\mathbf{q}}$ of the joint represented by the node. This algorithm is applied repeatedly to simulate the motion of an articulated body.

Chapter III

View-Dependent Dynamics of Articulated Bodies

We begin by introducing our view-dependent dynamics algorithm as a basic framework, which enables an automatic trade-off between visual precision and computational efficiency. We discuss the problem of simplifying the simulation based on visual criteria, and show that it raises a number of challenging questions. We then focus on articulated-body dynamics simulation, and propose a semi-predictive approach which relies on a combination of exact, a priori error metrics computations, and visibility estimations. We suggest several variants of semi-predictive metrics based on hierarchical data structures and the use of graphics hardware, and discuss their relative merits in terms of computational efficiency and precision.

3.1 View-Dependent Simulation

Although several methods have been proposed to take advantage of visibility to simplify simulations, it appears that very few authors have formally discussed *how* to choose an appropriate simulation level of detail based on visual criteria. In this section, we discuss this problem and iden-

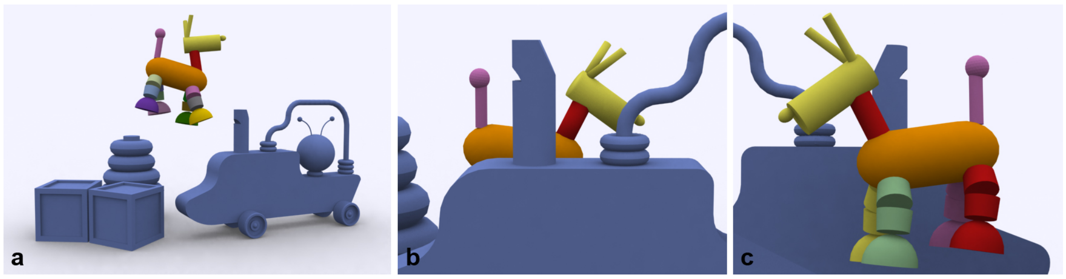


Fig. 3.1: **View-dependent dynamics simplification of a toy-like dog model in an interactive application (offline rendering).** **a***: the complete environment. The user controls the model (sixteen rigid bodies) with a haptic interface.* **b***: the user places the dog behind the environment. Our algorithm automatically rigidifies the legs of the model, resulting in a total of eight rigid groups.* **c:** *this back view shows the rigid groups corresponding to the position shown in* **b** *(one color per rigid group).*

tify a number of relevant issues. We find that the problem of quantifying and, most importantly, *predicting* the number of perturbed pixels due to an approximation in a simulation is surprisingly difficult, for a number of reasons. This may explain why the problem of simplifying a simulation based on its appearance has been relatively unexplored compared to, for example, the problem of view-dependent geometric simplification [31].

3.1.1 Defining an Error Measure

View-dependent simplification of dynamics first raises the problem of defining an appropriate error measure with which the quality of a simplification can be judged.

3.1.1.1 Static case

Consider first the *static* problem, *i.e.* simplifying a simulation at a single instant in time. In the static problem, we would like to simplify the dynamics of the objects in the scene at a given time step and still obtain, at the next time step, an image “close” to the one we would have obtained if the full dynamics had been simulated. We thus need an objective measure of the similarity between the simplified frame and the fully simulated one, at the next time step.

Fortunately, this question has already been raised within the graphics and scientific visualization research communities for geometric simplification (see [43] for an extensive overview). Often, the error between two images is measured in terms of the number of pixels that differ between the two images [31]. More generally, the characteristics of the visual system should be accounted for, in order to take advantage of *e.g.* attention [51], mesh saliency [40], etc. Although much progress has been made, the problem of defining a “perceptual distance” between two images is still largely open.

Note that the simulated objects may go through a complex, non-linear rendering stage, involving complex lighting and material models as well as various post-processing stages (*e.g.* motion blur, editing, etc.). Ideally, a completely integrated system would take the full simulation and rendering pipeline into account, to avoid spending too much time on motions that would later be hidden by post-process.

3.1.1.2 Dynamic case

The *dynamic* problem, *i.e.* considering the long-term impact of a simplification, raises additional questions. Indeed, we should probably consider the impact of a simplification at a given time step on *all* subsequent time steps. Ideally, all simplifications should be invisible to the viewer. One way to measure the error could thus be to compare the final images of the simplified segment and the original animation segment.

Because general systems are aperiodic, however, and may be extremely sensitive to perturbations, a simplified simulation might rapidly diverge from a non-simplified one. Thus, a better way to measure the error between two animation segments might thus be as the sum of (or bound on) successive static errors. This is typically how the quality of an integration method is evaluated, *i.e.* by computing a bound on the error occurring at each time step. Such an error measure might be able to solve the consistency and completeness problems defined by Chenney and Forsyth [14], by choosing sufficiently low error thresholds.

3.1.2 Simplifying the Dynamics

Adaptive methods can be roughly classified according to the way they evaluate (or estimate) and use error measures.

Ideally, the exact error should be computed *a priori*, *i.e.* *without* having to compute the exact solution to the problem. This is often extremely difficult, however, and a priori methods often *estimate* the error (*e.g.* [19]).

Note that another approach would be to design an *a posteriori* simplification method, similar to adaptive time-stepping integration methods. This would prove relatively easy: two images would be produced, one using full dynamics, and the other with view-dependent dynamics. If the images differ by less than a user-defined threshold (in terms of the number of pixels, for example), then the view-dependent simplification would be declared acceptable.

There are, however, at least three problems with such an approach. First, to make the scheme computationally worthwhile, we would have to assume that temporal coherency is high, so that the simplification test can be performed only once in a while (for example every one hundred frames). This might not be valid in numerous dynamics scenarios, especially when discontinuities resulting from collisions, external forces or user interactions may occur (moreover, regular "peaks" in computational costs might be inappropriate in interactive applications such as games and virtual environments, which favor consistent frame rates). Second, performing a full dynamics step might well be too slow when too many degrees of freedom are involved. Finally, it might be difficult to decide what simplification should be attempted. Should the degrees of freedom be removed because they have little variation? Should they be grouped together because they have *similar* variation? Because of the exponential number of combinations, some a priori assumptions have to be made to simplify the system before comparing it to the fully simulated one (*e.g.* merging and splitting degrees of freedom based on their acceleration [9]).

In this chapter, we propose a *semi-predictive* approach, which combines a priori computations of joint accelerations errors with visibility estimations.

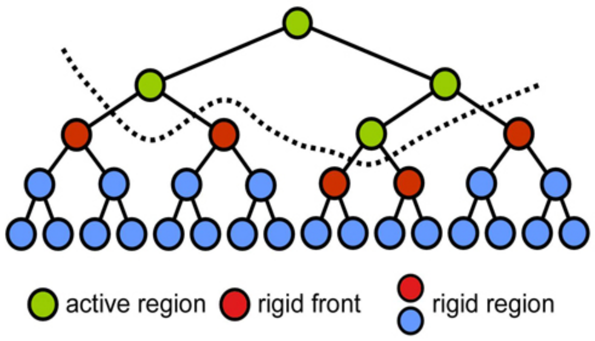


Fig. 3.2: **Assembly tree of an articulated body.** *Adaptive articulated-body dynamics simulates only some of the joints in the articulated bodies, which form the* active region.

3.2 Adaptive Articulated-Body Dynamics

The predictive component of our view-dependent method relies on the adaptive dynamics (AD) method introduced by Redon *et al.* [54]. For completeness, we briefly describe their approach here. We refer the reader to their paper for a detailed exposition.

3.2.1 Hybrid Bodies

The AD algorithm can be seen as a generalization of the Divide-and-Conquer Algorithm (DCA) proposed by Featherstone [23, 24]. In this algorithm, an articulated body is recursively defined by connecting two articulated bodies. A *binary assembly tree* describes the sequence of assembly operations, in which the leaf nodes represent rigid bodies, and the root node corresponds to the complete articulated body (see Figure 3.2). Each non-leaf node thus represents both a sub-articulated body and the joint used to connect its two child nodes.

In order to speed up articulated-body dynamics simulation, Redon *et al.* [54] *approximately* solve the problem by computing joint accelerations in a limited *sub-tree* of the assembly tree (*cf* Figure 3.2), called the *active region*. The remaining joints are *inactive*, and their accelerations are being implicitly set to zero. An articulated body with at least one inactive joint is called a *hybrid body*. The motion of a hybrid body is simulated by “rigidifying” the inactive joints. This results in

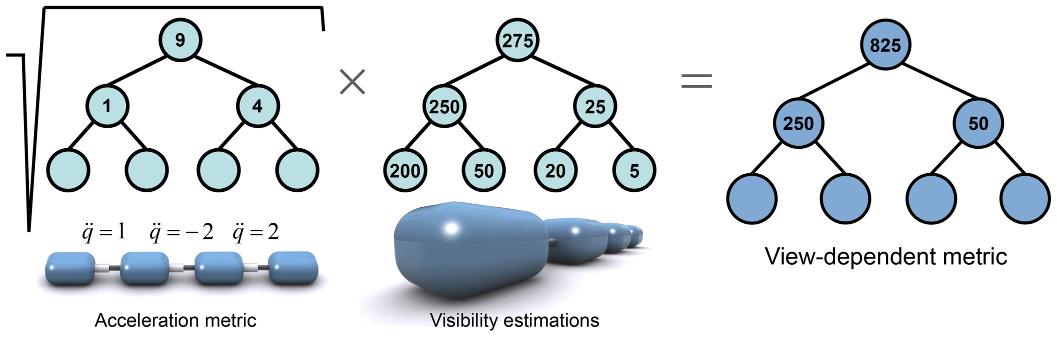


Fig. 3.3: **View-dependent motion metric.** *The view-dependent motion metric is obtained by combining a priori motion metrics with visibility estimations.*

hybrid inverse inertias and bias accelerations $\overline{\Phi}$ and $\overline{\mathbf{b}}$, that can also be computed from the bottom up. The complexity of simulating a hybrid body is then $O(n_a + n_f \log(n/n_f))$, where n_a is the number of active joints, n is the total number of joints, and n_f is the number of nodes where an external force of a torque is updated. This results in potentially significant performance speed-ups when the number of active joints and external forces updates are low.

3.2.2 Active Region Determination

To approximate the motion that would have been obtained if full dynamics were computed, Redon *et al.* [54] periodically updates the active region using *motion metrics*. If C is an articulated body, the *acceleration metric* $\mathscr{A}(C)$ of C is a weighted sum of its joint accelerations:

$$\mathscr{A}(C) = \sum \ddot{\mathbf{q}}_\mathbf{i}^T \mathbf{A}_i \ddot{\mathbf{q}}_\mathbf{i}, \tag{3.1}$$

where the $\mathbf{A}_i$ are symmetric, positive definite matrices. Similarly, the *velocity metric* $\mathscr{V}(C)$ of C is a weighted sum of its joint velocities:

$$\mathscr{V}(C) = \sum \dot{\mathbf{q}}_\mathbf{i}^T \mathbf{V}_i \dot{\mathbf{q}}_\mathbf{i}, \tag{3.2}$$

where the $\mathbf{V}_i$ are symmetric, positive definite matrices. Redon *et al.* [54] shows that the acceleration metric value of an articulated body is a quadratic function of the kinematic constraint forces:

$$\mathscr{A}(C) = (\mathbf{f}^\mathrm{C})^T \Psi^\mathrm{C} \mathbf{f}^\mathrm{C} + (\mathbf{f}^\mathrm{C})^\mathrm{T} \mathbf{p}^\mathrm{C} + \eta^\mathrm{C}, \tag{3.3}$$

where the *acceleration metric coefficients* Ψ^C, $\mathbf{p}^C$ and η^C can be computed from the bottom up (similar to the articulated-body coefficients). To update the active region, the acceleration metric is used to restrict the back-substitution pass to the most important sub-tree of the assembly tree. Then, the velocity metric is used to determine the new set of most important joints.

