INTERACTIVE AVATAR CONTROL: CASE STUDIES ON PHYSICS AND PERFORMANCE BASED CHARACTER ANIMATION

STEVIE GIOVANNI

(B.Sc. Hons.)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2013

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Stevie Giovanni

3 January 2013

ACKNOWLEDGMENTS

I would like to take this chance to express my sincere gratitude to my supervisor, Dr. KangKang Yin, for her patience and guidance on this report. She has given good advices in terms of the structures and technical content of the paper. She has also greatly inspired me on my research topic and helped me on the preliminary work by giving me the chance to work with her and getting more familiar with the field. Without her efforts, I would not have been able to complete the work in this report.

Also, I would like to thank my family, especially my mother, for all the support they have given me throughout my master's studies. They encouraged me to keep moving forward during my ups and downs, and their constant faith in me will forever be my motivation to face the hardships I might encounter in the future.

And finally, I would like to give special thanks to Terence Pettit for being a very good friend and for the time he dedicated to help me proofread this report.

TABLE OF CONTENTS

SUMMARY

This paper focuses on the problem of interactive avatar control. Interactive avatar control is a subclass of character animation wherein users are able to control the movement and animation of a character given its virtual representation. Movements of an avatar can be controlled in various ways such as through physics, kinematics, or performance-based animation techniques. Problems in interactive avatar control involve how natural the movement of the avatar is, whether the algorithm can work in real time or not, and how much details in the avatar's motions the algorithm allows users to control. In automated avatar control, it is also necessary for the algorithm to be able to respond to unexpected changes and disturbances in the environment.

For our literature review, we begin by exploring existing work on character animation in general from three different perspectives: 1) kinematic; 2) data-driven; 3) physics-based character animation. This will then serve as the starting point to discuss two of our works that focus on deploying a physics-based controller to different simulation platforms for automated avatar control, and performance-based avatar control for a virtual try-on experience. With our work, we illustrate how character animation techniques can be employed for avatar control.

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Character animation has become a major evolving area of focus in computer science. The main concern in character animation is to design algorithms to synthesize motions for virtual characters through kinematic, data-driven, or physics-based approaches. Among the different subfields of character animation is interactive avatar control. The focus of interactive avatar control is not just to produce motions for the characters, but also to provide users with an interactive method to control the synthesized motions. Throughout the years, ongoing research has delved with major issues on avatar control. These issues involve how fast is the algorithm, how natural is the motion compared to real life characters, how much details in the avatar's motions the algorithm allows users to control, and is the character smart enough to response to unexpected changes and disturbances in the environment.

For interactive avatar control itself, there are more than one method of control that can be provided to users. In one way, controls of the character's motions can come in the form of offline control parameters that users can specify at the beginning of a physics-based character animation. Examples of these control parameters include the speed and style in walking animations, and the height of a jump in jumping animations. This interactive avatar control approach which allows users freedom of control over a physically simulated character animation is often called physics-based avatar control.

Another way to control an avatar interactively is to map the motions of real human actors directly to the avatar on the fly. Recent advances in technology have enabled us to perform skeletal tracking on multiple humans using cheap devices such as the Kinect sensor. The information obtained from skeletal tracking (e.g. joint position and orientation) can be directly mapped to the avatar's joints to control its movement. This approach is called performance-based avatar control.

Online avatar control can also be done using kinematic approaches by providing users with controls over the position and orientation of some or all of the joints in the character's skeletal structure, or by using target points to control the character's end effectors. In animation software such as Maya for example, users often use similar structures called rigs to animate virtual characters interactively.

In this paper we provide case studies for both physics-based and performance-based avatar control. Before we move further on the topic of interactive avatar control, we first review existing work on character animation as the foundation of our work. The rest of this paper is organized as follows. Section 2 contains literature review on existing work on character animation. Section 3 describes preliminary work on porting a physics-based controller framework to various simulation engines for avatar control. In section 4, we explain our work on using performance-based avatar control for interactive virtual try-on application. We conclude the paper in section 5 with some insights on future work.

CHAPTER 2

LITERATURE REVIEW

In this section we discuss previous work related to character animation. This section serves to provide necessary background for our work on interactive avatar control. The objective is to cover approaches and techniques that have been done in the past few years regarding character animation. This section is divided into four sections covering the work on kinematic character animation, data-driven character animation, and physics-based character animation. We include a separate subsection to discuss interactive character animation interface, a growing subfield in character animation that serve as fundamental for performance-based avatar control.

2.1 KINEMATIC CHARACTER ANIMATION

We begin by describing character animation in its simplest form. Character animation is the process of animating one or more characters in an animated work. From computer science perspective, it involves using algorithms to create motions of artificial characters to be visualized using computer graphics. Most of the time, the characters are represented as rigid bodies (links) connected with joints. The collection of rigid bodies and joints forms a hierarchy of a multibody character (Figure 2.1). Some links serve as manipulators that interact with the environment. These links, often the lasts in the hierarchy, are called end-effectors.

Given this character configuration, we can animate the character in two ways. First, by specifying the position and orientation of the root, and the values of the

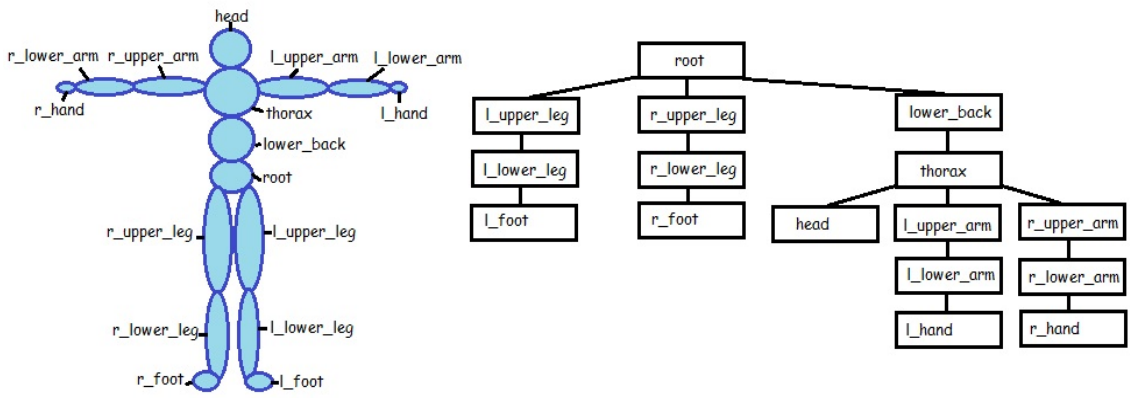

Figure 2.1. Character hierarchy

joint angles. This method is called Forward Kinematics (FK). The second method to animate the character is to manipulate the position of the end-effectors. The latter is called Inverse Kinematics (IK). Compared to IK, FK is straight forward. Each joint configuration can be represented as 4x4 transformation matrix specifying the position and orientation of the joint. Given a series of transformation matrices $M_1, \ldots, M_n$ which represent the configuration of a series of joints beginning at the root all the way to the end-effector, we can compute the transformation of the end-effector given as

$$M_e = M_1 M_2 \ldots M_n$$

IK on the other hand is more challenging. Given the target positions for the end-effectors, there may be more than one joint configuration that meet the constraint, or there may be none (e.g. when the target is too far away). Even if we manage to find a solution, some of the solutions may look unnatural. We discuss the problem of IK in the following subsection.

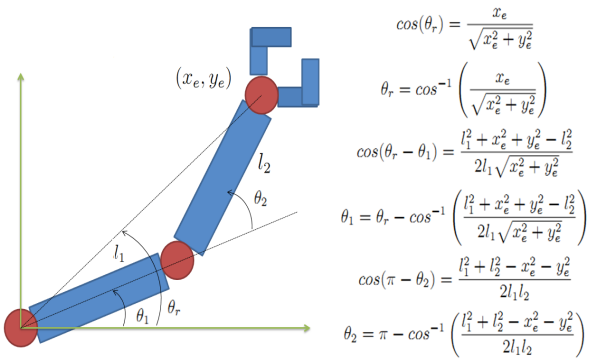

Figure 2.2. Analytical IK

2.1.1 Inverse Kinematics

Practically we can solve IK problem analytically. However, analytical method only works for fairly simple structures. For complex structures that involve many connected rigid bodies it is almost impossible to solve IK analytically. Figure 2.2 shows a simple example of analytical IK to solve for the joint configuration of a simple 2D structure.

Another way is to solve IK numerically. Numerical method involves iterative techniques and optimization to solve IK. Cyclic Coordinate Descent (CCD) is an example of iterative method used to solve IK. CCD performs an iterative heuristic search for each joint angle, so at the end, the end-effector could reach a desired location in space. In every iteration, CCD minimizes the distance between the end-effector and the target position by adjusting each joint one at a time, starting from the last link working backwards.

Most recently, [2] introduce a fast iterative solver for inverse kinematics problem called Forward And Backward Reaching Inverse Kinematics (FABRIK). FABRIK works by assuming points $p_1, \ldots, p_n$ as the positions of joints connecting multiple

rigid bodies, starting from the root p_1 all the way to the end-effector p_n. Given a new target position for the end-effector p'_n, in the first stage, the algorithm estimates each joint position starting from the end-effector, moving inwards to the manipulator base. To do this, the algorithm finds a line which connects p'_i to p_{i-1}. p'_{i-1} is the point in the line having distance d, where d is the distance between p_i and p_{i-1}. In the second stage, the algorithm continues to work in the reverse direction starting from the manipulator base, all the way to the end-effector, with the initial position of the manipulator base as the target. This is to make sure that the manipulator base does not change position. The process is repeated for a number of iterations until the target is reached.

[5] did a survey on inverse kinematics using numerical method, more specifically the jacobian transpose, pseudoinverse, and damped least squares methods. We define the positions for the end-effectors as a column vector $\vec{s} = (s_1, s_2, \ldots, s_k)^T$ where s_i is the position for the i_{th} end-effector. The target position is defined in the same manner as a column vector $\vec{t} = (t_1, t_2, \ldots, t_k)^T$, and the joint angles are written as a column vector $\vec{\theta} = (\theta_1, \ldots, \theta_n)^T$, where θ_j is the joint angle of the j_{th} joint. The desired change of the end-effectors is then $\vec{e} = \vec{t} - \vec{s}$. We can approximate the function that maps $\vec{\theta}$ to the position of the end effectors using the jacobian matrix given as

$$J(\vec{\theta}) = \left(\frac{\partial s_i}{\partial \theta_j}\right)_{i,j}$$

Thus, the desired change of the end-effector can also be written as

$$\vec{e} = J\Delta\vec{\theta} \tag{2.1}$$

To move the end effectors to the target location is the same as adding $\Delta\vec{\theta}$ to the current joint angles. $\Delta\vec{\theta}$ can be computed by solving equation 2.1 using jacobian transpose, pseudoinverse, or damped least square methods (see [5]).

It is also possible to combine analytical and numerical method to solve IK. Researchers often call this as hybrid method. Hybrid method uses different methods to solve IK for different parts of the body. Inverse Kinematics using ANalytical Methods (IKAN) is an example of using hybrid method to solve IK. AN in ANalytical here stands for Analytical and Numerical. The complete work is explained in detail in [40].

Other than analytical and numerical method, it is also common to solve IK using data-driven approach. Work on data-driven IK will be covered in the following section.

2.2 DATA-DRIVEN CHARACTER ANIMATION

With the advance of machine learning, the use of data-driven technique becomes another interesting subfield in character animation. In the following subsections, we discuss previous work on data-driven character animation. We begin by describing motion capture as a way to acquire motion data, and examine deeper on machine learning and its application in character animation in subsequent section.

2.2.1 Motion Capture

Motion capture is a technology used in data-driven character animation, which allows the recording of motion data from real-life subject. There are more than one way to do motion capture. One method is to equip the subject with exo-skeleton that moves with the actor. Inertial Measurement Unit (IMU) can also be used to capture motions by attaching them to the subject's body. Another common technique is to use the advance of optical cameras. Often the technique involves attaching multiple retro-reflective markers onto a subject and tracking the positions of these markers with special cameras that produce near infra red light which is reflected back by the markers to the cameras.

The product of motion capture is motion data. From this data, we can infer the position and orientation of the root link of the captured multibody system along with the joint angles. The data consists of the configuration for the multibody for the entire frames and allows a simple playback of the captured motion using a specially programmed player.

Motion capture has been heavily used to get realistic character animation. However, to actually set up a motion capture environment and capture lots of motion data is an expensive and time-consuming process. Another interesting subject for research in character animation is to design methods capable of utilizing the data effectively and efficiently, and also to help with the capture process.