3.3 View-Dependent Metrics

We can now present our approach for view-dependent simplification of articulated-body dynamics. Our method can be regarded as *semi-predictive*, since we are able to predict the exact error in joint accelerations before actually computing all of them (using the adaptive dynamics framework), but we make some assumptions about how the visual error is affected by errors in joint accelerations.

3.3.1 Simplifying the Simplification Problem

In order to make the view-dependent simplification practical, we make two fundamental assumptions.

First, we assume that the motion of a joint only has a *local* effect on the motions of the rigid bodies (in cartesian coordinates). This is generally the case when there is little correlation in neighboring joints, and cross-coupling inverse inertias tend to have low ranks and not transmit applied torques and forces [64]. This allows us to examine each sub-assembly of the articulated body independently of the others, by assuming that the visual impact of a joint acceleration error in a sub-assembly is approximately restricted to this sub-assembly.

Second, we assume that the visual error caused by the rigidification of a sub-assembly can be roughly obtained by decoupling the contribution of the sub-assembly motion (acceleration or velocity) from the visibility of objects under such motion (*e.g.* the number of rasterized pixels). The major benefit of this assumption is that we can easily customize the motion metrics in the adaptive dynamics framework, and exploit well-established measures of visibility used in other types of applications, in particular in rendering.

These two assumptions allow us to formulate view-dependent motion metrics which can be computed efficiently, while still providing reasonable estimations of the visual error caused by partial rigidifications (*cf* Section 6.2.1).

3.3.2 Semi-Predictive Metrics

Let us call $\mathscr{N}_{\mathbf{M}}(C)$ the projected area of an articulated body C onto the screen, under the viewing transformation $\mathbf{M}$. Then, the view-dependent acceleration and velocity metrics, $\mathscr{A}_{\mathbf{M}}(C)$ and $\mathscr{V}_{\mathbf{M}}(C)$, are defined as follows (*cf* Figure 3.3):

$$\begin{aligned} \mathscr{A}_{\mathbf{M}}(C) = dt^2 \mathscr{N}_{\mathbf{M}}(C)\sqrt{\sum \ddot{\mathbf{q}}_{\mathbf{i}}^T \mathbf{A}_i \ddot{\mathbf{q}}_{\mathbf{i}}} = dt^2 \mathscr{N}_{\mathbf{M}}(C)\sqrt{\mathscr{A}(C)}, \\ \mathscr{V}_{\mathbf{M}}(C) = dt \mathscr{N}_{\mathbf{M}}(C)\sqrt{\sum \dot{\mathbf{q}}_{\mathbf{i}}^T \mathbf{V}_i \dot{\mathbf{q}}_{\mathbf{i}}} = dt \mathscr{N}_{\mathbf{M}}(C)\sqrt{\mathscr{V}(C)}, \\ \mathbf{A}_i = \mathbf{V}_i = \mathbf{D}_i, \end{aligned}$$

where dt is the size of the time step, and $\mathbf{D}_i$ is a diagonal weight used to homogenize joint types (to mix ball-socket joints and prismatic joints, for example — $\mathbf{D}_i$ can be set to the identity matrix if all degrees of freedom are of the same type).

Intuitively, these metrics estimate the visual error caused by zeroing accelerations or velocities by assuming the worst possible case: all joints in the sub-assembly have correlated motions, and contribute to the displacement of the whole visible surface. Once the acceleration and velocity metrics have been obtained as in the adaptive dynamics framework, these view-dependent metrics can be obtained in constant time, provided we know the values of $\mathscr{N}_{\mathbf{M}}(C)$. We now present different ways of obtaining $\mathscr{N}_{\mathbf{M}}(C)$ and discuss their relative advantages in terms of precision and computational efficiency.

3.3.3 Calculation of $\mathscr{N}_{\mathbf{M}}$ using Bounding-Volume Hierarchies

For a given sub-assembly C, we can quickly approximate $\mathscr{N}_{\mathbf{M}}(C)$ using bounding volumes (BVs) such as spheres, oriented bounding boxes [26] or axis-aligned bounding boxes. Before the simulation begins, we compute a bounding volume for each rigid body in the articulated body, that we store in the local reference frame attached to the rigid body. We also associate a bounding volume to each internal node of the assembly tree. These internal bounding volumes are updated at runtime, so as to enclose the bounding volumes of their children. In order to approximate $\mathscr{N}_{\mathbf{M}}(C)$, we can then use either the bounding volume associated to C, or the bounding volumes associated to the rigid bodies composing C. The resulting approximations of $\mathscr{N}_{\mathbf{M}}(C)$, denoted by $\widetilde{N}_{\mathbf{M}}(C)$, is taken as the minimum of these two projections (see also Figure 3.4):

$$\begin{aligned}
\widetilde{\mathscr{N}^1_{\mathbf{M}}}(C) &= \int\int_{\mathbf{M}(\partial \mathrm{BV}(C))} dxdy && (3.4)\\
\widetilde{\mathscr{N}^2_{\mathbf{M}}}(C) &= \sum_{i\in C}\int\int_{\mathbf{M}(\partial \mathrm{BV}(C_i))} dxdy && (3.5)\\
\widetilde{\mathscr{N}_{\mathbf{M}}}(C) &= \min\{\widetilde{\mathscr{N}^i_{\mathbf{M}}}(C) | i = 1,2\} && (3.6)
\end{aligned}$$

Even though we need to sacrifice additional time to update the internal bounding volumes at runtime, we may obtain a tighter estimation of $\mathscr{N}_{\mathbf{M}}$, depending on the configuration of the articulated body and the camera position. For example, if an articulated body forms a long but folded chain, then $\widetilde{\mathscr{N}^1_{\mathbf{M}}}$ will be smaller than $\widetilde{\mathscr{N}^2_{\mathbf{M}}}$. On the other hand, if this articulated body is stretched, then $\widetilde{\mathscr{N}^1_{\mathbf{M}}}$ will be larger.

The choice of the bounding volume also affects the theoretical complexity of the visibility estimation. If we use spheres or oriented bounding boxes, we can store their parameters in the local reference frame associated to the internal nodes. As a result, we do not have to update these bounding volumes in the rigid region, and the complexity of updating the bounding-volume hierarchy is linear in the number of *active* joints. In order to update the internal bounding volumes in constant time, however, we need to compute their parameters directly from those of their children, and not from the bounded geometry. This might result in overly conservative bounding volumes. In such a case, axis-aligned bounding volumes can be preferable despite the need to update all of them (*i.e.* including in the rigid region).

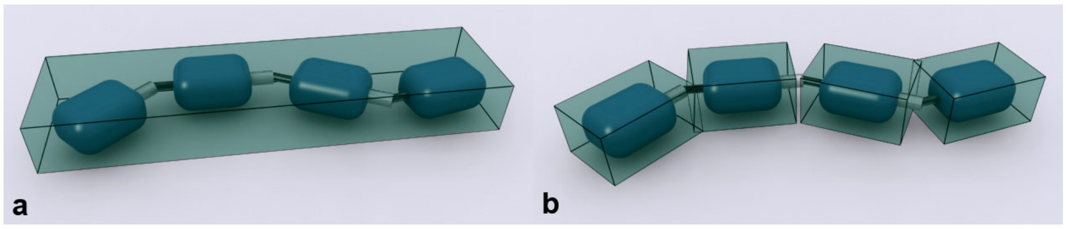


Fig. 3.4: **Visibility estimation from bounding volumes.** *The visibility estimation* $\widetilde{N}_{\mathbf{M}}(C)$ *is taken as the minimum of two projections.* **a***: Projected bounding volume of the entire articulated body.* **b***: Sum of the projected bounding volumes of individual rigid bodies.*

3.3.4 Calculation of $\mathscr{N}_{\mathbf{M}}$ using GPUs-Based Occlusion Queries

Estimating visibility using bounding volumes has the following disadvantages:

- Without a proper clipping procedure or visible surface determination, the $\widetilde{\mathscr{N}_{\mathbf{M}}}(C)$ value in Eq. 3.6 is always positive.
- Depending on the choice of bounding volumes, the projected area can be quite conservative.

In order to address these issue, we rely on occlusion queries supported by commodity graphics hardware [46]. Using occlusion queries, one can quickly obtain the number of visible pixel coverage of an articulated body C on the screen. However, this GPUs-based approach takes a linear time with respect to the number of links in C. More specifically, we get the number of visible pixels $\mathscr{N}_{\mathbf{M}}(C)$ for an articulated body C in two passes as follows:

1. Render the entire simulation scene including C and other objects in the environment.
2. Render each rigid body C_i (*i.e.* leaf-level node in the assembly tree) in C and use occlusion queries to determine the number of its visible pixels $\mathscr{N}_{\mathbf{M}}(C_i)$.
3. Recursively add up $\mathscr{N}_{\mathbf{M}}(C_i)$'s to get the number of visible pixels for the parent node of C_i's until we get $\mathscr{N}_{\mathbf{M}}(C)$.

By using a two-pass rendering method, we can get the number of visible pixels for C, occluded by the objects in the environment as well as by some of its own links in C (self-occlusion).

3.3.5 Comparisons Between the Two Methods for Calculating $\mathcal{N}_{\mathbf{M}}$

We have presented two methods to calculate $\mathcal{N}_{\mathbf{M}}$ earlier. Each method has its own cons and pros. The bounding-volume hierarchy-based method has a sublinear time complexity with respect to the number of links in an articulated body to calculate $\mathcal{N}_{\mathbf{M}}$; it is the same as that of the original adaptive dynamics [54]. However, in this case, it is not straightforward to take into account occlusion between links. Moreover, $\mathcal{N}_{\mathbf{M}}$ can be overly conservative depending on the relative configurations between links. On the other hand, the GPU-based method provides a tight estimation of $\mathcal{N}_{\mathbf{M}}$ and can easily handle different types of visibility including occlusion and screen space clipping. However, it requires a linear time complexity with respect to the number of links. But, in practice, the linear time complexity is almost negligible on modern graphics hardware thanks to its rapid rasterization capability.

Chapter IV

Continuous Collision Detection for Adaptive Simulation of Articulated Bodies

In this chapter, we introduce a CCD algorithm which can be applied to the adaptive dynamics simulation. Our algorithm efficiently detects the first time of contact by taking advantage of the simplification in adaptive dynamics simulation, and it guarantees non-penetration between the objects which is necessary to model proper interactions during the simulation. Our algorithm is based on a novel hierarchical set of transforms that represent the kinematics of an articulated body recursively, as described by an assembly tree. The performance of our CCD algorithm significantly improves as the number of active degrees of freedom in the simulation decreases.

4.1 Preliminaries

Our CCD algorithm is an adaptation of the algorithm proposed by Redon et al. [60], for an adaptive dynamics (AD) framework [56]. We will briefly describe these approaches to CCD and AD

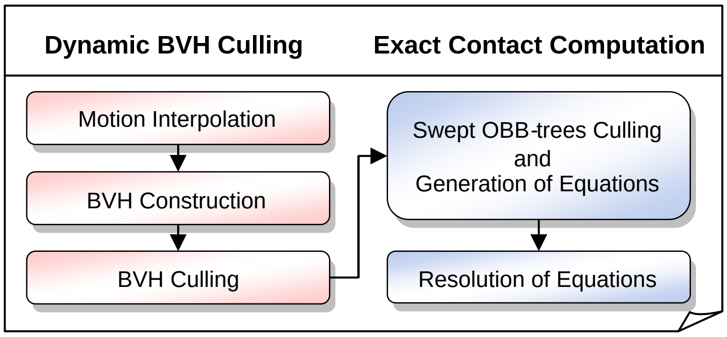


Fig. 4.1: **The two-stage pipeline of our continuous collision-detection algorithm (adapted from [60]).** *The algorithm computes the time of collision and the contact location from two successive configurations of an articulated model.*

prior to presenting our own contribution.

4.1.1 Continuous Collision Detection

In discrete simulations, CCD models the continuous motion between two successive time steps, and finds any collisions between the parts of each model, between models, or between the models and their environment. It reports the times of contact for any such collisions.

Let us consider an articulated model A composed of p rigid links $\mathsf{A}_1, ..., \mathsf{A}_p$, where $\mathbf{T}_i$ and $\mathbf{P}_i$ denote the position and orientation of the reference frame associated with link i. Then $\mathbf{M}_i^{i-1}(t)$ is the motion of $\mathbf{P}_i$ at time t in the reference frame of the parent link $\mathbf{P}_{i-1}$, and can be represented by the following 4×4 homogeneous matrix [60, 61]:

$$\mathbf{M}_i^{i-1}(t) = \begin{pmatrix} \mathbf{P}_i^{i-1}(t) & \mathbf{T}_i^{i-1}(t) \\ (0,0,0) & 1 \end{pmatrix}. \tag{4.1}$$

The motion of link i in the world reference frame can be computed by recursively multiplying the motions of its parents:

$$\mathbf{M}_i^0(t) = \mathbf{M}_1^0(t) \cdot \mathbf{M}_2^1(t) \cdots \mathbf{M}_i^{i-1}(t). \tag{4.2}$$

From this equation, we can represent the CCD problem as testing whether the following set is non-empty or not:

$$\{t \in [0,1] \, | \mathbf{M}_i^0(t)\mathsf{A}_i \cap \mathsf{O} \neq \emptyset, i = 1, ..., p\}. \tag{4.3}$$

Furthermore, if the above set turns out to be non-empty, we want to find the minimum value of t (TOC). To solve this problem, our CCD algorithm performs a two-stage process (Figure 4.1) that consists of a culling step, using a dynamic bounding-volume hierarchy (BVH), followed by an exact contact computation:

1. The motion between two successive time steps is computed using Equation (4.2).
2. Based on the continuous motions of each link of the articulated bodies, a hierarchy of axis-aligned bounding boxes (AABBs) is built for the entire model using interval arithmetic.
3. Based on this dynamic BVH, we cull those links that do not collide with other objects (these three steps correspond to the first step in Figure 4.1).
4. Finally, the exact contact is computed using a combination of interval arithmetic and subdivision. In this step, geometry is culled using trees of oriented bounding boxes (OBBs), followed by exact collision detection which yields the precise TOC (the second step in Figure 4.1).

4.1.2 Adaptive Dynamics

Adaptive dynamics (AD) is a forward dynamics method based on the divide-and-conquer algorithm (DCA) proposed by Featherstone [23, 24].

AD is an adaptation of Featherstone's DCA algorithm and is able to perform approximated forward dynamics based on a customizable motion metric. In AD, the assembly tree can be partially traversed during the two passes mentioned above. The nodes, joints and regions being traversed are called active nodes, active joints and active regions respectively, while the remaining nodes, joints and regions are said to be inactive. An articulated body with active and inactive regions is called a hybrid body.

An active region determination scheme with motion metrics [56] is used to update the set of active regions. The motion metrics consist of an acceleration metric

$$\mathscr{A}(C) = \sum \ddot{\mathbf{q}}_{\mathbf{i}}^T \mathbf{A}_i \ddot{\mathbf{q}}_{\mathbf{i}} \tag{4.4}$$

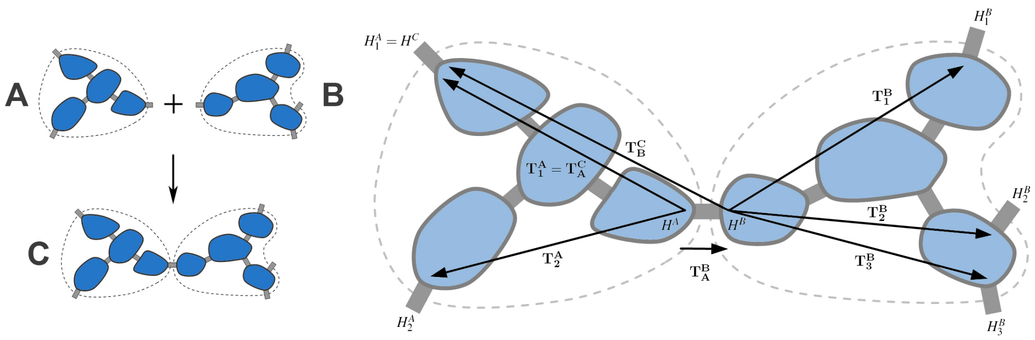


Fig. 4.2: **The kinematic representation used in our continuous collision-detection algorithm.** *The figure shows the principal joint transformation $\mathbf{T}_A^B$, the principal-to-secondary handle transformations $\mathbf{T}_1^A$, $\mathbf{T}_2^A$, $\mathbf{T}_1^B$, $\mathbf{T}_2^B$ and $\mathbf{T}_3^B$, and the child-to-parent transformations $\mathbf{T}_A^C$ and $\mathbf{T}_B^C$.*

and a velocity metric

$$\mathscr{V}(C) = \sum \dot{\mathbf{q}}_\mathbf{i}^T \mathbf{V}_i \dot{\mathbf{q}}_\mathbf{i} \tag{4.5}$$

where C is an articulated body and $\dot{\mathbf{q}}_\mathbf{i}, \ddot{\mathbf{q}}_\mathbf{i}$ are a joint velocity and acceleration, respectively. $\mathbf{A}_i$ and $\mathbf{V}_i$ are symmetric, positive definite matrices, which can be seen as weights on the joint accelerations or velocities. During simulation, the coefficients of the motion metrics are updated bottom-up, and the metric values are computed before the joint accelerations and velocities.

4.2 Kinematics

Based on the two methods above, our goal is to develop a new algorithm to exploit adaptive dynamics, which takes advantage of its simplified approach to simulation by adjusting its complexity as the active region changes.

We will now introduce a novel recursive representation of the kinematics of an acyclic branched mechanism and show how it can be used in AD. We start from the recursive definition of a branched articulated body proposed by Featherstone [23, 24], and introduce a set of transformations to describe the kinematics of the mechanism. Compared to the hierarchical representation we introduced earlier [62], this new representation is simpler and more efficient. In particular, updating the transforms that apply to a node in the assembly tree now has a linear complexity in the number of handles of the node, compared to a quadratic complexity in the hierarchical representation, due to the quadratic number of transformations per node.

4.2.1 Definitions

As in the DCA [23, 24], we define a (possibly branched but acyclic) articulated body C as two articulated bodies A and B connected by a joint J, and this sequence of assembly operations is described in a binary assembly tree. In this recursive description, the leaf nodes of this assembly tree are rigid bodies, while its internal nodes represent partial articulated bodies and the root node corresponds to the complete articulated body1). The internal nodes may also be taken to represent the joints used in the binary assembly operation.

In this representation, each sub-articulated body has a set of handles, i.e. locations where other sub-articulated bodies may be attached. For the sake of convenience, we will call the handle H^A used to connect A to another articulated body the *principal handle* of A, while the k other free handles H_i^A of A ($1 \leqslant i \leqslant k$) are called its *secondary handles*. Finally, if an articulated body C is formed by assembling A and B, we call the joint used to carry out the assembly the *principal joint* of C.

To describe the kinematics of the mechanism, we rigidly attach a reference frame to each handle H, and define the following sets of rigid transformations:

1. **Principal joint transformations**: the principal joint of each sub-assembly C with children A and B has an associated transformation $\mathbf{T}_A^B$, that relates (the reference frame of) the principal handle of A to (the reference frame of) the principal handle of B.

2. **Principal-to-secondary handle transformations**: each sub-assembly C (possibly a link) stores transformations $\mathbf{T}_i^C$ from its principal handle to its secondary handles H_i^C ($1 \leqslant i \leqslant k$).

1) Note how this differs from the usual representation, in which a linkage is recursively defined by connecting a single link to another linkage.

3. **Child-to-parent transformations**: each internal sub-assembly A with parent C stores a transformation $\mathbf{T}_A^C$ from its principal handle to the principal handle of its parent.

4. **World transformations**: each sub-assembly C stores a transformation $\mathbf{T}_C$ from its principal handle to the global reference frame.

These transformations are illustrated in Figure 4.2.

4.2.2 Recursive Transformation Updates

In our kinematic representation, the principal transformation of any joint is updated in constant time to match a new joint configuration. It can be shown that the next two types of transformation can be recursively updated, from the leaves of the assembly tree to its root, when the joint configurations evolve.

Assume a linkage C is formed by joining a linkage A with $l+1$ handles H^A, H_1^A, $\ldots$, H_l^A to a linkage B with $m+1$ handles H^B, H_1^B, $\ldots$, H_m^B. The linkage C has $l+m$ handles: the principal handle of C is either a secondary handle of A or a secondary handle of B, while its $l+m-1$ secondary handles H_i^C are the remaining secondary handles in A and B.