2.2.2 Applications of Data-Driven Animation

Applications of data-driven animation often utilizes machine learning algorithms to extract latent information contained within the motion data. Example of such approach is shown by [49] in which the algorithm learns a nonlinear dynamic model from a few capture sequences. The learned model is used in the synthesis phase to produce new sequences of motions responsive to interactive user perturbations. The proposed method only requires a small number of examples and is able to generate a variety of realistic responses to perturbations that are not presented in the training data initially. A similar approach to learn a statistical dynamic model from motion capture data is done by [7], where motion priors are used to generate natural human motions that matches user's constraints.

The use of machine learning technique can also benefit the motion capture process. [9] develop a framework which enables users to shift the focus of the motion capture process to poorly performed tasks. The algorithm uses a machine learning approach called reinforcement learning to refine a kinematic controller that is assessed on every iteration in which a new motion is acquired by the system. The process is repeated until the controller is capable of performing any of the tasks needed by the users. With this framework, users can use the assessment of the kinematic controller as guidelines to determine which of the motions are still required by the system.

Another application of data-driven animation is motion graph. Motion graph is a useful technique in character animation to be used alongside motion capture.

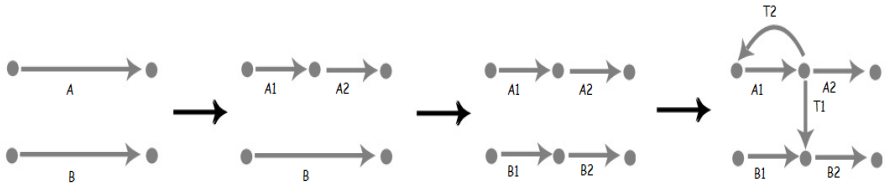

Figure 2.3. A motion graph built from two initial clips A and B. Clip A is cut into A1 and A2, and clip B is cut into B1 and B2. Transition clips T1 and T2 are inserted to connect the segments. Similar to figure 2 in [21].

Motion graph encodes motion capture data as directed graph. With motion graph, it is possible to synthesize motions just by a simple graph walk. There exists various implementation of motion graph. [27] implement motion graph as connected Linear Dynamics System (LDS). [21] constructs a directed graph where edges on the graph represent pieces of motion, and nodes serve as points where these pieces of motion join seamlessly. Their motion graph works by initially placing all the motion clips in the database as disconnected arcs in the graph. A greater connectivity for the graph is then fulfilled by connecting multiple clips, or by inserting a node into a clip and branching it to another clip (see Figure 2.3). In contrast, [1] uses nodes to represent motion sequences and edges to connect them. [25] model the motion data as first order markov process.

Since its publication, many improvements have been made for motion graph. [19] introduce parametric motion graphs which has similar idea but works with parameterized motion spaces. [4] also use the method to construct graph that connects clusters of similar motions termed motion-motif graphs. [53] extend the graph reconstruction step to achieve better connectivity and smoother motion by inter-

polating the initial motion clips before searching for candidate transitions. [35] use optimization-based method to construct motion graphs which combines the power of continuous constrained optimization to compute complex non-existent motions with the power of discrete optimization used in standard motion graph to synthesize long motions.

Data-driven IK is another major application of data-driven character animation. Data-driven IK uses optimization technique to search the motion database for a pose that match the target positions of end-effectors given as input to the system. [43] show an example of using data-driven IK to pose a character subjected to user constraints using a database of millions sample poses. The algorithm treats pose reconstruction problem as energy minimization and selects a pose from the database that satisfies the given user constraint.

2.3 PHYSICS-BASED CHARACTER ANIMATION AND CONTROL

Until now, we have only discussed methods for doing character animation that do not consider physical properties. Using physics engines, we can have an approximate simulation of physical systems, such as rigid body dynamics, soft body dynamics, and fluid dynamics, to be used in creating character animation. Physics engines come as both commercial and free software. Some of the well-known physics engines include PhysX, ODE, Vortex, and Bullet. "In recent years, research on physics-based character animation has resulted in improvements in controllability, robustness, visual quality and usability" [15]. In this section, we review some of the work in physics-based character animation. We begin by discussing forward and

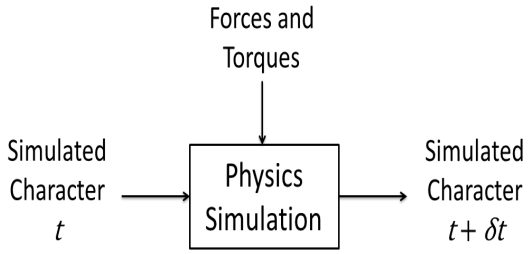

Figure 2.4. Physics Simulation

inverse dynamics which are used heavily in physics-based character animation, and move to the discussion about control theories in character animation later. In writing this section, we refer to the review by [15] on physics-based character animation.

2.3.1 Forward and Inverse Dynamics

Motions in physics-based character animation is a visualization of on-line physics simulation. A physics engine or physics simulator iteratively updates the state of the simulated character, based on its current state, and external forces and torques (see Figure 2.4). Physics simulator often consists of three components, collision detection which determines intersection and computes information on how to prevent it, forward dynamics which computes linear and angular accelerations of the simulated objects, and a numerical integrator which updates the positions, rotations, and velocities of objects, based on the accelerations (see [15]).

In forward dynamics, the state of a rigid body consists of its position, orientation, and linear and angular velocities. The goal is to solve the equation of motions given by equation 2.2 for $\ddot{q}$, where q is the vector of generalized Degrees-Of-Freedom(DOFs) of the system, $\dot{q}$ and $\ddot{q}$ are velocity and acceleration of these

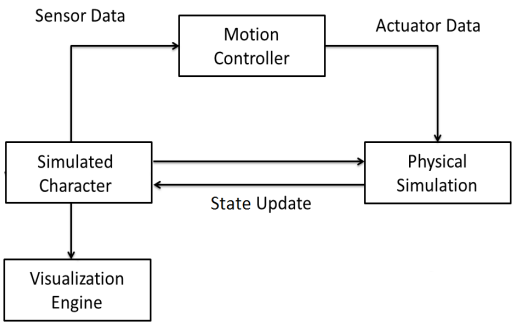

Figure 2.5. Physics-based character animation system with integrated controller.

generalized DOFs. $M(q)$ is a pose-dependent matrix describing mass distribution. τ is the vector of moments and forces acting on the generalized DOFs. The vector $c(q, \dot{q})$ represents internal centrifugal and Coriolis forces. The vector $e(q)$ represents external forces and torques, caused by gravity or external contact. The matrix $T(q)$ is a coefficient matrix, whose form depends on $M(q)$.

$$M(q)\ddot{q} + c(q, \dot{q}) + T(q)\tau + e(q) = 0 \tag{2.2}$$

Using the same equation, inverse dynamics computes the torques and forces required for a character to perform a specific motion by solving equation 2.2 for τ. Inverse dynamics are often used in motion control, to find the torques required to achieve a desired acceleration. Both forward and inverse dynamics are useful in creating physically realistic character animation for actuated systems.

2.3.2 Physics-Based Controller Modeling

Work on physics-based controller involves designing controllers for various different tasks such as stepping [45], balancing [30], contact-rich motion [29], and

locomotion [31]. The character models used for the tasks also varies. Most of them model characters after humans, such as the work by [20] who makes a controller for human athletic animation. Early work by [34] models character after animals. Some characters may not be modeled after any creature existing in nature [36]. Nevertheless, these controllers shares some common features. They act as components which feed actuator data needed to produce the motions of the characters, as torques and forces into a physics simulator.

Depicted in Figure 2.5 is a general architecture of a physics-based character animation system with integrated controller. Motion controllers use sensor data retrieved from the simulated character as feedback to adapt to the current state and compute actuator data which is passed to the simulator. Commonly used sensor data includes joint state, global orientation, contact information, Center Of Mass (COM), Center Of Pressure (COP), angular momentum [30], Zero-Moment Point (ZMP), and target position. The simulator then does the job of updating the state of the character which is visualized using graphics engine to produce animation.

Within the motion controller is the character model, its actuation model, and the control algorithm itself. The character model consists of the collision shapes and how they are connected using joints, and its dynamic properties (e.g. mass, inertia). The actuation model defines how to actuate the character. [15] classified four different ways to actuate physics-based characters. Muscle-based actuation occurs through muscles, which are attached to bones through tendons. Servo-based actuation, which is the most commonly used actuation model assumes a servo motor

in each joint, directly controlling the character. Virtual forces method uses jacobian (see Section 2.1.1) to compute the joint torques that imitate the effect of applying a virtual force at some point on the body [10]. Lastly, in character animation, it is also possible to apply forces or torques on unactuated body, such as global translation and rotation. These forces, often termed as hand-of-God, are only possible in animation since we are free from the restriction of using only internal forces to actuate the character. Aside of the body and actuation model, it is also useful to model additional properties into the character such as in [34] who models the padding material some creature have under their feet using non-linear springs.

Inverted Pendulum Model

Some of the main focus of research in physics-based character control is on biped locomotion tasks such as walking and running. Another commonly used model in locomotion controller is the Inverted Pendulum Model (IPM). An inverted pendulum is a pendulum which has its mass above its pivot point. The mass is connected to the pivot using a rigid stick. Unlike a normal pendulum, an inverted pendulum is inherently unstable, and must be actively balanced in order to remain upright, either by applying a torque at the pivot point or by moving the pivot point horizontally as part of a feedback system.

[39] uses the inverted pendulum model to analyze the relation between the ground reaction force and the center of mass trajectory for human walking gait. The inverted pendulum is used in both the single support phase and double support phase, where the acceleration of the center of mass is derived using the the informa-

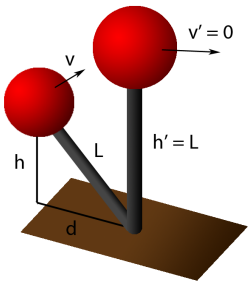

Figure 2.6. Inverted pendulum model for foot placement

tion of its height, projection to the ground, gravity, and the ground reaction force acting on the support. An analysis of the work done to the center of mass is also done in [22] using an inverted pendulum model.

Inverted pendulum model is also commonly used to compute foot placement for locomotion and balancing behavior. Figure 2.6 shows a simple way to determine foot placement for walking using inverted pendulum model. The sum of potential and kinetic energy is conserved for the inverted pendulum, i.e. $\frac{1}{2}mv^2 + mgh = \frac{1}{2}mv'^2 + mgh'$. To reach zero velocity on the next step, it is required for $v' = 0$ and $h' = L = \sqrt{d^2 + h^2}$, and thus $d = v\sqrt{h/g + v^2/4g^2}$. Taking a shorter step will achieve a positive velocity while taking a larger step will achieve a negative velocity, i.e., walking backwards. This insight is used to infer information of where to step in [51] and [10].

An extended version of the inverted pendulum called Inverted Pendulum on Cart (IPC) model is also used in [23] to model the trajectories of the center of mass of human running motion.

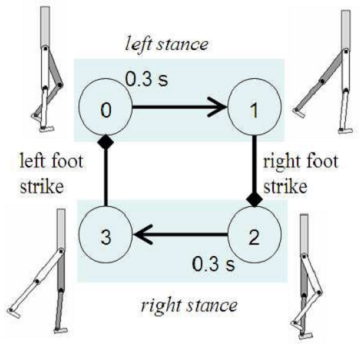

Figure 2.7. Finite state machine for walking from Figure 2 in [51]. Permission to use figure by KangKang Yin.

Finite State Machine

Tasks in motion controller are often modeled as phases and events. Common phases for walking controller are single stance and double stance phases. For running, phases are often divided into flight and contact phases. Common events in human locomotion are heel-strike and toe-off. A pose-control graph or Finite State Machine (FSM) can be used to define these phases. Each state in the FSM represents a phase. The controller goes from one state to another, doing different task on each state such as tracking different key poses associated with every state. The controller usually stays in a certain state for a specific duration, or until some events occurs (e.g. transition from double stance phase to single stance phase is triggered by toe-off event).

Figure 2.7 shows a simple FSM used for walking taken from [51]. The FSM defines target poses for the controller to track on each different state. Transitions between states happen after an elapsed time, or after foot contact. [20] use six states (flight, loading, heel contact, heel and toe contact, toe contact and unloading) to

model a running task, while [34] use five states (flight, loading, compression, thrust, and unloading). [24] also use FSM to model jumping and running tasks.

2.3.3 Control Algorithms

Besides the model for the character, within the motion controller, the control algorithm also needs to be designed carefully. Through the years, massive work on controller design has been done with little attempts on how to organize them. [15] might be the first to organize approaches of designing control algorithms into four different categories. Joint-space motion control works by minimizing the error between current and desired state using local feedback control to achieve specified kinematic trajectories for each actuated joint. Stimulus-response network control uses optimization to find optimal control parameters based on high level fitness function. Optimization-based motion control optimized for actuator values on-line in each iteration. Lastly, meta control combines existing motion controllers. They also evaluate each approach based on five criteria (skills repertoire, robustness, style and naturalness, user control, and usability).

Proportional Derivative Controller

The simplest control algorithm is to minimize the difference between the current and desired state using feedback control. Proportional derivative controller is the most commonly used feedback control. It tracks a desired setpoint by adjusting the output according to the measured error and its derivative. Equation 2.3 shows the

generalized form of a PD-controller

$$u(t) = K_p e(t) + K_d \frac{d}{dt} e(t) \tag{2.3}$$

where $u(t)$ is the output for the iteration at time t, $e(t)$ is the error measured at time t, and K_p and K_d are proportional and derivative gain.

Proportional derivative controller has been used in character animation, often to generate torque to track desired joint angle. The previous equation is modified as stated in equation 2.4, where τ, θ, θ_d, $\dot{\theta}$, and $\dot{\theta}_d$ are the torques, current joint angle, desired joint angle, the current rate of change of the joint angle, and the desired rate of change of the joint angle. Often the desired rate of change is specified to be zero ($\dot{\theta}_d = 0$) to mimic a spring damper system [34]. The two gain parameters K_p and K_d need to be tuned carefully to produce a robust controller, often using optimization. The desired joint angle can be specified manually with key poses [11], using abstract model that mimics certain biological or physics system [20] [38], a pose-control graph [51] [10], or reference motion capture data [30] [29]. A method to track multiple trajectories is also introduced in [32].

$$\tau = K_p(\theta - \theta_d) + K_d(\dot{\theta} - \dot{\theta}_d) \tag{2.4}$$

2.3.4 Controller Optimization

It is often the case that a controller is optimized to work robustly for a specific environment. Optimization within controller can happen in two stages. Off-line op-

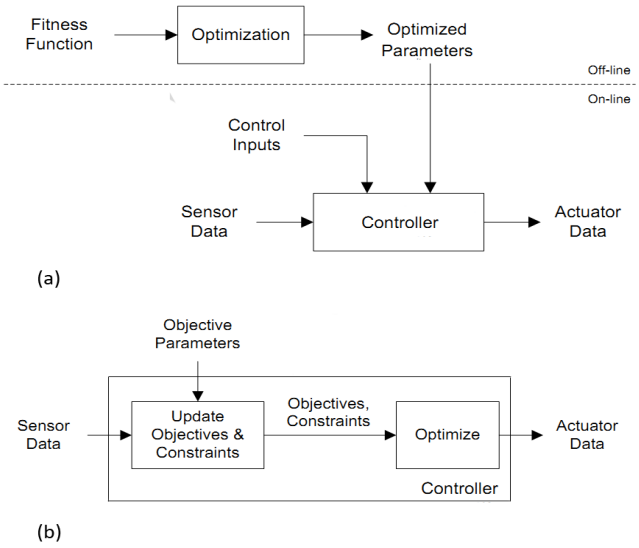

Figure 2.8. Two different optimization framework. (a) off-line optimization to find the optimal control parameters. (b) on-line optimization to find the actual actuator data. Similar to figure 11 and 13 from [15].

timization is used to find the optimal control parameters based on fitness functions, while on-line optimization is used to solve constraints modeled after the dynamics of the character and its environment, to compute the optimal set of actuator values. The difference between the two approaches is made clear in Figure 2.8.

The control parameters mentioned in the first approach varies from one work to another. [41] optimize the walking controller introduced in [51]. The control parameters used in their work are the character's starting pose which involves six global DOFs, 30 joint DOFs, and their generalized velocities, and the controller's K_p, K_d, and target angles. The optimization is done by evaluating simulations of ten seconds duration using objective function which considers the average forward speed and step length, deviations in y and z direction, symmetric timing for left and right state, angular momentum of the body, and power efficiency. [42] serve as a follow

up work on optimizing walking controllers for uncertain inputs and environments.

[50] show another example of parameters optimization. In their works, the parameters include a subset of joint target angles for one or more states of the FSM used in the controller. The states dwell time is also considered as another control parameter. The optimization is done using continuation method, where initially known control parameters that have been proven to work well for a certain task is adapted gradually to be able to perform more challenging tasks. In their work, a regular cyclic walk have been successfully adapted to climb a 65cm step, step over a 55cm sill, pushing heavy furniture, walking up steep inclines, and walking on ice.

The latter approach for controller optimization uses a constraint solver to find the optimal actuator data by solving user specified constraints which model the dynamics of the character and the environment. Since the optimization is done online, this approach has the advantage of capable of adapting to various disturbances during the simulation, with the price that the designer should be familiar with the dynamics formulation beforehand. Some example of this approach is by [30] who optimize the joint acceleration values which are fed into inverse dynamics to produce the torques needed to maintain balance on a character while still enabling the character to follow a motion reference. They use objectives that control the angular momentum which also used in [24] along with minimum torque objective, setpoint, and target objective for balancing, jumping, and walking animation.

Additionally, a look-ahead policy is required when the tasks are too complicated. Such approach, often termed as preview control, is commonly used in planned

locomotion or tasks which require the character to navigate through virtual environments with obstacles and uneven terrains. Example of such tasks can be found in [46], [48], [31], [26]. Often, the complexity of the dynamics constraints makes it impossible to plan far ahead in time. Low-dimensional abstract models can be used instead to approximate the high-dimensional model. [31] is an example of using abstract models of a Spring Load Inverted Pendulum (SLIP) and projectile motion to construct an optimal plan for walking, running, jumping, uneven terrain navigation, and external disturbances and projectile avoidance.

Lastly, it is also possible to design a controller by concatenating a series of controllers. [15] call this meta-control. The work of [26] serves as an example for this approach. The planner selects a series of controller and their corresponding parameters and duration to navigate a simulated character through a virtual environment.

2.4 INTERACTIVE CHARACTER ANIMATION INTERFACE

A growing subfield in computer animation which utilizes all the techniques we have covered so far is interactive character animation interface. Interactive character animation interface is associated with the design of a system which allows users to interact with it to drive virtual characters and produce character animation. To better help understand the problem, Figure 2.9 shows the entirety of an interactive character animation interface system. An interactive character animation interface system consists of three components: the interface itself, used as a front end for users to interact with the system; the output or the back end of the system, often an ani-

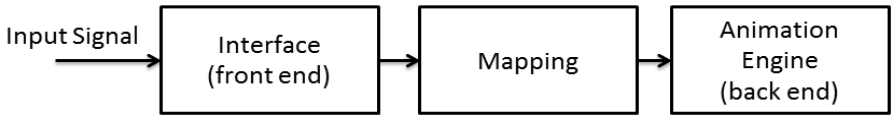

Figure 2.9. Components of animation interface.

mation engine; and the mapping between the two. Thus the problem of interactive character animation interface involves the design of these three components.

We begin by discussing several points of concern in the design of the interface component. The issues regarding the design of the interface encompass the type of the input signal and how it is retrieved. The available technologies nowadays have come up with numerous options for animation interface. One of the common trend comes from the computer vision community through the use of specialized camera capable of capturing depth image, often termed as depth camera. Depth images or depth map can be used as the standard input signal to an animation system [14]. Using 3D-depth map, it is possible to approximate the motion of users which can be used later to drive a virtual character. Microsoft's Kinect [1] is an example of depth camera which is freely available in the market for gamer and researchers. Initially intended as a device for gaming interface, Kinect is considered a breakthrough in computer vision communities due to its capability to capture color and depth image at low cost. Kinect technology has been used in several interactive animation interface research, for example, the work done by [3] on full body pose reconstruction using depth camera. Other available options for animation interface which do not utilize vision-based approach include the use of force plates which is

[1] http://www.microsoft.com/en-us/kinectforwindows/

covered in detail in [52] and [18], accelerometer sensors (Nintendo Wiimote [2]) used in [47], touch sensor [13], or the more bizarre neuro-signal acquisition device (e.g. emotiv [3]).

A problem to be considered in choosing the interface is the quality of the input signal. Input signal capture using different devices differs in quality. However, in most cases, this input signal is noisy. For example, an interface which captures voice signal will unintentionally record background noises. This problem is similar in all types of input signal, although they may differ in quantity and severeness. Another problem that may arise is the low dimensionality of the input signal. The input signal that comes from the use of pressure plate for example only contains information regarding the pressure caused by an object put on it. The issue of how this low dimensional information can then be used to drive a full body character needs to be considered.

The second component of an interactive character animation interface system is the mapping component. The task of the mapping component of the system is to take the input signal retrieved by the interface component and transform it into another form more suited to drive a virtual character, commonly the character's joints configuration. The problem is more challenging because the character may involve a large number joints which makes it highly dimensional. Data-driven technique plays a significant role here. The mapping problem can be reduced to a mapping from the low dimensional input signal to a high dimensional output signal. The work by

[2]http://www.nintendo.com/wii/
[3]http://www.emotiv.com/

[6] may serve as an example. The interface of their system consists of two standard cameras for capturing the positions of a small number of retro-reflective markers treated as the input signals to drive a skeleton. The mapping is done by initially searching the motion capture database for examples that are close to the input signals using fast K-nearest Neighbor and further building an online local linear model from these examples. The pose (output signal) is then reconstructed using the local linear model and the input signal. A parallel work by [28] analyze for input signals which capture the most information about the motion (e.g. the walking motion can be reduced to the motion of the ankles and wrists) that can be used as reference to decide which of the input signals are sufficient to drive the character.

Mapping using data-driven approach often requires training data that does not only include motion capture data. Other data such as the foot pressure distribution image which corresponds to the type of interface in used is also needed. This data which mainly consists of sample of input signals is essential to build the mapping model between the input signal that is interface specific and the output signal or the motion data. Often, the interface specific data is collected and synchronized in parallel with the motion capture data [52] [18] [47].

Lastly, various options are also available to decide for the back end of the system. Although the end product of the system will most likely be animation, as discussed in previous section, there are multiple methods to construct a system that produce animation. The system's back end can be as simple as only incorporating kinematics character animation, or it can be more advance with integrated physics

simulator. Other than that, the influence from the robotics community also opens up possibilities to extend the concept of virtual character control to controlling real humanoid robots.

CHAPTER 3

CASE STUDY ON PHYSICS-BASED AVATAR CONTROL

In the previous section, we have looked at some approaches in character animation. We have also covered interesting topics in interactive character animation interface. In this section we summarize our preliminary work, which was published at Motion In Games 2011, on deploying the generalized locomotion controller in cartwheel-3d [10] to multiple simulation platforms (e.g. PhysX, Bullet, and Vortex). We start with the author's original code release that can be downloaded from their website (http://code.google.com/p/cartwheel-3d/). Cartwheel-3d utilizes some of the physics-based character animation techniques we discussed earlier to build a controller for walking avatars. The goal of our project is to push active character control into game, and also to emphasize a robust design on physics-based animation engine for avatar control. We overview the main characteristics of the physics engines we used and illustrate the major steps of integrating active character controllers with physics SDKs, together with necessary implementation details. We also evaluate and compare the performance of the locomotion control on different simulation platforms including the one used in the original work (ODE). We refer interested readers to [17] for the complete work. The code is released online to encourage more follow-up works, as well as more interactions between the research community and the game development community.