Now assume, without loss of generality, that the principal handle H^C of C is the secondary handle H_u^A of A. If a secondary handle of C, H_i^C, is a secondary handle of A, say H_j^A, then $\mathbf{T}_i^C = \mathbf{T}_j^A(\mathbf{T}_u^A)^{-1}$. If, however, H_i^C is also a secondary handle of B, say H_j^B, then $\mathbf{T}_i^C = \mathbf{T}_j^B\mathbf{T}_A^B(\mathbf{T}_u^A)^{-1}$. The principal handle transformations can be computed easily: because $\mathbf{T}_A^C$ is actually equal to $\mathbf{T}_u^A$, and $\mathbf{T}_B^C = \mathbf{T}_u^A(\mathbf{T}_A^B)^{-1}$. Finally, if the world transformation $\mathbf{T}^C$ of C is up-to-date, then $\mathbf{T}^A = \mathbf{T}^C\mathbf{T}_A^C$, and $\mathbf{T}^B = \mathbf{T}^C\mathbf{T}_A^C(\mathbf{T}_A^B)^{-1}$. The case where the principal handle of C is a secondary handle of B is treated similarly.

4.2.3 Bounding Transformations

The recursive computations presented above allow us to determine the positions and orientations of moving bodies over time. Once the principal joint transformations have been updated for a given time t (for instance, by evaluating sine and cosine functions for rotational joints), all the other transformations at time t are computed by multiplying 4×4 homogeneous matrices.

Our CCD algorithm uses these transformations and conservative bounds on them over progressively refined time intervals (cf. Section 4.3). In order to compute these bounds efficiently, we use interval arithmetic [45, 60]. First we bound the elementary functions in the principal joint transformations, and then perform interval counterparts of the matrix multiplications needed to compute the other types of transformations.

4.3 Continuous Collision Detection

Adaptive articulated-body dynamics [56] works by determining and simulating only a relevant subset of joints in the articulated body, which form a *sub-tree* of the assembly tree (the nodes above the dotted line in the assembly trees in Figure 4.3). Thus, at a given time step, only the positions of these nodes can change (or angles, for revolute joints). We will now show how the kinematic representation introduced in Section 4.2 takes advantage of this fact to speed up the computation of the positions and bounds associated with the rigid bodies, and allows us to design a CCD algorithm which benefits from the adaptivity of the simulation. This algorithm shares some similarities with our previous work [60], but the key difference is in the computation of the positions and bounds on the rigid bodies and the exploitation of the adaptivity. Moreover, we will demonstrate self-CCD within the same articulated body as well as CCD between multiple articulated bodies in Section 6.2.2.

Our continuous collision detection algorithm is composed of two main steps: a body-level culling step that uses axis-aligned bounding boxes (AABBs), and an exact contact computation step with a hierarchy of oriented bounding boxes (OBBs) for each rigid body.

4.3.1 AABB Culling

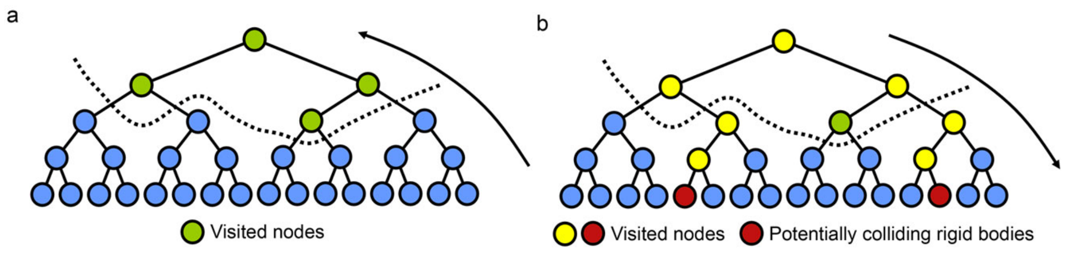


Fig. 4.3: **Adaptive computation of rigid-body transformations using an assembly tree.** *Only the nodes above the dotted curve are being simulated at the current time step, which allows us to limit the transformation updates to a limited number of nodes: (***a***) updating the principal joint, the principal-to-secondary and child-to-parent transformations of the simulated nodes (green nodes); and (***b***) updating the world transformations for the potentially colliding rigid bodies (yellow nodes).*

We begin by computing bounds on the positions of all moving bodies over the current time interval, by recursively bounding the first three types of transformations of all active joints, from the bottom up (the green nodes in Figure 4.3(a)). Once these bounds have been updated, we bound the world transformations of all rigid bodies, by accumulating world transformations over all nodes. We then use these bounds to compute a single AABB for each rigid body, by multiplying the interval world transformations to the vertices of the root OBB which bounds the rigid body. This produces eight AABBs, each of which bounds the trajectory of the OBB vertex over the whole time interval. We then compute the AABB that bounds these eight AABBs. By a simple convexity argument, this AABB also bounds the rigid body over the current time interval. Note that the cost of this step is linear in the number of rigid bodies, but the constant is small because the bounds on the world transformations are only computed once for each rigid body. The AABBs are then used to determine the pairs of potentially colliding rigid bodies (collisions may occur within the same articulated body, or with rigid bodies in the static environment).

4.3.2 Computing Contact Information

Once the potentially colliding rigid bodies have been determined, we can compute the time of first contact and identify the contacting features using interval arithmetic. The key computation in this step is to determine the positions and orientations of the rigid bodies that might collide, as well as to construct conservative bounds on these positions and orientations, over smaller and smaller time intervals. These are used to bound the trajectories of the OBBs (for efficient culling) and of the geometric features (vertices, edges and triangles, for precise contact time computation), of the potentially colliding rigid bodies. This is similar to Step 3 in Redon et al. [60], but we can now perform these computations adaptively, using the simulated joints.

Assume we want to determine the positions and orientations of the potentially colliding rigid bodies at a given time, or over a given time interval. As in the AABB culling step, we start by updating the first three types of transformation for all active joints, from the bottom up (the green nodes in Figure 4.3(a)). However, we now compute the world transformations of the potentially colliding rigid bodies only (the red nodes in Figure 4.3(b)). Thus, we only accumulate the world transformations as we traverse the assembly tree from the top to these rigid bodies (the yellow and red nodes in Figure 4.3(b)).

Assuming the assembly tree of an articulated body with n joints is balanced, an upper bound[2)] on the complexity of computing the world transformations of k rigid bodies when m joints are active is $O(m+k\log(n))$. We show in the next section how this reduced complexity allows us to obtain significant performance improvements over a non-adaptive continuous collision detection approach.

2) In practice, the complexity is smaller since the world transformations of the internal nodes are shared by multiple rigid bodies.

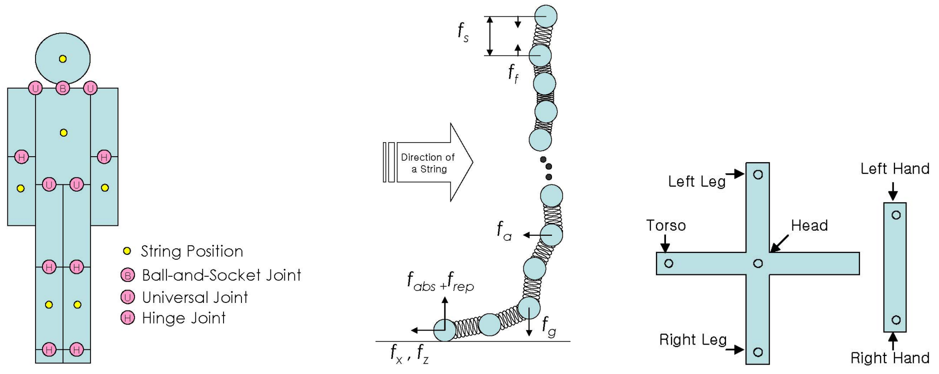
Chapter V

Haptic Puppetry for Interactive Games

In this chapter, as a capable technique for controlling articulated characters, we propose the traditional marionette control [17] 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.

5.1 Virtual Marionette

A real-world marionette is mainly composed of three parts: a control, strings, and a marionette itself [17]. 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



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

Fig. 5.1: **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}.*

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 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.

5.1.1 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

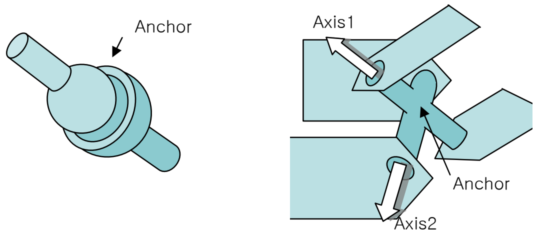


Fig. 5.2: **Different Joint Types.** *ball and socket joint, universal Joint, hinge joint (from left to right)*

Fig. 5.1.

We use three types of joints to connect the bodies as shown in Fig. 5.2 (terms borrowed from [68]): 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 [68]:

$$\begin{aligned} J_1 v_1 + \Omega_1 \omega_1 + J_2 v_2 + \Omega_2 \omega_2 &= c + C\lambda \\ \lambda &\geq l \\ \lambda &\leq h \end{aligned} \tag{5.1}$$

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. 5.1.1 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 [68].

The yellow circles in Fig. 5.1 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 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:

$$f_{body} = -k(x_{body} - x_{string}) - k_d(\dot{x}_{body} - \dot{x}_{string}) \tag{5.2}$$

,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 $f = -f_{body}$.

5.1.2 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

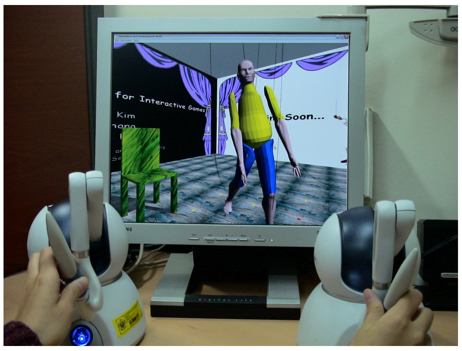
After accumulating all the aforementioned forces (F), we numerically solve the Newtonian second order ordinary differential equation (ODE) for a particle system (i.e., $\ddot{\mathbf{x}} = \frac{\mathbf{F}}{m}$) using the implicit Euler's method [72].

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:

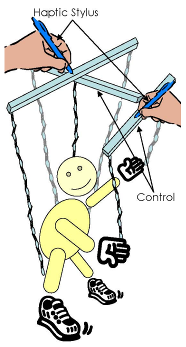
- Our system requires to be highly interactive so that we can not afford such costly collision detection and response time.
- The goal of our system is a character motion control, not string simulation.
- In real-world marionette control, there is no technique using twisted strings requiring collision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

5.1.3 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.



(a) System Setup



(b) Control

Fig. 5.3: **System Setup and Control Mapping.** *(a) shows the system setup and (b) shows of mapping the control to the haptic stylus.*

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. 5.1 (the white circles).

5.2 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. 5.3. As a result, we can provide a more intuitive, easier way to manipulate the marionette, instead of using complicated key combinations.

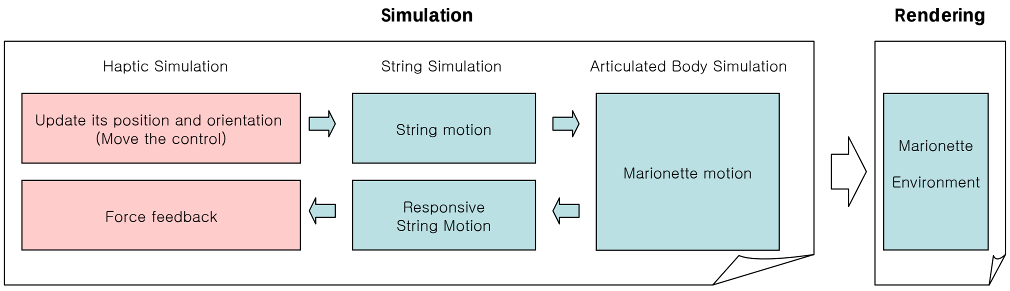


Fig. 5.4: **System Diagram.** *The red block is an asynchronous process whereas the blue blocks are synchronized with each another.*

5.2.1 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. 5.3.