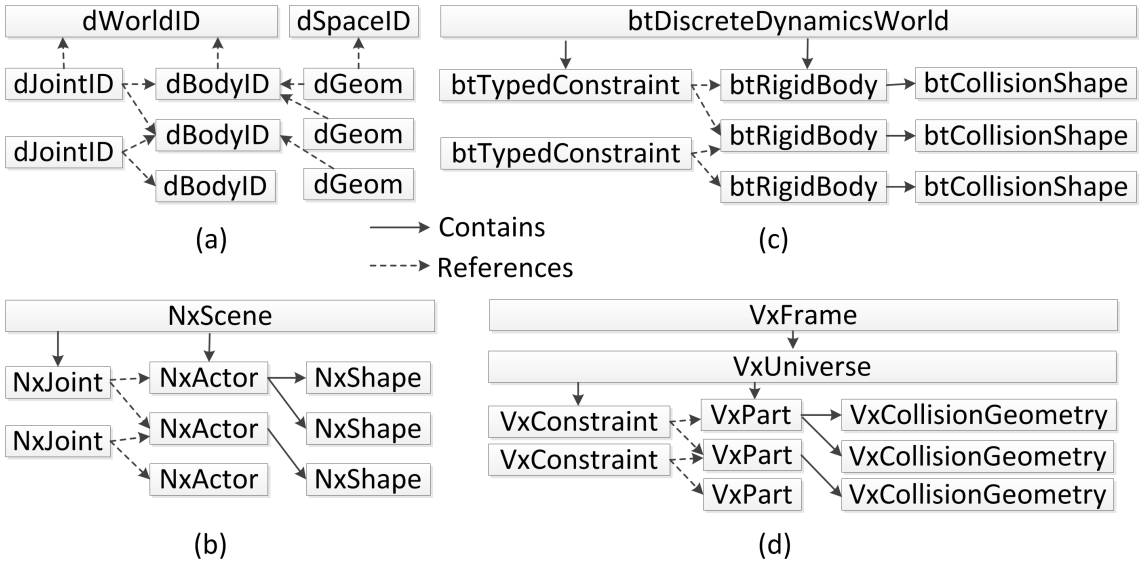

Figure 3.1. Partial architecture diagrams relevant to active rigid character control and simulation of four physics SDKs: (a) ODE (b) PhysX (c) Bullet (d) Vortex.

3.1 SIMULATION PLATFORMS OVERVIEW

We port a generalized locomotion controller to multiple physics engines. The original cartwheel-3d is a built on top of an open-source dynamics engine ODE (Open Dynamics Engine), which is commonly used in the animation research community for its superior constraint accuracy. ODE's rigid body dynamics and collisions library is mostly written in C++, and provides both C and C++ interface [37]. The latest ODE version 0.11.1 was released in October 2009. Currently, ODE's development seems to be suspended.

We choose two other simulation engines that are more popular in the game development community: PhysX and Bullet. PhysX is supported by NVIDIA and currently dominates the market share among the released game titles. Bullet is open source and has great overall performance. Both libraries are written in C++

Pseudocode 1 : Character control and simulation pipeline

```
1: Create a simulation world;
2: Create objects;
3: Create joints;
4: Loop:
5:     Apply control and accumulate forces;
6:     Detect contact and collisions;
7:     Timestep;
```

and support rigid body and soft body dynamics, collision detection, and vehicle and character controllers, with the addition of fluid dynamics being supported in PhysX as well [33] [12].

To satisfy the demand of biomechanics and robotics communities, who are often skeptical of simulation results mentioned in the graphics literature achieved by the use of physics simulators that are also associated with games, we choose the fourth engine Vortex. Vortex is currently the leading dynamics platform in the mechanical engineering and robotics industries. Its simulation tools have been widely used in virtual prototyping and testing, virtual training, mission rehearsal, as well as in serious games. Vortex is written in C++ and supports rigid body dynamics, collision detection, fluids, particles, cables and vehicles [8]. Figure 3.1 shows partial architecture diagrams of the four engines.

3.2 CHARACTER CONTROL AND SIMULATION PIPELINE

Our simulation pipeline for active rigid character control is listed in Pseudocode 1. Each step are mapped to SDK-specific code. Line 5 of Pseudocode 1 is where the motion controller (see Figure 2.5 of Section 2.3.2) computes active control torques and accumulates external forces for each rigid link of the character which

are passed to the physics engine. We refer interested readers to the paper [10] and our source code for details of the control algorithm. Line 7 of Pseudocode 1 steps the dynamics system forward in time for a small time step: constrained equations of motion are solved and kinematic quantities are integrated. This step is equivalent to the update step in Figure 2.5 from Section 2.3.2. Here the major difference of the four engines is whether they separate collision detection and timestepping into two parts. ODE separates them into two distinct steps as shown in Line 6 and 7; while PhysX, Bullet, and Vortex integrate collision detection into their timestepping so Line 6 should really be merged into Line 7 for these three platforms.

Different engines use different methods for their forward dynamics (see Section 2.3.1) simulation. ODE provides two timestepping functions: *dWorldStep* and *dWorldQuickStep*. The first one uses the Dantzig algorithm to solve the LCP (Linear Complementarity Problem) formulation of the constrained equations of motion, while *dWorldQuickStep* uses the iterative PGS (Projected Gauss Seidel) method which sacrifices accuracy for speed. Both methods use a fixed user-defined time step. We use ODEquick to refer to simulations on ODE using the iterative solver. The PGS method is also used in PhysX and Bullet, and the users can specify the maximum number of iterations allowed. Vortex provides both a standard and an iterative LCP solver, just like ODE. However, we only test its standard solver *kConstraintSolverStd* in our work. PhysX, Bullet, and Vortex can advance fixed time steps as well as variable time steps during timestepping. For ease of comparison, we use fixed time steps for all four engines in all our experiments.

	ODE	PhysX	Bullet	Vortex
simulation world	*world* *space*	*scene*	*world*	*frame*
timestepping function	dSpaceCollide(...) dWorldStep(...)	->simulate(...)	->stepSimulation(...)	->step()

Table 3.1. Different concepts of simulation world within multiple simulation engines. The decoupling of the simulation world into a dynamics world and a collision space in ODE leads to the use of different timestepping functions for each world.

3.3 IMPLEMENTATION DETAILS

We summarize the commonalities and differences between ODE, PhysX, Bullet, and Vortex in supporting character control during the implementation phase which consists of managing the simulation world; rigid bodies and joints instantiation; and contacts and collisions handling.

Table 3.1 shows how the four different engines differ in their concept of the simulation *world*. The simulation *world* is the virtual counterpart of the physical world that hosts the virtual objects and constraints governed by physics laws in dynamics engines. ODE decouples the concept of the simulation world into a dynamics world which handles rigid body dynamics and a collision space which handles collision detection. In PhysX, Bullet, and Vortex, however, one world handles both dynamics and collision. Also depicted in Table 3.1 is the timestepping function used by each engine.

Different simulation engines support different types of objects. The object within the simulation world can be classified into three categories: body which carries the kinematic and dynamic properties including position, velocity, and mass and inertia; shape which specifies the geometric properties, or shape, of an object;

	body (dynamic object)	shape (geometric object)	static shape (static object)
ODE	*body*	*geom*	*geom* (not attached to a body)
PhysX	*actor*	*shape*	*actor* (with no dynamic properties)
Bullet	*rigidBody*	*collisionShape*	*collisionObject*
Vortex	*part*	*collisionGeometry*	*part* (frozen by users after creation)

Table 3.2. Dynamic, geometric, and static objects have different names in different SDKs. We refer them as bodies, shapes, and static shapes in our discussion.

#DoFs	0	1R	2R	3R	1T	1R1T	6
ODE	fixed	hinge	universal	ball-and-socket	slider	piston	
PhysX	fixed	revolute		spherical	prismatic	cylindrical	freedom
Bullet		hinge	universal	point2point	slider		freedom
Vortex	RPRO	hinge	universal	ball-and-socket	prismatic	cylindrical	

Table 3.3. Common joint types supported by the four simulation engines. We classify the types of joints by counting how many rotational and translational DoFs a joint permits. For example, 1R1T means 1 rotational DoF and 1 translational DoF.

static shape which represents static object such as ground which usually do not move and are only needed for collision detection. We use these objects to construct our simulated characters often consist of multiple rigid bodies connected with joints (see Section 2.1). Table 3.2 lists these types of objects implemented under different names in different physics SDKs.

Similar to objects, different physics engines implement different types of joint. Joints used to connect multiple body parts are classified by the number of rotational and translational DOFs they permit. Table 3.3 lists the common joint types supported by the four SDKs. Again the same type of joint may be named differently on different platforms.

Another subtle difference related to objects and joints is how they are instantiated. Instantiating objects means specifying their kinematic and dynamic properties, while instantiating joints means specifying joint parameters (e.g. the bodies

	object pairs	group pairs	bit mask	callback
ODE		✓	✓32-bit	must
PhysX	✓	✓	✓128-bit	optional
Bullet			✓32-bit	optional
Vortex	✓	✓		optional

Table 3.4. Collision filtering mechanisms.

they connect, joint limits). Different methods are provided by the four SDKs. In PhysX every object has a descriptor, which is used to specify all the arguments of an object before its creation. ODE, on the other hand, allows users to specify parameters for objects after their creation. Bullet adopts yet another approach by providing object-creating functions with long lists of arguments, while Vortex supplies two types of procedures, one with many arguments as in Bullet; and the other with less arguments during creation, giving users the flexibility to specify additional parameters later on as in ODE (see [17] for code snippets).

One last difference we observe is how each engine detects and processed collisions. PhysX, Bullet, and Vortex provide fully automatic collision detection and seamless integration with the dynamics solver. Users only need to define collision shapes, and all the collision constraints are generated implicitly and desired collision behavior will automatically computed. In ODE, however, users have to call the broad phase collision detection manually before timestepping, as shown in Table 3.1. In addition, users need to supply a callback function to further invoke the narrow phase collision detection. Lastly, users must explicitly create contact joints from detected collisions for the dynamics solver to take into account the collisions. Table 3.4 lists the collision filtering mechanisms provided by each physics engine.

walk style		inplace	normal	happy	sneak	chicken	drunken	jump	snake	wire
control parameter	desired speed	0.0	1.0	2.0	1.5	2.5	0.5	1.0	3.0	-1.0
	cycle duration	0.6	0.6	0.5	0.6	0.5	0.6	0.5	0.5	0.6
	step width	0.12	0.12	0.1	0.14	0.12	0.2	0.15	0.13	0.1
motion distance	ODEquick-ODE	2.3e-4	1.6e-4	2.8e-4	9.5e-4	1.0e-3	1.6e-3	7.9e-4	6.6e-4	9.1e-4
	PhysX-ODE	5.5e-2	1.9e-2	9.6e-2	8.3e-1	8.4e-2	5.6e-2	9.6e-2	7.3e-2	1.5e-1
	Bullet-ODE	1.7e-2	1.2e-2	1.5e-2	5.9e-2	1.3e-2	1.7e-2	2.9e-2	1.5e-2	1.4e-2
	Vortex-ODE	1.8e-3	3.5e-3	7.1e-3	2.7e-2	6.4e-3	1.9e-3	6.1e-3	1.1e-2	3.7e-3

Table 3.5. Motion deviation analysis. The simulated motion on ODE serves as the baseline. ODEquick uses the iterative LCP solver rather than the slower Dantzig algorithm. We investigate nine walking controllers: inplace walk, normal walk, happy walk, cartoony sneak, chicken walk, drunken walk, jump walk, snake walk, and wire walk.

For collision postprocessing, it is common to read back information from the physics engine, for example if the locomotion controller wishes to regulate the Ground Reaction Forces(GRFs) between the character and the ground, or to calculate GRF-based feedback controls, or simply to monitor or visualize the GRFs. ODE reads back the contact information from an array initialized on the specification of the collision callback. Bullet initializes a collision dispatcher during the creation of the simulation world, and from the dispatcher the contact information can be obtained for postprocessing. In PhysX, users subclass *NxUserContactReport* and register an instanced object of this class during the initialization of the simulation world. Then *onContactNotify(...)* of the object receives the contact information for each pair of shapes or actors that has requested contact notification through proper flag setting. Similar to PhysX, Vortex users can subclass *VxIntersectSubscriber* to access contact events before or after timestepping. However, if users just need to read back the contacts after the dynamics has stepped forward, a simpler way is to access *VxDynamicsContact* via a pointer of *VxUniverse*.

3.4 PERFORMANCE EVALUATION AND COMPARISON

We test nine controllers walking in various styles provided by the original cartwheel-3d online distribution. For common simulation parameters we use the cartwheel-3d defaults on all platforms, e.g., a ground friction of 0.8; a fixed time step of $0.5ms$ etc. Since each controller can walk the character successfully with a range of parameter settings, such as the desired walking speed, duration of one walk cycle, and width of the steps, we manually choose one point in the control parameter space as listed in table 3.5. Then we measure the distance between the motion simulated on each engine and that on ODE. That is, we use the motion simulated on ODE as the baseline. The distance $d(m, \tilde{m})$ between two simulated motions m and $\tilde{m}$ is defined as follows:

$$d(m, \tilde{m}) = \frac{\sum_{k=1}^{n} \sum_{i=1}^{l} ||\mathbf{p}_k^i - \tilde{\mathbf{p}}_k^i||}{nlh} \tag{3.1}$$

where n is the number of frames in $\tilde{m}$ and l is the number of links of the character. We record ten cycles of a simulated walk at $30Hz$ as m and $\tilde{m}$, starting from the fifth cycle when the walk has converged onto its limit cycle. Then m is time aligned with $\tilde{m}$ and resampled to n frames for comparison. $\mathbf{p}_k^i$ is the center of mass location of each limb in the character root frame. h is the height of the character for normalization.

Perceptually the limit cycles of the simulated walks from all nine controllers are quite similar, although there are cases the step length or width is noticeably different (demo video provided on the website). The difference of the beginning start-up

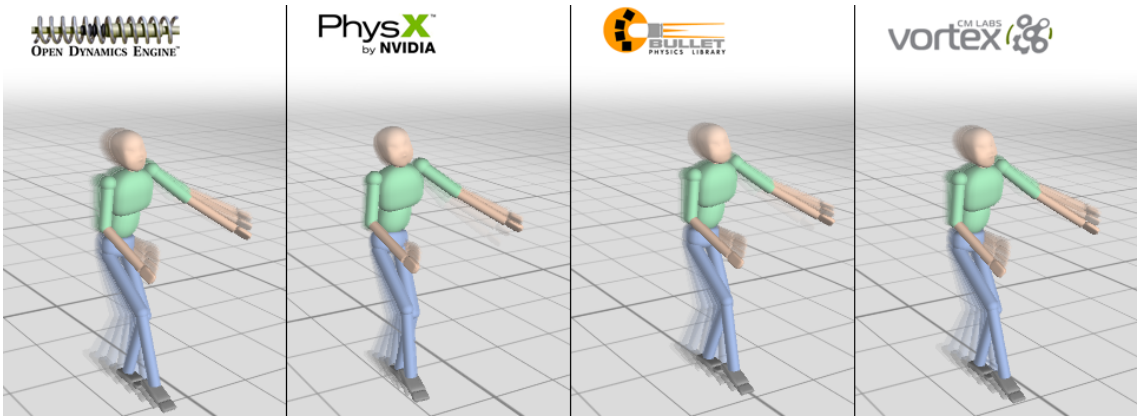

Figure 3.2. Screen captures at the same instants of time of the happy walk, simulated on top of four physics engines: ODE, PhysX, Bullet, and Vortex (from left to right).

cycles is also notable. In fact, due to the initial difference, some characters walk diagonally rather than on the default straight line. Quantitatively, the simulations on ODE and Vortex resemble more, and their averaged limb position difference never exceeds 3% of the character height. Usually the simulations from PhysX differ more. Figure 3.2 shows a side-by-side comparison of the same frames of the happy walk simulated on each platform.

We test the stability and robustness of the normal walk controller on each engine with respect to the size of the simulation time step. The original cartwheel-3d uses a default simulation time step of $0.5ms$ on top of ODE. We further search for three time steps as shown in table 3.6. bound1 is the largest time step before the simulation becomes unstable; bound2 is the largest time step before the simulated character falls; and bound3 is the largest time step before the distance between the simulated motion from the default motion simulated at the default time step becomes larger than 0.1. Note that in PhysX the character falls using a time step

Engine	bound1	bound2	bound3
ODE	18	1.6	1.0
ODEquick	18	1.4	1.0
PhysX	150	0.9	–
Bullet	100	1.3	1.0
Vortex	35	1.6	1.2

Table 3.6. Stability analysis with respect to the size of the time step in ms.

Engine	$d(m_{8/9}, m_{10/11})$	$d(m_{8/9}, m_{14/15})$
ODE	0.185	0.036
ODEquick	0.196	0.037
PhysX	0.172	0.006
Bullet	0.174	0.014
Vortex	0.157	0.002

Table 3.7. Stability analysis of the normal walk with respect to external push.

larger than $0.9ms$, when the motion distance is 0.06. We can see that simulations in PhysX and Bullet are much more stable at large time steps than in ODE and Vortex, but they do not differ much in terms of the stability of the walking controller once the simulation moves into the stable region.

We also test the robustness of the normal walk controller on each engine with respect to external perturbations. The character gets pushed by a planar force of (250,150)N backward and sideways for $200ms$ at the onset of the tenth cycle m_{10}. Using the motion from the eighth and ninth cycles, denoted as $m_{8/9}$, as the baseline motion, we then compare how much $m_{10/11}$ and $m_{14/15}$ differ from the baseline $m_{8/9}$. Table 3.7 shows that the motion error caused by the external push diminishes quickly, and the character eventually goes back to the original limit cycle. Note here we only measure the distance between the feet positions, as the controller uses a foot placement strategy to regain balance so the upper body postures do not differ

too much after the perturbation.

The computational cost for simulating 10 characters is roughly distributed as follows: 2% on rendering; 18% on control; 65% on simulation including collision detection and constraint solving (except ODEQuick which is faster). Table 3.8 shows the average timing of one simulation step using the default time step $0.5ms$, for one character, ten characters, and a hundred characters. We see a near linear degradation of the performance, probably mainly because our characters all walk independently. We also tried simulating multiple characters with GPU acceleration turned on in PhysX, but we did not observe any performance gain with our NVIDIA graphics card GeForce GTX 570. This is because rigid body collisions are still processed by the CPU in the PhysX version we use. Furthermore, we tested the multithreading capability of PhysX and Vortex. Unfortunately our tests with 6 threads on our 6-core machine did not show significant speedup either. This contradicts with released tests from PhysX and Vortex. In diagnosing this problem, we found that only two of our six cores are active no matter how many threads we specify for the engine. This may be caused by the Python interface or wrapper used in our software, or unknown issues in the interaction between the Windows thread scheduler and Python. Bullet also provides multithreading and GPU acceleration, which we have not tested due to lack of documentation.

Our work shows that given a well design control framework, it is plausible and straightforward to deploy it to another physics engine. Integrating the chosen locomotion controller into game physics today is highly possible. The porting part of

Engine	1 character	10 characters	100 characters	100 characters with 6 threads
ODE	0.099	0.96	9.9	×
ODE-quick	0.064	0.61	6.6	×
PhysX	0.300	1.57	14.2	11.6
Bullet	0.107	0.96	11.0	–
Vortex	0.120	1.60	17.4	12.4

Table 3.8. Average wall clock time of one simulation step in milliseconds using the default simulation time step $0.5ms$. Timing measured on a Dell Precision Workstation T5500 with Intel Xeon X5680 3.33GHz CPU (6 cores) and 8GB RAM. Multithreading tests with PhysX and Vortex may not be valid due to unknown issues with thread scheduling.

this project was completed within four weeks by a first year graduate student who had no experience in simulation and control but had basic knowledge on computer animation. We also show that without much parameter tuning, the original controllers can immediately walk the character successfully after porting, although in slightly different styles.

We emphasize that our results are specific to the type of controller being tested [10], and its specific implementation in cartwheel-3d. This implementation controls the walking style through explicit PD torques at every simulation time step. The advantages of such an implementation include: computing the control at each time step is cheap and easy; porting the controller to off-the-shelf physics engines is straightforward; and the simulated character exhibits natural compliance when pushed. The disadvantage is that the simulation has to take smaller time steps, compared to other methods which integrate the equations of motion directly into each control time step.

CHAPTER 4

CASE STUDY ON PERFORMANCE-BASED AVATAR CONTROL

In this section we show another application of animation techniques for avatar control. The work we discuss here was published at Motion in Games 2012. In this project we utilize avatar control to create a virtual try-on experience using a Kinect sensor and an HD camera. Our system is called EON Interactive Mirror. We elaborate on several key challenges such as calibration between the Kinect and HD cameras, and shoulder height estimation for individual subjects. Quality of these steps is the key to achieving seamless try-on experience for users. We also present performance comparison of our system implemented on top of two skeletal tracking SDKs: OpenNI and Kinect for Windows SDK (KWSDK). Lastly, we discuss our experience in deploying the system in retail stores and some potential future improvements. We refer interested readers to [16] for the complete work.

Since users feel more comfortable purchasing outfits after physically trying them out, the fashion industry greatly relies on traditional retail outlets. The consequences of this fact include that Internet shopping is hard for clothing; and fitting rooms in brick-and-mortar stores are always packed during peak hours. This motivates us in developing a virtual try-on system that enables shoppers to digitally try out clothes and accessaries. Our system, named EON Interactive Mirror(http://www.eonreality.com/), utilizes one Kinect sensor and one High-Definition (HD) camera. The Interactive Mirror enables the shopper to virtually try-on clothes,

dresses, handbags and accessories using gesture-based interaction. Customers experience an intuitive and fun way to mix-and-match collections without having to queue for fitting rooms or spend time changing items. Customers can also snap pictures of their current selections and share them on Social Media to get instant feedback from friends, which can potentially shorten the decision time for making the purchase.

Our system has been deployed since April 2012 in one of Singapore's largest shopping centers with approximately three million visitors passing through every month. With EON Interactive Mirror, walk-by customers can be convinced to walk into the store. Within the store, it has created unique customer experiences of virtually trying on the latest fashion 'on-the-go' in a fun and engaging way, and made the store stand out from the highly competitive market.

In order to achieve a believable virtual try-on experience for the end user, several challenges have to be addressed. First, the Kinect sensor can only provide low-resolution VGA quality video recording, yet high quality video is essential for attractive visual appearance on large screens. We thus opt to use an HD camera to replace the role of Kinect's built-in RGB camera. This necessitates a calibration process between the HD camera and the Kinect depth camera in order to map the 3D clothes seamlessly to the HD video recording of the customers. Second, digital clothes need to be resized to fit to a user's body. Yet the Kinect depth data is noisy, the skeletal motion tracked by third-party SDKs is not accurate, and sometimes the lower part of the body is not even in the camera's field of view. We thus need a

Figure 4.1. The front view of the Interactive Mirror with Kinect and HD camera placed on top.

reliable and robust procedure to estimate the shoulder height of the users for the clothes-body fitting process.

4.1 SYSTEM OVERVIEW

Our virtual try-on system consists of a vertical TV screen, a Microsoft Kinect sensor, an HD camera, and a desktop computer. Fig. 4.1 shows the front view of the Interactive Mirror together with the Kinect and HD camera. The Kinect sensor is an input device marketed by Microsoft, and intended as a gaming interface for Xbox 360 consoles and PCs. It consists of a depth camera, an RGB camera, and microphone arrays. Both the depth and the RGB camera have a horizontal viewing range of 57.5 degrees, and a vertical viewing range of 43.5 degrees. Kinect can also tilt up and down within -27 to +27 degrees. The range of the depth camera is [0.8∼4]m in the normal mode and [0.4∼3]m in the near mode. The HD camera supports a full resolution of 2080 × 1552, from which we crop out the standard HD resolution

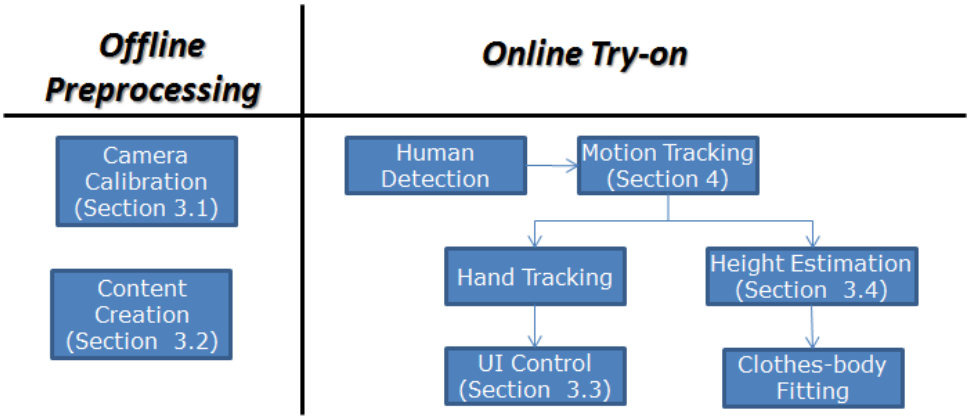

Figure 4.2. Major software components of the virtual try-on system.

1920 × 1080. It supports a frame rate of 60Hz with a USB 3.0 interface of up to 5 Gbit/s transfer rate. The Interactive Mirror is a 65" TV screen mounted in portrait mode with HD resolution 1920 × 1080. We recommend a space of [1.5∼2.5]m × [2.0∼2.5]m × [2.0∼2.5]m (width×legth×height) in front of the mirror as the virtual fitting room.