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. 5.3. Furthermore, since each haptic device has its own device coordinate system, we need to get its proper position in world coordinate system.

5.2.2 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:

$$F_i = -k(x_{proxy} - x_{string}) - mg - k_d(\dot{x}_{proxy} - \dot{x}_{string}) \tag{5.3}$$

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.

Chapter VI

Results and Discussion

In previous chapters, we have introduced three relevant approaches for a view-dependent simulation system with haptic interface. In this chapter, we present several benchmarks and applications of our methods and demonstrate their effectiveness.

6.1 Implementation Platform

We have implemented our methods using C++ and OpenGL graphics library under windows XP. The view-dependent dynamics is written in C++ based on Redon et al.'s work [56]. For haptic puppetry methods, additionally, we have used the Open Dynamics Engine (ODE) [68] as a basic physics engine for articulated body dynamics, Sensable's Omni haptic devices and Openhaptics library [67] as haptic APIs.

We have demonstrated our view-dependent dynamics algorithm in benchmarking scenarios on a 2.26GHz Intel Pentium M processor laptop with 2GB RAM. The CCD and haptic puppetry algorithms are performed on a 2.19GHz AMD Opteron PC with 2GB RAM.

6.2 Benchmarks and Applications

6.2.1 View-Dependent Dynamics of Articulated Bodies

We present several benchmarks and demonstrate how our view-dependent articulated-body dynamics method allows an animator (or a physics engine) to finely tune the visual quality and obtain potentially significant speed-ups during interactive or off-line simulations.

Swinging pendulum : a pendulum model consisting of three hundred rigid bodies swings because of gravity. View-dependent dynamics is applied to the swinging motion of a pendulum in two series of tests: one by varying the threshold value for motion metrics (Figure 6.4) and one by varying the viewer's distance to the pendulum (Figure 6.5). As can be seen from the graphs, our method allows the user to finely tune the performance of the view-dependent dynamics by changing the error threshold or, for a given threshold, to benefit from the automatic simplification and corresponding speed-up when the distance to the viewer varies.

Haptic-enabled dog puppet: a toy-like dog model consisting of sixteen rigid bodies is interactively manipulated using a haptic interface (Figure 3.1.**a**). We use Sensable's Omni haptic device and map its end-effector to a virtual control stick attached to the toy dog by virtual strings. As the user interactively controls the toy dog, some links of the dog can be hidden by objects in the environment (Figure 3.1.**b**). Our view-dependent algorithm automatically rigidifies these links (Figure 3.1.**c**).

Hanging toy dogs: 100 toy-like dog models are attached to springs whose other ends are fixed in space. Initially, the dogs are twisted from the equilibrium state in order to create an initial rotational velocity. Then, the dogs are released and create dynamics simulation (Figure 6.3). During the simulation, a random torque is intermittently applied to the dogs.

Falling character: a character consisting of twenty-nine rigid bodies falls on a floor due to gravity. As the viewer moves away from the character, the view-dependent dynamics automatically rigidifies some joints (Figure 6.2) but preserves the overall look of the motion (Figure 6.1).

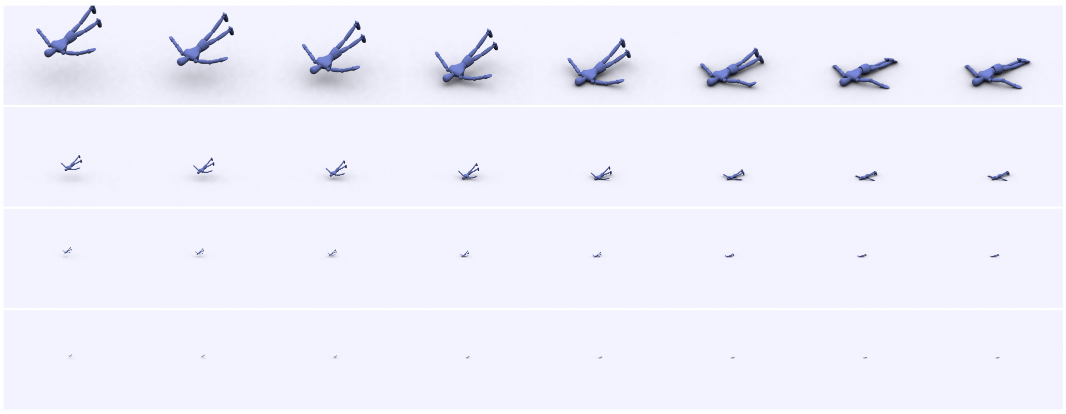


Fig. 6.1: **View-dependent dynamics simplification of a falling character.** *Our algorithm automatically simplifies the dynamics of the falling character while preserving the overall visual aspect of the impact (*e.g. *legs motion, see also Fig. 6.2 for a close-up on the final frames.).*

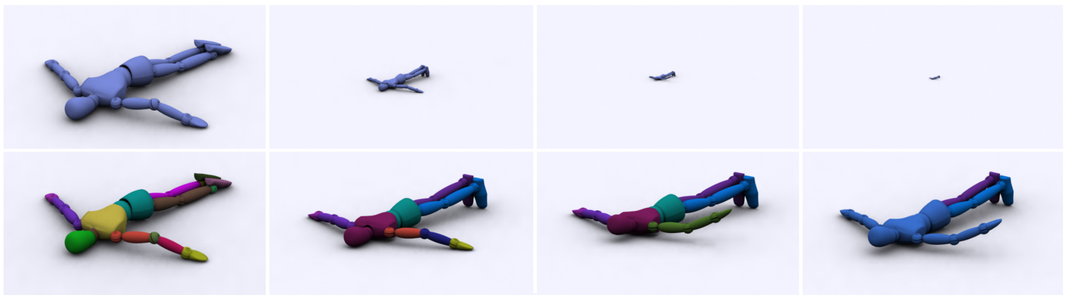


Fig. 6.2: **View-dependent dynamics simplification. Top***: Our algorithm automatically simplifies the dynamics of a falling character as its distance to the viewer increases.* **Bottom***: Corresponding rigidification at this time step (one color per rigid group). See also Fig. 6.1 for the corresponding motion strips.*

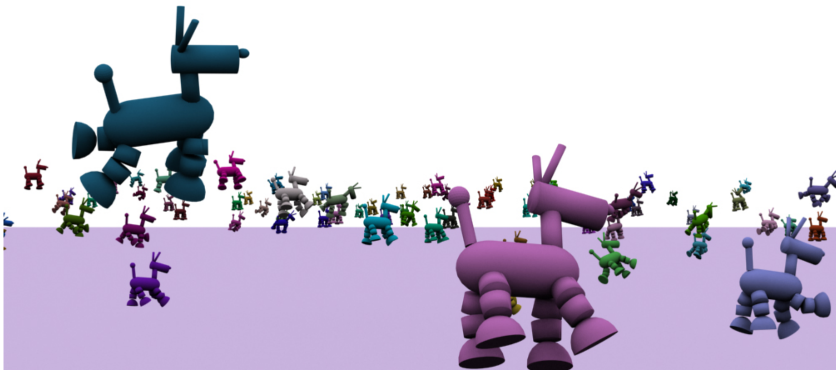


Fig. 6.3: **View-dependent Simulation of 100 Swinging Toy Dogs.** *100 toy-like dogs consisting of 1600 rigid bodies are attached to virtual springs (not shown in the image) in space and simulate dynamics in a view-dependent manner. In this scene, as many dogs are occluded by other dogs, clipped against the viewport, or seen far from the viewer, the number of simulated, active joints is reduced from 1500 to 493 on average without incurring visual deterioration in the simulation. Each simulation frame takes 14 msec on average.*

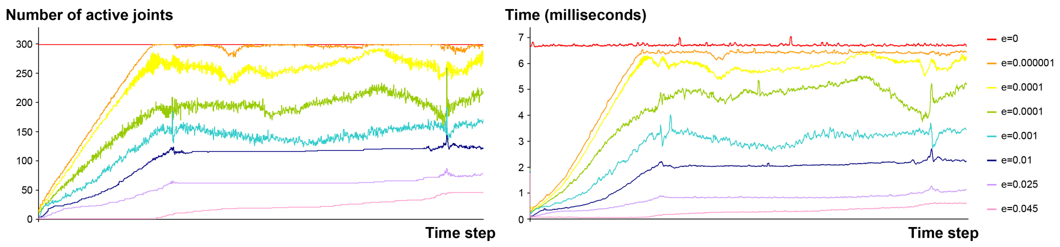


Fig. 6.4: **Performance of our algorithm depending on visual error thresholds.** *As the visual error threshold e increases, the number of active nodes is automatically decreased by the view-dependent algorithm (***left***), and the computational cost is reduced (***right***).*

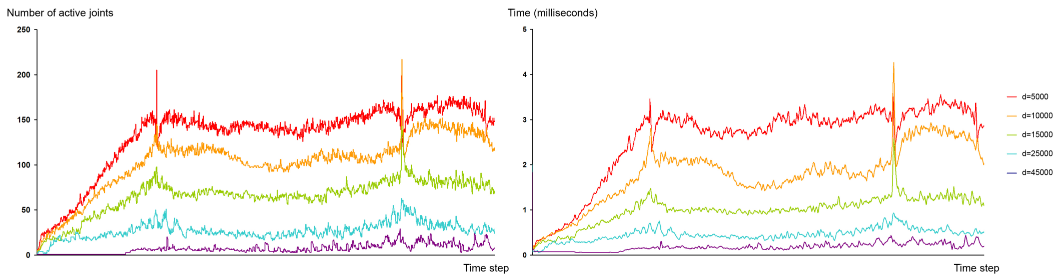


Fig. 6.5: **Performance of our algorithm depending on the viewer's distance.** *Our algorithm automatically decreases the number of active joints when the distance d between the viewer and the pendulum increases (***left***), reducing the computational cost of the simulation (***right***).*

6.2.2 Continuous Collision Detection for Adaptive Simulation of Articulated Bodies

We have implemented our CCD algorithm within an adaptive dynamics framework [56]. We will now assess the performance of our algorithm in different benchmarking scenarios[1], called wooden men, swinging pendulum and falling wooden man, by varying the number of active joints in the simulation (Figure 6.6). The complexities of the benchmarking models are summarized in Table 6.1.

Wooden men: a pair of mannequins are pulled together by the spring that connects them. Initially, the mannequins are placed at random configurations.

Pendulum: a pendulum consisting of many small balls swings under gravity. In this benchmark, we check for self-collision between each pair of balls.

Falling wooden man: a mannequin is falling from the sky under gravity and collides with obstacles such as pots and plates on the ground.

In these benchmarks, the models' dynamics are governed by adaptive dynamics with different numbers of active joints, and the time for collision detection are measured and averaged over several (e.g. 50) runs. Figure 6.7 shows the resulting timings for each scenario.

Benchmarks	Models	Tris	Links
Wooden Men	Wooden man	11K	29
	Wooden man	11K	29
Pendulum	Pendulum	119K	30
Falling Wooden Man	Wooden man	11K	29
	Environmental Obstacles	153K	-

Table 6.1: **Model complexities of benchmarking models.** *The third and fourth column show the triangle counts and number links of the benchmarking models, respectively.*

1) The accompanying videos can be seen at *http://graphics.ewha.ac.kr/CCD4AD*

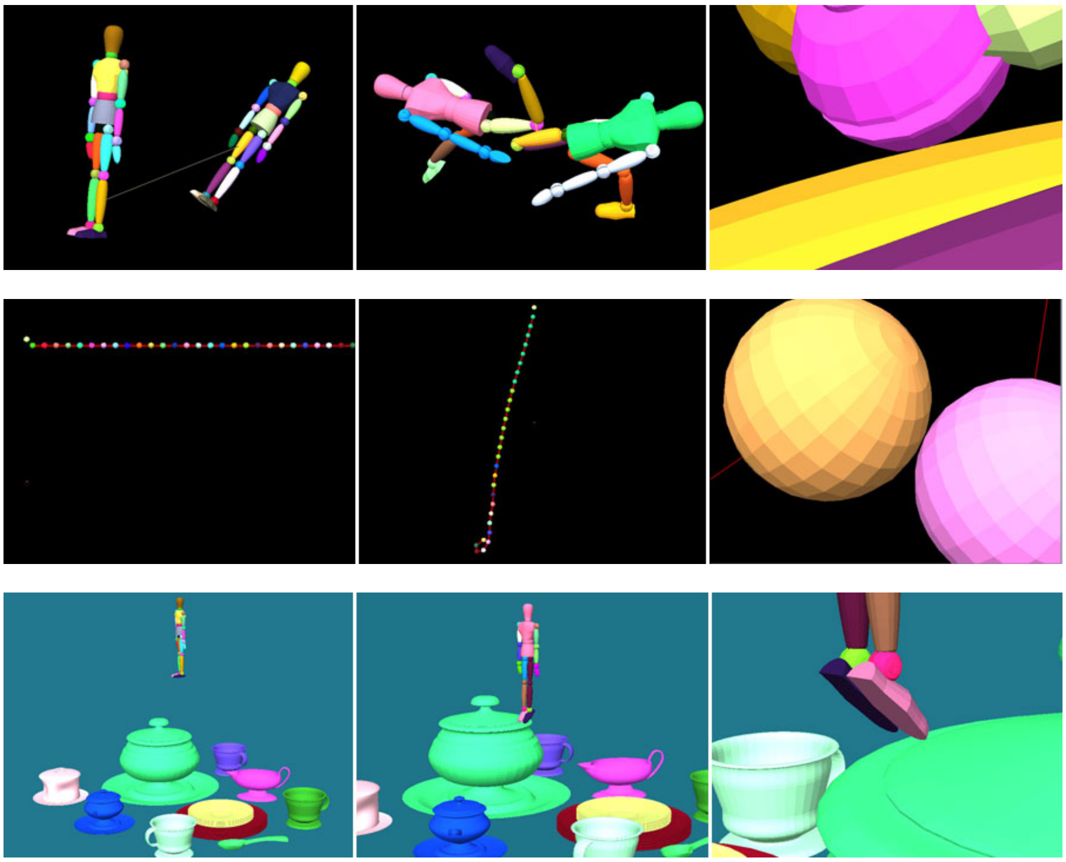
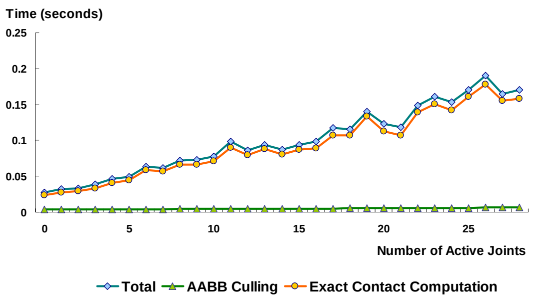
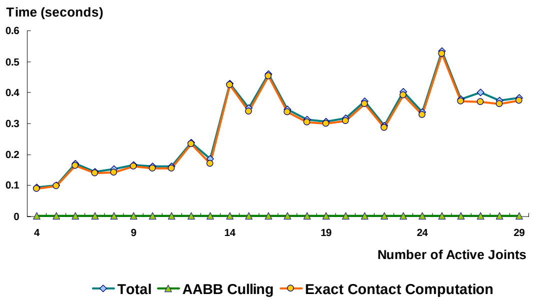


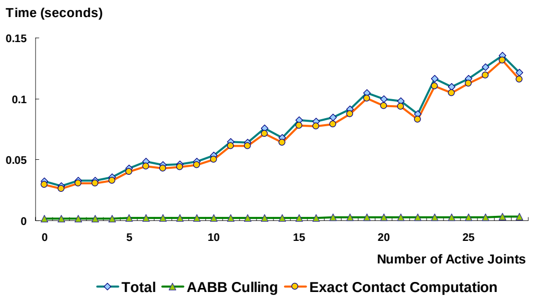
Fig. 6.6: **Benchmarking examples.** *In these figures, the number of active joints is 15 and rigid bodies with identical colors belong to the same group of inactive links. Also, for each row of the figures, the third column is a zoomed-up version of the second column to show collision area.* **Top row:** *two wooden mannequin models consisting of 29 rigid bodies, 11K triangles for each model are pulled together by a spring-like string and collided with each other.* **Middle row:** *a pendulum consisting of 30 rigid bodies and 119K triangles swings because of gravity and collides with itself.* **Bottom row:** *a wooden mannequin falls under gravity and collides with static tableware. The static environment consists of 153K triangles in total.*



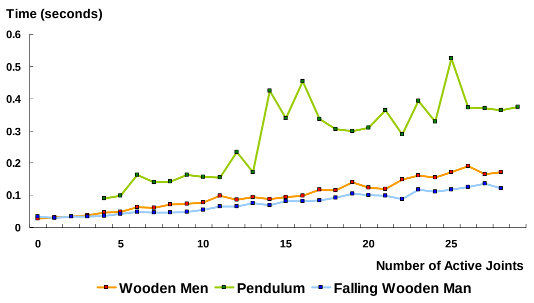
(a) Wooden Men



(b) Pendulum



(c) Falling Wooden Man



(d) Comparison

Fig. 6.7: **CCD timings for three benchmarks and a comparison between them.** *Each graph shows a timing profile of CCD performance with AABB culling and exact contact computation steps for three benchmarking scenarios, and their comparison. The CCD time complexity gradually increases as a function of the number of active joints.*

6.2.3 Haptic Puppetry

We have implemented the haptic puppetry system, and our experimentations show that our system can create reasonably complicated motions for articulated characters in an easy and quick manner at highly interactive rates$^{2)}$.

To perform articulated body dynamics for a marionette at interactive rates, we bound each

2) The accompanying videos can be seen at *http://graphics.ewha.ac.kr/HPuppetry*

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) [68] 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. 5.4. 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 [55].

6.3 Discussions

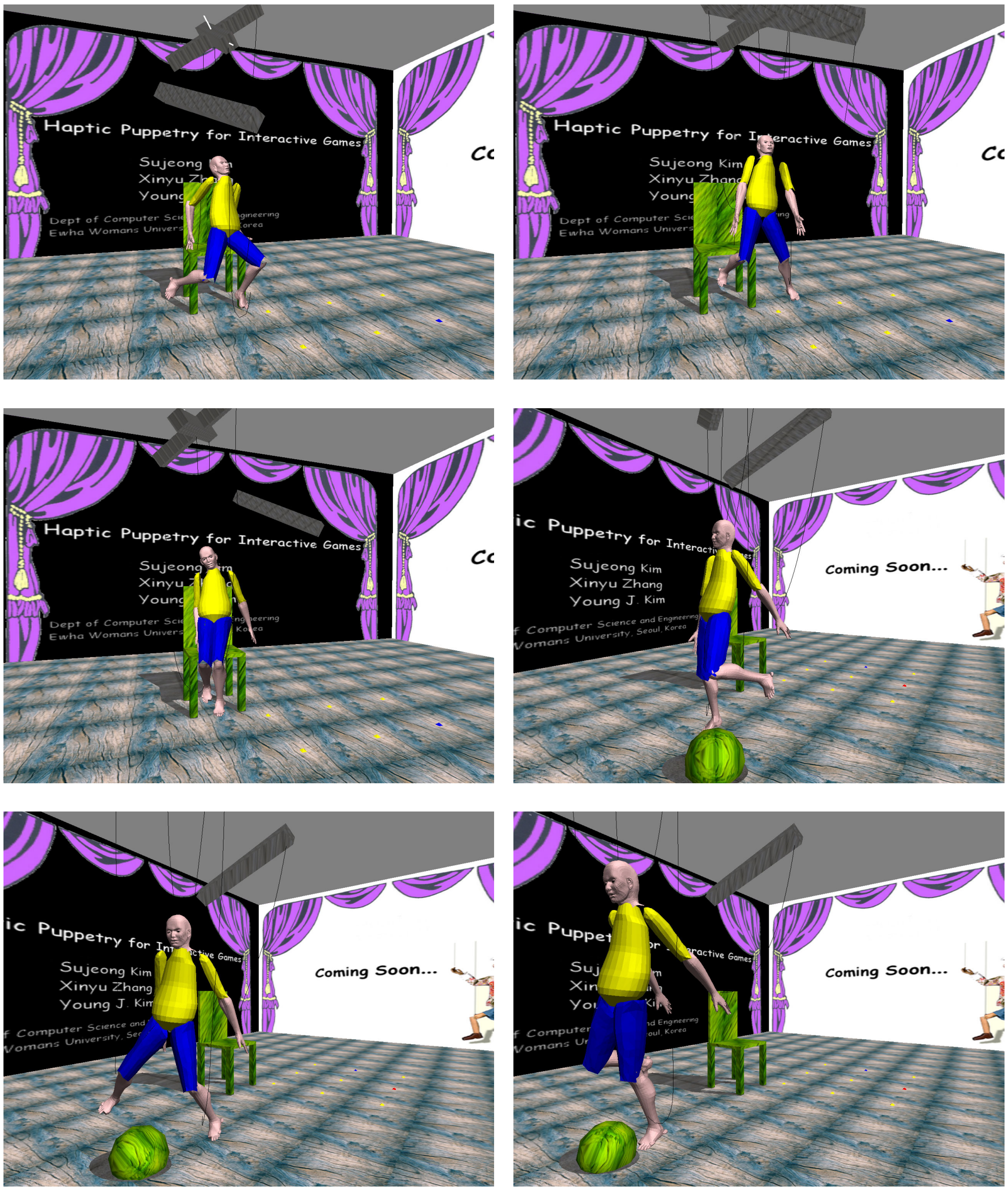

Haptic Puppetry for Interactive Games
Sujeong Kim
Xinyu Zhang
Young J. Kim
Dept of Computer Science and Engineering
Ewha Womans University, Seoul, Korea
Coming Soon...