Fig. 4.2 illustrates the major software components of the virtual try-on system. During the offline preprocessing stage, we need to calibrate the Kinect and HD cameras, and create 3D clothes and accessories. These two components will be discussed in more details in Sections 4.1.1 and 4.1.2 respectively. During the online virtual try-on, we first detect the nearest person among the people in the area of interest. This person will then become the subject of interest to be tracked by the motion tracking component implemented on two publicly available Kinect SDKs, as will be discussed in Section 4.2. The user interacts with the Interactive Mirror with her right hand to control the User Interface (UI) and select clothing items. The UI layout will be discussed in more details in Section 4.1.3.

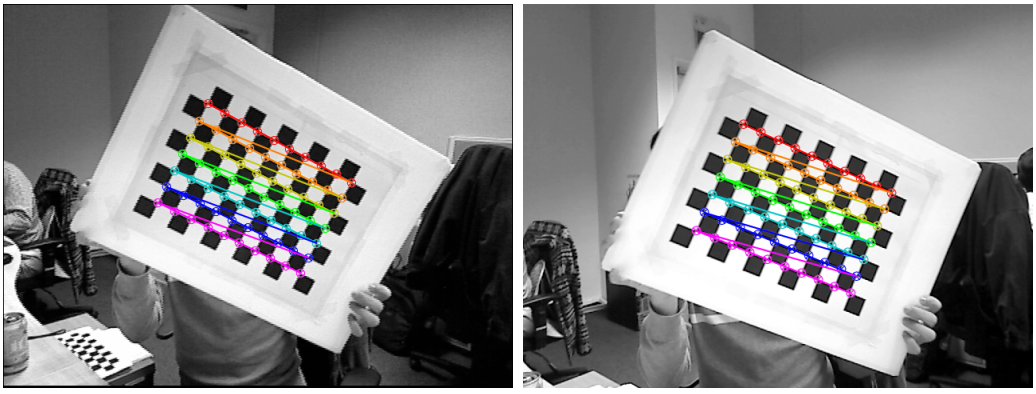

Figure 4.3. The camera calibration process. The checkerboard images seen by the Kinect RGB camera (left) and the HD camera (right) at the same instant of time.

For good fitting of the clothes onto the body, we need to estimate the height of the user to resize the digital clothes appropriately. We discuss two ways of height estimation in Section 4.1.4. The ratio between the height of the real user and that of the default digital model will then be used to scale the clothes uniformly in three dimensions. Finally, the resized digital clothes are skinned to the skeleton, rendered with proper camera settings, and merged with the video stream of the user.

4.1.1 Camera calibration

Vision-based augmented reality systems need to trace the transformation relationship between the camera and the tracking target in order to augment the target with virtual objects. In our virtual try-on system, precise calibration between the Kinect sensor and the HD camera is crucial in order to register and overlay virtual garments seamlessly onto the 2D HD video stream of the shoppers. Furthermore, we prefer a quick and semi-automatic calibration process because the layout between Kinect and HD camera with respect to the floor plan may be different for different

stores, or even for the same store at different times. To this end, we use the *CameraCalibrate* and *StereoCalibrate* modules in OpenCV [44] for camera calibration. More specifically, we recommend to collect a minimum of 30 pairs of checkerboard images seen at the same instant of time from Kinect and HD camera, and calculate each pair's correspondences, as shown in Fig. 4.3.

In addition, the Kinect sensor is usually not perfectly perpendicular to the ground plane, and its tilting angle is needed to estimate the height of users later in Section 4.1.4. We simply specify the floor area from the Kinect depth data manually, and the normal vector of the floor plane in Kinect's view can be calculated. The tilting angle of Kinect is then the angle between this calculated floor normal and the gravity normal.

Furthermore, to seamlessly overlay the virtual garments on top of the HD video, we also need to estimate the tilting angle of the HD camera, and a correct FoV (Field of View) that matches the TV screen's aspect ratio. Subsequently precise perspective transformations can be applied by our rendering engine to properly render the deformed digital clothes for accurate merging with the HD video.

To summarize, the output of the camera calibration procedure include extrinsic camera parameters (translation and rotation) of the HD camera with respect to the Kinect depth camera, the tilting angles of the Kinect sensor and the HD camera with respect to the horizontal ground plane, and FoV of the HD camera.

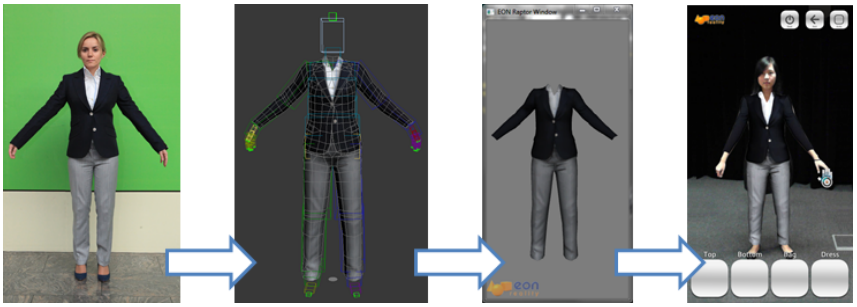

Figure 4.4. Major steps for content creation. Catalogue images are first manually modeled and textured offline in 3DS Max. We then augment the digital clothes with relevant size and skinning information. At runtime, 3D clothes are properly resized according to a user's height, skinned to the tracked skeleton, and then rendered with proper camera settings. Finally, the rendered clothes are merged with the HD recording of the user in realtime.

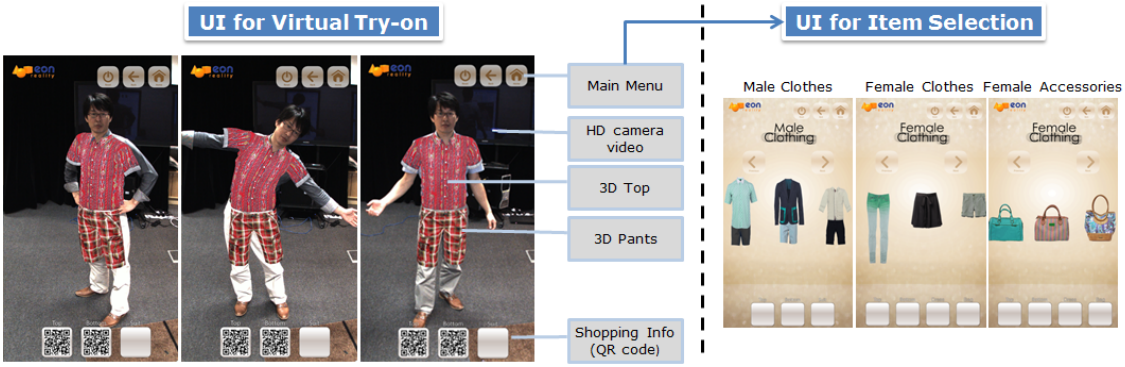

Figure 4.5. Left: the UI for virtual try-on. Right: the UI for clothing item selection.

4.1.2 Content creation

Our virtual 3D clothes are based on actual catalogue images, so that new fashion lines can be added to the system quickly. Fig. 4.4 shows the major steps of converting catalogue images to 3D digital clothes. In the preprocessing stage, our artists manually created one standard digital male mannequin and one female mannequin. Then they modeled the catalogue images into 3D clothes that fit the proportions of the default mannequins. Corresponding textures were also extracted

and applied to the digital clothes. Then we augment the digital clothes with relevant size and skinning information. At runtime, 3D clothes are properly resized according to a user's height, skinned to the tracked skeleton, and then rendered with proper camera settings. Lastly, the rendered clothes are merged with the HD recording of the user in realtime.

Our content development team modeled 115 clothing items in total, including male clothes, female clothes, and accessories. On average it took about two man days to create and test one item for its inclusion into the virtual try-on system.

4.1.3 User interface

Fig. 4.5 depicts the user interface of the Interactive Mirror. Because our clothes are 3D models rather than 2D images, users are able to turn their body within a reasonable range in front of the Interactive Mirror and still have the digital clothes properly fit to their body, just like what they can see in front of a real mirror. The user selects menu items and outfit items using hand gestures. Different tops, bottoms, and accessories can be mixed and matched on the fly.

4.1.4 Height estimation

Digital clothes need to be rescaled according to users' body size, for good fitting and try-on experiences. We propose two methods to estimate a user's shoulder height. The first one simply uses the neck to feet height difference, when both the neck and the feet joints are detected by Kinect skeletal tracking SDKs. As illustrated in Fig. 4.6, however, sometimes the feet are not located within the field

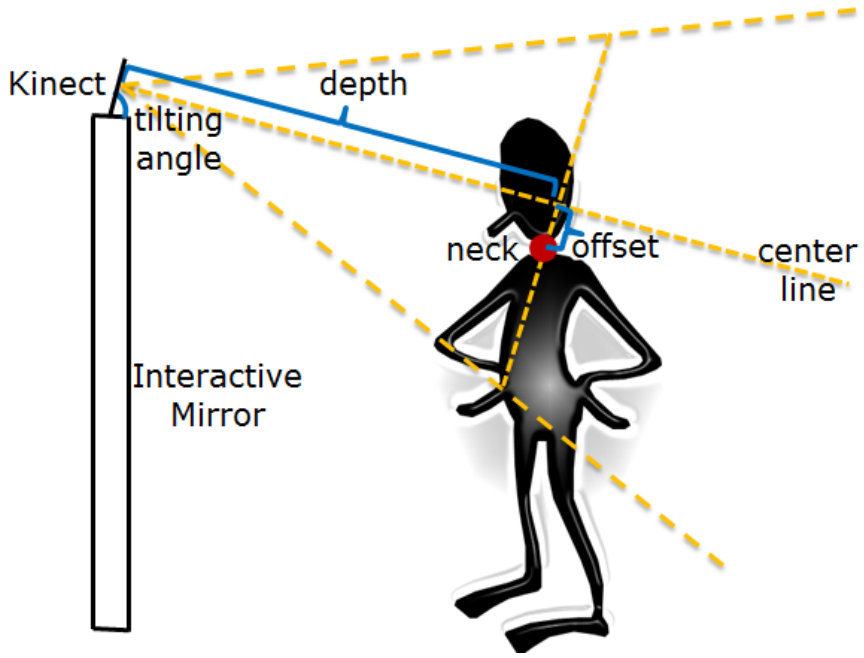

Figure 4.6. Shoulder height estimation when the user's feet are not in the field of view of Kinect. The tilting angle of the Kinect sensor, the depth of the neck joint, and the offset of the neck joint with respect to the center point of the depth image can jointly determine the physical height of the neck joint in the world space.

of view of Kinect. In such scenarios, we can still estimate the neck height from the tilting angle of the Kinect sensor, the depth of the neck joint in the Kinect depth image, and the offset of the neck joint with respect to the center point of the depth image. After the shoulder/neck height is estimated, we then uniformly resize the digital clothes in three dimensions for a better fit to the user's body.

4.2 SKELETAL MOTION TRACKING: OPENNI VS. KWSDK

One key component of a virtual try-on system is to track the motion of the user. We built our motion tracking component on the successful Kinect sensor and publicly available SDKs developed for Kinect. More specifically, we have experimented with

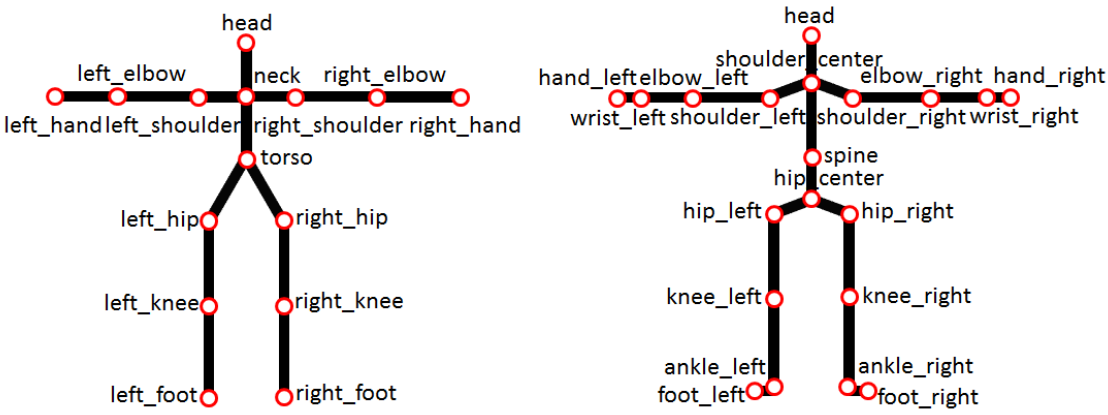

Figure 4.7. Human skeletons defined by OpenNI (left) and KWSDK (right).

two SDKs that provide skeletal motion tracking capability for Kinect. The first one is the OpenNI 1.1.0.41. OpenNI is a set of open source SDKs released by an organization of the same name. It aims to standardize applications that access natural interaction devices. The other SDK we use is Kinect for Windows SDK (KWSDK) 1.5, released by Microsoft to support developers who wish to work with Kinect. Here we begin with an overview of both SDKs, and then we compare their performance related to skeletal motion tracking.

Both OpenNI and KWSDK can query the Kinect sensor for RGB images and depth images up to 30 frames per second. Additionally, both can track a user's skeleton that includes information of positions and orientations of each joint. Their major difference lies in the structure of the returned skeletons, shown in Fig. 4.7. Note that in OpenNI the neck joint always lies on the line that connects the left and right shoulders, while the KWSDK shoulder_center joint does not necessarily lie on the shoulder line.

OpenNI requires a skeleton calibration step before it can track user's poses;

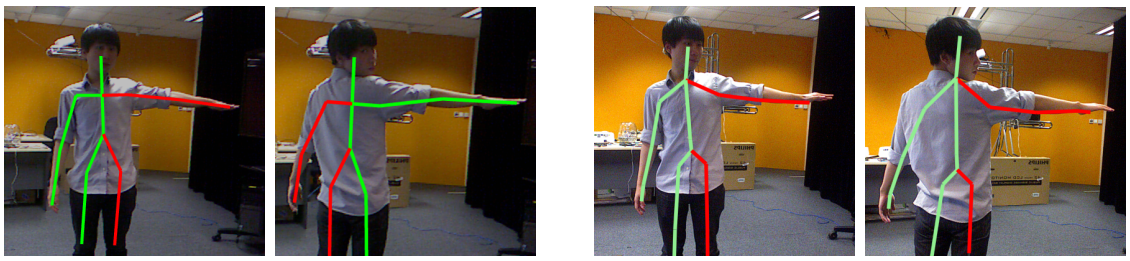

Figure 4.8. Left: OpenNI correctly identifies the left limbs (colored red) regardless the facing direction of the user. Right: yet KWSDK confuses the left and right limbs when the user faces backwards.

while KWSDK can work in a walk in/walk out situation. On the other hand, KWSDK is more prone to false positives, such as detecting chairs as users. In addition, KWSDK cannot correctly identify the right vs. left limbs of the user when she faces backwards away from Kinect. This problem is depicted in Fig. 4.8. The red lines represent the limbs on the left side of the tracked skeleton.

In addition to full-body skeletal tracking, OpenNI provides functionalities such as hand tracking, gesture recognition, background foreground separation etc. KWSDK supports additional capabilities such as seated skeletal tracking, face tracking, speech recognition, background separation etc. Our system currently does not utilize these features and components.

4.2.1 Performance Comparison

We first compare the performance of OpenNI and KWSDK in terms of their joint tracking stability. To this end, we recorded 30 frames (1s) of skeleton data from three subjects holding the standard T-pose standing from various distances (1.5m, 2m, and 2.5m) to the Kinect sensor. The Kinect was placed 185cm above the ground

measured shoulder height (cm)	neck-to-feet OpenNI (cm)	shoulder_center-to-feet KWSDK (cm)	neck height OpenNI (cm)	shoulder_center height KWSDK (cm)
153.4	134.6	153.2	156.8	162.7
151.0	129.5	149.7	153.5	161.8
151.0	116.5	136.0	149.0	158.8
144.2	114.4	141.3	148.2	151.2
143.5	121.3	139.0	147.5	146.0
143.5	117.1	138.9	147.3	148.2
137.6	105.4	131.6	143.7	142.9
135.5	105.0	129.3	142.0	135.6
134.0	106.1	129.0	142.1	137.8

Table 4.1. Comparison of shoulder height estimation between OpenNI and KWSDK. Column 1: manual measurements; Column 2&3: height estimation using neck to feet distance; Column 4&5: height estimation using the method of Fig. 4.6 when feet positions are not available.

and tilted downward 20 degrees. Subject 1 and 2 were males wearing a polo or T-shirt, jeans, and casual sneakers, of height 190.5cm and 173cm respectively. Subject 3 was a 163cm female wearing a blouse, jeans, and flat flip-flops. We then calculated the standard deviation of each joint position for all the visible joints. Fig. 4.9 shows the results, which suggest that the joint tracking stability of OpenNI and KWSDK are roughly comparable. Note that we recorded the T-pose trials with KWSDK while doing online tracking. We then fed the recorded depth data to OpenNI to do offline tracking. Thus the same T-pose trials were used for both SDKs to eliminate the difference caused by users' motion variations. Ideally, we should also capture the same trials using a high-end motion capture system such as Vicon, so that the joint tracking stability of the two SDKs from Kinect data can be compared with ground truth data. Due to space and time constraints, however, we did not perform such comparison. From the average individual joint stability charts in the right column of Fig. 4.9, we can also see that end-effectors such as hands and feet are more unstable compared to inner body joints in both SDKs.

We also compare how OpenNI and KWSDK integrate with our height esti-

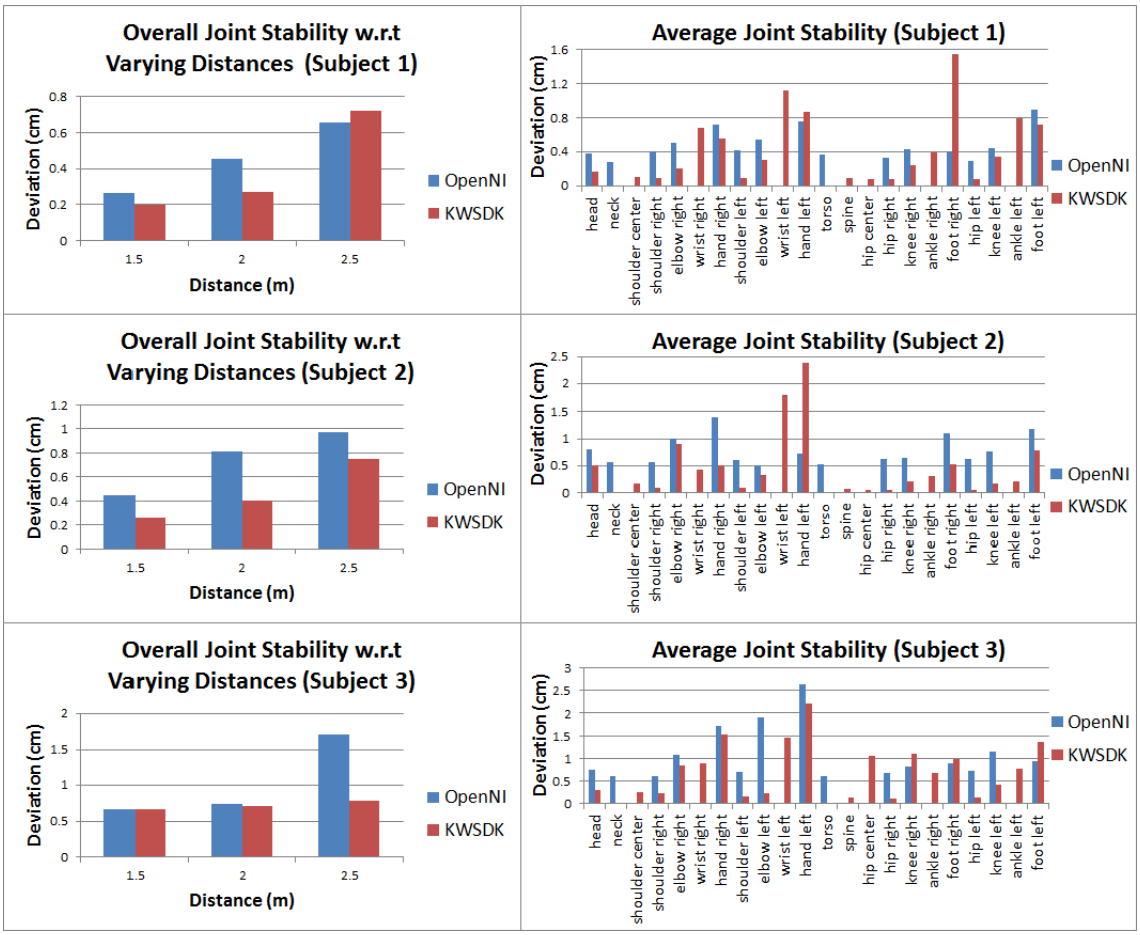

Figure 4.9. Comparison of joint tracking stability between OpenNI and KWSDK. Left: average standard deviation of joint positions in centimeters for all joints across all frames. Right: average standard deviation for each individual joint across all frames.

mation methods described in Section 4.1.4. With Kinect placed 185cm above the ground and titled down 20 degrees, we captured nine subjects wearing T-shirts and jeans and holding the T-Pose for one second two meters away from the mirror. At this distance, the subject's full-body skeleton could be seen. We first simply calculated the average distance from the neck joint (in OpenNI) or shoulder_center joint (in KWSDK) to the mid-point of the feet joints as shoulder height estimation. The results are shown in the second and third columns of Table 4.1. Second,

we used the method depicted in Fig. 4.6 for height estimation without using the feet positions. The results are shown in the fourth and fifth columns of Table 4.1. The first column of Table 4.1 lists our manual measurement of the vertical distance between the floor to the mid-point of the clavicles. This is the shoulder height that our clothes-body fitting algorithm expects to overlay the virtual clothes. We can see from Table 4.1 that feet-to-neck heights tend to underestimate the shoulder heights, mainly because there is usually a distance between the feet and the ground that is not compensated for by the first height estimation method. For the second approach that does not use feet positions, such underestimation is eliminated. On the other hand, KWSDK tends to overestimate the height now, mainly because its shoulder_center joint usually locates above the shoulder line, as shown in Fig. 4.7 right.

4.3 DISCUSSION

Our system offers several advantages over traditional retailing. It attracts more customers through providing a new and exciting retail concept, and creates interest in the brand and store by viral marketing campaigns through customers sharing their experiences in Social Media such as Facebook. Furthermore, it reduces the need for floor space and fitting rooms, thereby reducing rental costs and shortening the time for trying on different combinations and making purchase decisions. We encourage interested readers to search our demo videos with keywords EON Interactive Mirror at http://www.youtube.com.

We have closely engaged participating retailers during the content creation

process in an iterative review process to ensure the high quality of interactive 3D clothes from catalog images. Thus the retailers and shopping mall operators were confident and excited to feature their latest fashion lineups with EON Interactive Mirror. The try-on system was strategically placed in the high traffic flow area of the shopping mall, and successfully attracted many customers to try on the virtual clothes and bags. The retailers appreciated the value of the system as a crowd puller, and to allow other passers-by to see the interaction when somebody is trying clothes with the Interactive Mirror. We have also observed that interactions with the system were often social, where either couples or group of shoppers came together to interact with the mirror. They took turns to try the system, and gave encouragement when their friend or family was trying. Notably the system also attracted families with young children to participate. In this case, the parents would assist the children in selecting the clothes or bags. Due to limitations of Kinect SDKs, the system would not be able to detect or has intermittent tracking for children shorter than one meter. However, this limitation did not stop the young children from wanting to play with the Mirror.

Currently there are several limitations of our system. First, the manual content creation process for 3D clothes modeling is labor intensive. Automatic or semi-automatic content creation, or closer collaboration and integration with the fashion design industry will be needed to accelerate the pace of generating digital clothing for virtual try-on applications. Additionally, our current clothes fitting algorithm scales the outfit uniformly. This is problematic when the user is far way from the

standard portion. For instance, a heavily over-weighted person will not be covered entirely by the virtual clothes because of her excessive width. Extracting relevant geometry information from the Kinect depth data is a potential way to address this problem.

In the future, we wish to augment the basic try-on system with an additional recommendation engine based on data analytics, so that the system could offer customers shopping suggestions 'on the fly' regarding suitable sizes, styles, and combinations to increase sales of additional clothes or promote matching accessories. The system could also be used to gather personalized shopping preferences, and provide better information for market research on what create just an interest to try versus a decision to buy. We would also like to explore the possibility of adapting our system for Internet shopping, for customers who have a Kinect at home. In this scenario, we will simply use the RGB camera in Kinect rather than an additional HD camera.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this paper, we have reviewed some approaches to do character animation from three different perspectives; kinematic, data-driven, and physics-based. We further applied these techniques to interactive avatar control. We showed two of our works, one of which is about deploying physics-based avatar control onto several simulation engines and the other is about using performance-based avatar control to create a virtual try-on experience. These two case studies showcased physics-based and performance-based approaches for interactive avatar control. The physics-based approach is capable of producing physically realistic motions. Using a PD controller to track a small number of key poses obtained using a data-driven animation approach, physics-based controllers are also able to produce motions that look less robotic and are responsive enough to disturbances. Physics-based approaches are not perfect however, since they are often configured to work well under specific circumstances and take a lot of computation. The performance-based approach on the other hand offers a cheap and fast avatar control mechanism with currently available technologies, although high end systems are still required to get high quality results that are less noisy and more accurate.