Fig. 6.8: **Animation sequences in our virtual marionette system.** *The virtual marionette is standing up from a chair, and kicking and chasing a ball.*

Now, we discuss the implementation results of our view-dependent simulation of articulated bodies with force feedback.. Each of them shows that the capability of our methods.

Firstly, the view-dependent dynamics simulation shows how our view-dependent articulated-body dynamics method allows an animator (or a physics engine) to finely tune the visual quality and obtain potentially significant speed-ups during interactive or off-line simulations. Notice that our view-dependent dynamics performs exactly like adaptive dynamics [54] except that the adaptivity is automatically determined by the visibility of the simulated bodies.

The view-dependent dynamics algorithm presented in this dissertation has essentially two limitations:

- Our view-dependent metric is only semi-predictive. We do not currently have a (not overly) conservative way to bound the visual error caused by a simplification. We only estimate this error a priori, by making an assumption on the effect of a joint acceleration on the global articulated-body motion.
- Because of the way we combine acceleration metrics with visibility estimations, the error threshold set by the user is not as intuitive as it should be. However, the graceful, “continuous” degradation of the dynamics produced by our algorithm when the threshold or distance vary makes it relatively easy to choose this parameter.

Secondly, as expected, the results of CCD benchmarks show that the time required for collision detection is roughly linearly related to the number of active joints in the articulated body. Notice that, when all the joints in an articulated body are active (e.g. 29 for the wooden man), our CCD algorithm is essentially the same as that of Redon et al. [60]. However, as the number of active joints is reduced, the relative performance of our CCD algorithm improves, as we have seen in Figure 6.7. Finally, with our haptic puppetry system, an animator or a puppeteer can easily sketch motions for an articulated character as shown in Fig. 6.8. 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. 6.8, a virtual marionette is manipulated to stand up from a chair, kick a ball and chase it.

Chapter VII

Conclusion

In this dissertation, we have addressed the issues for view-dependent simulation of articulated bodies with haptic feedback. We have implemented and demonstrated our work in challenging benchmarking scenarios.

In chapter 3, we have introduced a method for view-dependent simulation of articulated-body dynamics. We have first discussed the general problem of simplifying a simulation based on visual criteria, a question which seems to have received relatively little attention in the past. We have showed how this problem raises a number of new challenges, even though it is strongly related to the well-known geometric simplification problem. Focusing on articulated-body dynamics, we have proposed *semi-predictive motion metrics*, which combine predictive error metrics based on joint accelerations with visibility estimations. The metrics, based on bounding volume hierarchies or graphics hardware visibility queries, allow us to automatically simplify a simulation based on visual criteria.

In chapter 4, We have introduced a CCD algorithm for the adaptive dynamics simulation of articulated bodies. This algorithm uses a novel hierarchical representation of the kinematics of an articulated body, which can be selectively updated during an adaptive simulation. By simplifying the dynamics that have to be considered, we can reduce the complexity of the computation of the first time of contact, and of the contact information.

In chapter 5, 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.

The view-dependent dynamics simulation shows that our approach allows a user to finely trade between visual quality and performance. The dynamics are gracefully simplified as the distance to the viewer, or the error threshold, is increased. From the experimental results of our CCD algorithm, we demonstrate that this strategy leads to a worthwhile performance improvement if the dynamics can be significantly simplified. And finally, we have implemented real-time haptic puppetry system using dual haptic devices as input devices, and show how the new interface is applied to physically-based articulated body simulation and quickly generate complex motions.

With these approaches, we give new and effective solutions for articulated body simulation and its interactive control. The resulting system is an interactive character control system with haptic feedback which provides a fast and realistic simulation resulted from view-dependent simplification.

As future work, we would like to address the limitations of view-dependent simulation mentioned in chapter 3., which include considering perceptual factors (*e.g.* [65]). In particular, we would like to perform user studies and examine how these studies could lead to more general view-dependent simulation methods. And for the CCD algorithm, we plan to investigate several applications of this work, in particular to haptics and motion planning.

For haptic puppetry, we would like to apply our technique to other types of puppets (e.g., non-string type puppet). 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.

References

[1] A. Bar-Lev, A.M.B., Elber, G.: Virtual marionettes: A system and paradigm for real-time 3D animation. Tech. rep., Technion, I.I.T., Israel (2004)

[2] Abdel-Malek, K., Blackmore, D., Joy, K.: Swept volumes: foundations, perspectives, and applications. International Journal of Shape Modeling (2002)

[3] Agarwal, P.K., Basch, J., Guibas, L.J., Hershberger, J., Zhang, L.: Deformable free space tiling for kinetic collision detection. In: Workshop on Algorithmic Foundations of Robotics, pp. 83–96 (2001)

[4] Bae, D., Haug, E.: A recursive formulation for constrained mechanical systems dynamics: Part 1. open-loop systems. Mechanical Structures and Machines, Vol. 15(3), pp. 359-382 (1987)

[5] Baraff, D.: Fast contact force computation for nonpenetrating rigid bodies. In: A. Glassner (ed.) Proc. SIGGRAPH ’94, pp. 23–34 (1994). ISBN 0-89791-667-0

[6] Basdogan, C., Srinivasan, M.: Virtual Environments HandBook, chap. Haptic rendering in virtual environments (2001)

[7] Beaudoin, J., Keyser, J.: Simulation levels of detail for plant motion. In Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation (2004)

[8] van den Bergen, G.: Ray casting against general convex objects with application to continuous collision detection. Journal of Graphics Tools (2004)

[9] Bertails, F., Kim, T.Y., Cani, M.P., Neumann, U.: Adaptive wisp tree - a multiresolution control structure for simulating dynamic clustering in hair motion. In Proceedings of ACM-SIGGRAPH/Eurographics Symposium on Computer Animation (2003)

[10] Boulic, R., Fua, P., Herda, L., Silaghi, M., Monzani, J., Nedel, L., Thalmann, D.: An anatomic human body for motion capture. In: EMMSEC (1998)

[11] Brandl, H., Johanni, R., Otter, M.: A very efficient algorithm for the simulation of robots and similar multibody systems without inversion of the mass matrix. IFAC/IFIP/IMACS Symposium, pp. 95-100 (1986)

[12] Canny, J.F.: Collision detection for moving polyhedra. IEEE Trans. Pattern Analysis and Machine Intelligence **8**, 200–209 (1986)

[13] Carlson, D.A., Hodgins, J.K.: Simulation levels of detail for real-time animation. In Proceedings of Graphics Interface (1997)

[14] Chenney, S., Forsyth, D.: View-dependent culling of dynamic systems in virtual environments. In: Proc. ACM Symposium on Interactive 3D Graphics (1997)

[15] Chenney, S., Ichnowski, J., Forsyth, D.A.: Dynamics modeling and culling. IEEE Computer Graphics and Applications (1999)

[16] Choi, Y.K., Wang, W., Liu, Y., Kim, M.S.: Continuous collision detection for elliptic disks. IEEE Trans. on Robotics (2006)

[17] Currell, D.: Making and Manipulating Marionettes. The Crowood Press (2004)

[18] 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)

[19] Debunne, G., Desbrun, M., Cani, M.P., Barr, A.H.: Dynamic real-time deformations using space and time adaptive sampling. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques (2001)

[20] Erleben, K., Sporring, J., Henriksen, K., Dohlmann, H.: Physics Based Animation. Charles River Media (2005)

[21] Faure, F.: Fast iterative refinement of articulated solid dynamics. IEEE Trans. on Visualization and Computer Graphics **5**(3), 268–276 (1999)

[22] Featherstone, R.: Robot Dynamics Algorithms. Kluwer, Boston, MA (1987)

[23] Featherstone, R.: A divide-and-conquer articulated body algorithm for parallel o(log(n)) calculation of rigid body dynamics. part 1: Basic algorithm. International Journal of Robotics Research **18**(9), 867–875 (1999)

[24] Featherstone, R.: A divide-and-conquer articulated body algorithm for parallel o(log(n)) calculation of rigid body dynamics. part 2: Trees, loops, and accuracy. International Journal of Robotics Research **18**(9), 876–892 (1999)

[25] Featherstone, R., Orin, D.E.: Robot dynamics: equations and algorithms. IEEE International Conference on Robotics and Automation, pp. 826-834 (2000)

[26] Gottschalk, S., Lin, M.C., Manocha, D.: Obbtree: a hierarchical structure for rapid interference detection. In ACM Transactions on Graphics (SIGGRAPH 1996) (1996)

[27] Grinspun, E., Krysl, P., Schroeder, P.: Charms: a simple framework for adaptive simulation. ACM Transactions on Graphics, 21(3) (2002)

[28] Harrison, J., Rensink, R.A., Panne, M.V.D.: Obscuring length changes during animated motion. ACM Transactions on Graphics 23(3) (2004)

[29] Healey, M.: Ragdoll Kungfu. *http://www.ragdollkungfu.com/*

[30] Hollerbach, J.: A recursive lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity. IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-10, No. 11 (1980)

[31] Hoppe, H.: View-dependent refinement of progressive meshes. ACM Transactions on Graphics (SIGGRAPH 1997 Proceedings) (1997)

[32] Jorissen, P., Wijinants, M., Lamotte, W.: Dynamic interactions in physically realistic collaborative virtual environments. IEEE Transactions on Visualization and Computer Graphics **11**(6), 649–660 (2005)

[33] Kim, B., Rossignac, J.: Collision prediction for polyhedra under screw motions. In: ACM Conference on Solid Modeling and Applications (2003)

[34] Kim, D., Guibas, L., Shin, S.: Fast collision detection among multiple moving spheres. IEEE Trans. on Visualization and Computer Graphics **4**(3), 230–242 (1998)

[35] Kim, Y.J., Otaduy, M.A., Lin, M.C., Manocha, D.: Six-degree-of-freedom haptic display using incremental and localized computations. Presence **12**(3) (2003)

[36] Kirkpatrick, D., Snoeyink, J., Speckmann, B.: Kinetic collision detection for simple polygons. In: Proc. of ACM Symposium on Computational Geometry, pp. 322–330 (2000)

[37] Kovar, L., Gleicher, M., Pighin, F.: Motion graphs. In: ACM SIGGRAPH (2002)

[38] Lasseter, J.: Principles of traditional animation applied to 3D computer animation. In: Proc. of ACM SIGGRAPH (1987)

[39] Laszlo, J., Panne, M., Fiume, E.: Interactive control for physically-based animation. In: Proceedings of SIGGRAPH 2000, pp. 201–209 (2000)

[40] Lee, C.H., Varshney, A., Jacobs, D.W.: Mesh saliency. In ACM Transactions on Graphics (SIGGRAPH 2005), 24(3) (2005)