The advancement of character animation still leaves us with many open problems, for example physics-based avatar control which incorporates soft-body dynamics. As we see from earlier sections, the work we discussed mainly focuses on

characters consisting of rigid bodies. Ongoing research is being done on modeling animated character using soft-bodies. The problem is challenging because it involves more sophisticated dynamics to consider.

Another open problem is to implement physics-based avatar control for games. Most games still utilize kinematic instead of physics-based avatar control due to the assumption of the heavy computation used by physics-based controllers. However, research on physics-based controllers has resulted in faster control algorithms, thus we feel that it has become more and more possible to actually implement it in today's games.

We have also briefly discussed interactive character animation interfaces which are often used in performance-based avatar control. The goal of interactive character animation interfaces is to design a robust system that enables users to interact and drive virtual characters in a simple, intuitive, and fun way. The various options to implement the components of the system, each with its own implementation issues, make it an interesting topic in character animation research. Future work might involve exploring more options for the components that make up a character animation interface system. A point of interest is to use various available interfaces such as Kinect, emotiv, and IPad as the system's front end to drive virtual characters. Another possible research direction is to utilize physics-based animation engines (e.g. LocoTest) and humanoid robots[1] as the system's back end.

[1] http://www.aldebaran-robotics.com/

REFERENCES

[1] Arikan, O., Forsyth, D.A.: Interactive motion generation from examples. In: Proceedings of the 29th annual conference on Computer graphics and interactive techniques. pp. 483–490. SIGGRAPH '02, ACM, New York, NY, USA (2002)

[2] Aristidou, A., Lasenby, J.: Fabrik: A fast, iterative solver for the inverse kinematics problem. Graph. Models 73, 243–260 (September 2011)

[3] Baak, A., Müller, M., Bharaj, G., Seidel, H.P., Theobalt, C.: A data-driven approach for real-time full body pose reconstruction from a depth camera. In: IEEE 13th International Conference on Computer Vision (ICCV). pp. 1092–1099. IEEE (Nov 2011)

[4] Beaudoin, P., Coros, S., van de Panne, M., Poulin, P.: Motion-motif graphs. In: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 117–126. SCA '08, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2008)

[5] Buss, S.R.: Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods. (Apr 2004)

[6] Chai, J., Hodgins, J.K.: Performance animation from low-dimensional control signals. In: ACM SIGGRAPH 2005 Papers. pp. 686–696. SIGGRAPH '05, ACM, New York, NY, USA (2005)

[7] Chai, J., Hodgins, J.K.: Constraint-based motion optimization using a statisti-

cal dynamic model. In: ACM SIGGRAPH 2007 papers. SIGGRAPH '07, ACM, New York, NY, USA (2007)

[8] CMLabs Simulations, I.: Vortex 5.0.1 vx developer guide (2011)

[9] Cooper, S., Hertzmann, A., Popović, Z.: Active learning for real-time motion controllers. In: ACM SIGGRAPH 2007 papers. SIGGRAPH '07, ACM, New York, NY, USA (2007)

[10] Coros, S., Beaudoin, P., van de Panne, M.: Generalized biped walking control. In: ACM SIGGRAPH 2010 papers. pp. 130:1–130:9. SIGGRAPH '10, ACM, New York, NY, USA (2010)

[11] Coros, S., Karpathy, A., Jones, B., Reveret, L., van de Panne, M.: Locomotion skills for simulated quadrupeds. In: ACM SIGGRAPH 2011 papers. pp. 59:1–59:12. SIGGRAPH '11, ACM, New York, NY, USA (2011)

[12] Coumans, E.: Bullet 2.76 physics sdk manual (2010)

[13] Fernando, C.L., Igarashi, T., Inami, M., Sugimoto, M., Sugiura, Y., Withana, A.I., Gota, K.: An operating method for a bipedal walking robot for entertainment. In: ACM SIGGRAPH ASIA 2009 Art Gallery & Emerging Technologies: Adaptation. pp. 79–79. SIGGRAPH ASIA '09, ACM, New York, NY, USA (2009)

[14] Ganapathi, V., Plagemann, C., Koller, D., Thrun, S.: Real time motion capture using a single time-of-flight camera. In: CVPR. pp. 755–762 (2010)

[15] Geijtenbeek, T., Pronost, N., Egges, A., Overmars, M.H.: Interactive Character Animation using Simulated Physics. pp. 127–149

[16] Giovanni, S., Choi, Y.C., Huang, J., Tat, K.E., Yin, K.: Virtual try-on using kinect and hd camera. In: MIG. pp. 55–65 (2012)

[17] Giovanni, S., Yin, K.: Locotest: Deploying and evaluating physics-based locomotion on multiple simulation platforms. In: MIG. pp. 227–241 (2011)

[18] Ha, S., Bai, Y., Liu, C.K.: Human motion reconstruction from force sensors. In: Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 129–138. SCA '11, ACM, New York, NY, USA (2011)

[19] Heck, R., Gleicher, M.: Parametric motion graphs. In: Proceedings of the 2007 symposium on Interactive 3D graphics and games. pp. 129–136. I3D '07, ACM, New York, NY, USA (2007)

[20] Hodgins, J.K., Wooten, W.L., Brogan, D.C., O'Brien, J.F.: Animating human athletics. In: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. pp. 71–78. SIGGRAPH '95, ACM, New York, NY, USA (1995)

[21] Kovar, L., Gleicher, M., Pighin, F.: Motion graphs. In: Proceedings of the 29th annual conference on Computer graphics and interactive techniques. pp. 473–482. SIGGRAPH '02, ACM, New York, NY, USA (2002)

[22] Kuo, A.D., Donelan, J.M., Ruina, A.: Energetic consequences of walking like an inverted pendulum: step-to-step transitions. Exercise and sport sciences reviews 33(2), 88–97 (Apr 2005)

[23] Kwon, T., Hodgins, J.: Control systems for human running using an inverted pendulum model and a reference motion capture sequence. In: Proceedings of

the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 129–138. SCA '10, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2010)

[24] de Lasa, M., Mordatch, I., Hertzmann, A.: Feature-based locomotion controllers. In: ACM SIGGRAPH 2010 papers. pp. 131:1–131:10. SIGGRAPH '10, ACM, New York, NY, USA (2010)

[25] Lee, J., Chai, J., Reitsma, P.S.A., Hodgins, J.K., Pollard, N.S.: Interactive control of avatars animated with human motion data. In: Proceedings of the 29th annual conference on Computer graphics and interactive techniques. pp. 491–500. SIGGRAPH '02, ACM, New York, NY, USA (2002)

[26] Levine, S., Lee, Y., Koltun, V., Popović, Z.: Space-time planning with parameterized locomotion controllers. ACM Trans. Graph. 30, 23:1–23:11 (May 2011)

[27] Li, Y., Wang, T., Shum, H.Y.: Motion texture: a two-level statistical model for character motion synthesis. In: Proceedings of the 29th annual conference on Computer graphics and interactive techniques. pp. 465–472. SIGGRAPH '02, ACM, New York, NY, USA (2002)

[28] Liu, G., Zhang, J., Wang, W., McMillan, L.: Human motion estimation from a reduced marker set. In: Proceedings of the 2006 symposium on Interactive 3D graphics and games. pp. 35–42. I3D '06, ACM, New York, NY, USA (2006)

[29] Liu, L., Yin, K., van de Panne, M., Shao, T., Xu, W.: Sampling-based contact-rich motion control. In: ACM SIGGRAPH 2010 papers. pp. 128:1–128:10. SIG-

GRAPH '10, ACM, New York, NY, USA (2010)

[30] Macchietto, A., Zordan, V., Shelton, C.R.: Momentum control for balance. In: ACM SIGGRAPH 2009 papers. pp. 80:1–80:8. SIGGRAPH '09, ACM, New York, NY, USA (2009)

[31] Mordatch, I., de Lasa, M., Hertzmann, A.: Robust physics-based locomotion using low-dimensional planning. In: ACM SIGGRAPH 2010 papers. pp. 71:1–71:8. SIGGRAPH '10, ACM, New York, NY, USA (2010)

[32] Muico, U., Popović, J., Popović, Z.: Composite control of physically simulated characters. ACM Trans. Graph. 30, 16:1–16:11 (May 2011)

[33] NVIDIA: Physx sdk 2.8 documentation (2008)

[34] Raibert, M.H., Hodgins, J.K.: Animation of dynamic legged locomotion. In: Proceedings of the 18th annual conference on Computer graphics and interactive techniques. pp. 349–358. SIGGRAPH '91, ACM, New York, NY, USA (1991)

[35] Ren, C., Zhao, L., Safonova, A.: Human motion synthesis with optimization-based graphs. Comput. Graph. Forum 29(2), 545–554 (2010)

[36] Sims, K.: Evolving virtual creatures. In: Proceedings of the 21st annual conference on Computer graphics and interactive techniques. pp. 15–22. SIGGRAPH '94, ACM, New York, NY, USA (1994)

[37] Smith, R.: Open dynamics engine v0.5 user guide (2006)

[38] Tan, J., Gu, Y., Turk, G., Liu, C.K.: Articulated swimming creatures. In: ACM SIGGRAPH 2011 papers. pp. 58:1–58:12. SIGGRAPH '11, ACM, New

York, NY, USA (2011)

[39] Tang, Z., Er, M.J., Chien, C.J.: Analysis of human gait using an inverted pendulum model. In: FUZZ-IEEE. pp. 1174–1178 (2008)

[40] Tolani, D., Goswami, A., Badler, N.I.: Real-time inverse kinematics techniques for anthropomorphic limbs. Graph. Models Image Process. 62(5), 353–388 (Sep 2000)

[41] Wang, J.M., Fleet, D.J., Hertzmann, A.: Optimizing walking controllers. In: ACM SIGGRAPH Asia 2009 papers. pp. 168:1–168:8. SIGGRAPH Asia '09, ACM, New York, NY, USA (2009)

[42] Wang, J.M., Fleet, D.J., Hertzmann, A.: Optimizing walking controllers for uncertain inputs and environments. In: ACM SIGGRAPH 2010 papers. pp. 73:1–73:8. SIGGRAPH '10, ACM, New York, NY, USA (2010)

[43] Wei, X.K., Chai, J.: Intuitive interactive human-character posing with millions of example poses. IEEE Computer Graphics and Applications 31(4), 78–88 (2011)

[44] WillowGarage: Opencv. `http://opencv.org/`

[45] Wu, C.C., Zordan, V.: Goal-directed stepping with momentum control. In: Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 113–118. SCA '10, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2010)

[46] Wu, J.c., Popović, Z.: Terrain-adaptive bipedal locomotion control. In: ACM SIGGRAPH 2010 papers. pp. 72:1–72:10. SIGGRAPH '10, ACM, New York,

NY, USA (2010)

[47] Xie, L., Kumar, M., Cao, Y., Gracanin, D., Quek, F.: Data-driven motion estimation with low-cost sensors. IET Conference Publications 2008(CP543), 600–605 (2008)

[48] Ye, Y., Liu, C.K.: Optimal feedback control for character animation using an abstract model. In: ACM SIGGRAPH 2010 papers. pp. 74:1–74:9. SIGGRAPH '10, ACM, New York, NY, USA (2010)

[49] Ye, Y., Liu, C.K.: Synthesis of responsive motion using a dynamic model. Comput. Graph. Forum 29(2), 555–562 (2010)

[50] Yin, K., Coros, S., Beaudoin, P., van de Panne, M.: Continuation methods for adapting simulated skills. In: ACM SIGGRAPH 2008 papers. pp. 81:1–81:7. SIGGRAPH '08, ACM, New York, NY, USA (2008)

[51] Yin, K., Loken, K., van de Panne, M.: Simbicon: simple biped locomotion control. In: ACM SIGGRAPH 2007 papers. SIGGRAPH '07, ACM, New York, NY, USA (2007)

[52] Yin, K., Pai, D.K.: Footsee: and interactive animation system. In: Eurographics/SIGGRAPH Symposium on Computer Animation. ACM (2003)

[53] Zhao, L., Safonova, A.: Achieving good connectivity in motion graphs. In: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 127–136. SCA '08, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2008)