[41] Li, L., Volkov, V.: Cloth animation with adaptively refined meshes. In Proceedings of the Twenty-eighth Australasian conference on Computer Science (2005)

[42] Losasso, F., Gibou, F., Fedkiw, R.: Simulating water and smoke with an octree data structure. ACM Transactions on Graphics (SIGGRAPH 2004 Proceedings) (2004)

[43] Luebke, D., Reddy, M., Cohen, J., Varshney, A., Watson, B., Huebner, R.: Level of detail for 3d graphics. Morgan Kaufmann Publishers (2003)

[44] McMillan, S., Orin, D.E.: Efficient computation of articulated-body inertias using successive axial screws. IEEE Trans. on Robotics and Automation, 11, pp. 606-611 (1995)

[45] Moore, R.E.: Interval Analysis. Prentice Hall, Englewood Cliffs, New Jersey (1966)

[46] NVIDA: SDK White Paper - Occlusion Query, Checking for Hidden Pixels. NVIDIA (2004)

[47] O'Brien, D., Fisher, S., Lin, M.C.: Automatic simplification of particle system dynamics. In Proceedings of Computer Animation (2001)

[48] Oore, S., Terzopoulos, D., Hinton, G.: A Desktop Input Device and Interface for Interactive 3D Character Animation. In: Proc. Graphics Interface, pp. 133–140 (2002)

[49] Ortega, M., Redon, S., Coquillart, S.: A six degree-of-freedom god-object method for haptic display of rigid bodies. In: IEEE International Conference on Virtual Reality (2006)

[50] Oshita, M.: Pen-to-mime: A pen-based interface for interactive control of a human figure. In: Sketch-Based Interfaces and Modelling, pp. 43–52 (2004)

[51] O'Sullivan, C.: Collisions and attention. ACM Transactions on Applied Perception (2005)

[52] O'Sullivan, C., Dingliana, J.: Collisions and perception. ACM Transactions on Graphics, 20(3) (2003)

[53] Perbet, F., Cani, M.P.: Animating prairies in real-time. In Proceedings of the 2001 symposium on Interactive 3D graphics (2001)

[54] Redon, S., Gallopo, N., Lin, M.C.: Adaptive dynamics of articulated bodies. In ACM Transactions on Graphics (SIGGRAPH 2005), 24(3) (2005)

[55] Redon, S., Galoppo, N., Lin, M.C.: Adaptive dynamics of articulated bodies. In: Proceedings of SIGGRAPH 2005 (2005)

[56] Redon, S., Galoppo, N., Lin, M.C.: Adaptive dynamics of articulated bodies. ACM Trans. on Graphics (SIGGRAPH 2005) **24**(3) (2005)

[57] Redon, S., Kheddar, A., Coquillart, S.: An algebraic solution to the problem of collision detection for rigid polyhedral objects. Proc. IEEE Conf. on Robotics and Automation (2000)

[58] Redon, S., Kheddar, A., Coquillart, S.: Fast continuous collision detection between rigid bodies. Proc. Eurographics (Computer Graphics Forum) (2002)

[59] Redon, S., Kheddar, K., Coquillart, S.: Gauss' least constraints principle and rigid body simulation. Proc. International Conference on Robotics and Automation (2002)

[60] Redon, S., Kim, Y.J., Lin, M.C., Manocha, D.: Fast continuous collision detection for articulated models. In: Proc. ACM Symposium on Solid Modeling and Applications (2004)

[61] Redon, S., Kim, Y.J., Lin, M.C., Manocha, D.: Interactive and continuous collision detection for avatars in virtual environments. In: Proc. IEEE Virtual Reality (2004)

[62] Redon, S., Lin, M.C.: An efficient, error-bounded approximation algorithm for simulating quasi-statics of complex linkages. In: Proc. ACM Symposium on Solid and Physical Modeling (2005)

[63] Redon, S., Lin, M.C.: Practical local planning in the contact space. In Proc. IEEE International Conf. on Robotics and Automation (2005)

[64] Redon, S., Lin, M.C.: An efficient, error-bounded approximation algorithm for simulating quasi-statics of complex linkages. In Computer-Aided Design, 38, pp. 300-314, Elsevier (2006)

[65] Reitsma, P.S.A., Pollard, N.S.: Perceptual metrics for character animation: Sensitivity to errors in ballistic motion. ACM Transactions on Graphics (SIGGRAPH 2003 Proceedings) **22(3)**, 537–542 (2003)

[66] Schwarzer, F., Saha, M., Latombe, J.C.: Exact collision checking of robot paths. In: Workshop on Algorithmic Foundations of Robotics (WAFR) (2002)

[67] SensAble: 3D Touch SDK OpenHaptics™ toolkit version 1.02 API reference. Http://www.sensable.com

[68] Smith, R.: Open Dynamics Engine user guide (2004)

[69] Thorne, M., Burke, D., Panne, M.: Motion doodles: An interface for sketching character motion. In: Proc. of ACM SIGGRAPH (2004)

[70] Vodislav, D.: A visual programming model for user interface animation. In: Visual Languages, pp. 348–355 (1997). URL citeseer.ist.psu.edu/105836.html

[71] Ward, K., Lin, M., Joohi, L., Fisher, S., Macri, D.: Modeling hair using level-of-detail representations. In Proceedings of Computer Animation and Social Agents (2003)

[72] Witkin, A., Baraff, D.: Physically based modeling: Principles and practice. In: SIGGRAPH Course Note (1997)

[73] Zhang, X., Lee, M., Kim, Y.J.: Interactive continuous collision detection for non-convex polyhedra. The Visual Computer (Proc. Pacific Graphics) **22**, 9–11 (2006)

[74] 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)

논문 개요

최근 관절체 시뮬레이션은 가상환경상의 캐릭터를 효율적으로 모델링 하는 방법으로 많이 이용되고 있다. 그 중에서도 물리 기반의 관절체 시뮬레이션은 캐릭터의 움직임과 가상 환경 내에서 일어나는 상호작용을 물리 법칙에 기반해 사실적이고 자연스럽게 보여준다.

물리기반 관절체 시뮬레이션과 관련된 연구 주제 중, 많은 수의 캐릭터를 빠르게 시뮬레이션하는 것과 쉽고 효과적으로 캐릭터를 컨트롤할 수 있는 인터페이스의 디자인은 중요한 문제로 인식되어 왔다. 본 논문에서는 이 문제에 대한 새로운 접근 방법으로 다음과 같은 세 가지 기법을 제안한다.

첫째로, 관찰자의 시각적 지각 정도에 따라 시뮬레이션의 복잡도를 자동적으로 조정하는 시점 기반 관절체 시뮬레이션을 제안한다. 이 방법은 시각적 기준을 바탕으로 한 유사 예측 알고리즘을 통하여 효율적인 시뮬레이션을 제공한다.

둘째로 적응적 시뮬레이션에서의 연속적 충돌검사 알고리즘을 소개한다. 이 알고리즘은 관절체의 움직임을 나타내는 새로운 자료구조를 바탕으로, 시점기반 동역학 시뮬레이션 상에서 충돌이 일어난 시점을 효율적으로 구한다.

마지막으로 높은 자유도의 캐릭터를 조작하여 복잡한 움직임을 생성할 수 있는 새로운 사용자 인터페이스로 줄인형 조작법과 햅틱 인터페이스를 결합하였다. 이 방법은 쉽고 직관적인 조작법을 제공하며, 포스 피드백을 통해 가상 환경상의 관절체를 인터랙티브하게 조작할 수 있도록 한다.

제안된 알고리즘을 바탕으로 다양한 벤치마킹을 구현하고 평가하였다. 그 결과를 통해 본 논문에서 제안된 기법들이 많은 수의 관절체의 물리 기반 시뮬레이션, 그리고 인터랙티브한 컨트롤에 대한 빠르고 효율적인 해결책을 제공한다는 것을 확인할 수 있다.

감사의 글

저의 길을 예비하시고 이끌어주시는 하나님께 감사드립니다.

부족함 많은 저를 가르쳐주시고 많은 기회를 주신 김영준 교수님께 감사드립니다. 제가 좋아하는 일을 찾을 수 있게 해주시고, 그 일을 할 수 있는 능력을 키울 수 있도록 해 주신 것 감사드립니다.

바쁘신 중에도 제 논문을 검토해주시고 격려와 조언을 해주신 김명 교수님과 용환승 교수님, 학부와 대학원 강의로 가르침을 주신 이화여대 컴퓨터학과의 이상호 교수님, 조동섭 교수님, 김명희 교수님, 채기준 교수님, 박승수 교수님, 최병주 교수님, 이미정 교수님, 박현석 교수님, 이민수 교수님, 반효경 교수님, 그리고 저에게 소중한 가르침을 주신 이화여대의 모든 교수님들께 감사드립니다.

프랑스에서의 공동연구기간 동안 세심하게 지도해주시고 그 후에도 연구하는 데 많은 도움을 주신 INRIA의 Stephane Redon 박사님께 감사드립니다. 박사님께 많은 것을 배웠고, 박사님께서 친절히 챙겨주신 덕분에 프랑스에서 좋은 추억을 남길 수 있었습니다.

힘들때마다 좋은 말씀 해주셨던 김대현 박사님과 최유주 박사님, 그리고 중국어를 가르쳐주시고 항상 모범을 보여주셨던 Zhang Xinyu 박사님께 감사를 드립니다.

모든 분들의 가르침으로 제가 더욱 성장할 수 있었던 것 같습니다.

서로 격려하면서 열심히 일했던 연구실 친구들 - 혜정, 민경, 연희. 옆에서 지켜봐주시고 격려해주시고 항상 맛있는 것을 해주셨던 미경언니.

일년 동안이었지만 같이 공부하면서 재미있는 추억도 많이 만들었던 은하언니, 은정언니. 모르는 것이 있을 때마다 많은 도움을 주셨던 소윤언니, 친언니같이 챙겨주신 나래언니, 정민언니, 혜령언니.

어디 있든지 생각나고 그리운 친구들 - 고오스, 용사, 퉤지혜, 김김주, 염소. 그리고 심언니.

옥계초등학교, 적암초등학교, 봉암초등학교, 아라초등학교, 유영초등학교, 충무여자중학교,

창원여자중학교, 신명여자중학교, 혜화여자고등학교, 미림여자고등학교에서 만났던 너무나 소중한 모든 친구들...

멀리 있지만 항상 힘을 주는 친구들 - Lenin, Nina, Nao..

프랑스에서 항상 같이 다니면서 많은 것들을 함께한 Ramya와 Anca, 같이 즐겁게 일했던 I3D 팀 멤버들 - Andreas, Michael, Romain, Sandy 그리고 Sabine 박사님. 음악과 미술과 만들기를 좋아하는 Laurence. 많은 도움을 주셨던 유진 언니와 국진 오빠. 그 밖에 INRIA와 프랑스에서 만난 모든 친구들...

모두들 곁에 있어주어서 감사합니다.

마지막으로 항상 저를 믿고 사랑해주시고 기도해주시는 부모님, 착하고 듬직한 동생.

그리고 저를 위해 기도해주시는 모든 분들께 진심으로 감사드립니다.