Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Graphics 29 (2005) 931–945
COMPUTERS & GRAPHICS
www.elsevier.com/locate/cag

Pen-to-mime: Pen-based interactive control of a human figure

Masaki Oshita*

Kyushu Institute of Technology, 680-4 Kawazu, Iizuka, Fukuoka, 820-8502, Japan

Abstract

In this paper a pen-based intuitive interface is presented, that controls a virtual human figure interactively. Recent commercial pen devices can detect not only the pen positions but also the pressure and tilt of the pen. We utilize such information to make a human figure perform various types of motions in response to the pen movements manipulated by the user. The figure walks, runs, turns and steps along the trajectory and speed of the pen. The figure also bends, stretches and tilts in response to the tilt of the pen. Moreover, it ducks and jumps in response to the pen pressure. Using our interface, the user controls a virtual human figure intuitively as if he or she were holding a virtual puppet and playing with it.

In addition to the interface design and implementation, this paper describes a motion generation engine to produce various motion based on varying parameters that are given by the pen interface. We take a motion blending approach and construct motion blending modules with a set of small number of motion capture data for each type of motions. Finally, we present the results from user experiments and comparison with a transitional gamepad-based interface.

Keywords: Pen-based interface; Motion control; Computer animation

1. Introduction

There are many demands for an interactive motion control of a virtual character on desktop environments. However, this has been a difficult challenge in the computer graphics field. Since human figures have a large number of degrees of freedom (DOF) and their movements are complicated, it is not easy to control them through a common device that has only a small number of DOF such as a mouse or gamepad. Using motion capture equipment, full body motion of a virtual character can be directly controlled by mapping the movements of an actor to the virtual character. However, motion capture is expensive and needs a large space. They are not suitable for a desktop application. Currently most interactive applications such as computer games employ a traditional input device such as a mouse, keyboard or gamepad. Therefore, the range of control is very limited. Although it is possible to control complex motions by combining multiple gamepads and/or mouse, the user needs practice to learn such an interface design since the mapping from the multiple input devices to the movements of a controlled figure may not be intuitive.

In this paper we present a pen-based intuitive interface to control a virtual human figure interactively. The key idea is that we use a pen as an approximation of a figure and map pen moments to the figure motion (Fig. 1). Recent commercial pen devices can detect not only the pen positions but also the pressure and tilt of the pen. We utilize such information to make a human figure perform various types of motions in response to the pen movements manipulated by the user. A figure walks, runs, turns, steps, and jumps along the trajectory and speed of the pen. The figure also bends, stretches

*Tel.: +81 948 29 7718; fax: +81 948 29 7709.
E-mail address: oshita@ces.kyutech.ac.jp.

0097-8493/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2005.09.010

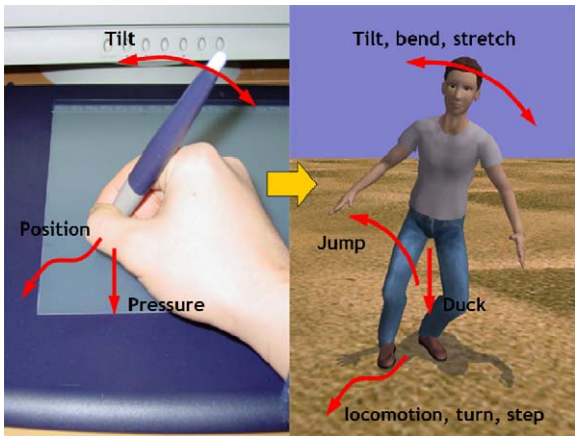

Fig. 1. The pen-based interface. The figure movement is associated with the pen manipulated by the use. The positions, pressure, and tilt of the pen are used to make the figure perform various motions.

and tilts in response to the tilt of the pen. In addition, it ducks in response to the pen pressure. Using this pen-based interface, the user controls a virtual human figure intuitively as if he or she were holding a virtual puppet and playing with it.

The main targets of this work are interactive applications that especially require various types of locomotion control such as telexistence [1], virtual theaters [2], multi-user environments, computer games, animation editing, etc. Current common gamepad devices are enough for a simple locomotion. However, they mostly do not provide any way to control the styles of motions and generated motions are monotonous. Using our interface, the styles and speed of motions naturally change in response to the pen movements. The users can express their feelings by making a virtual figure move in various ways. We think that our interface is interesting for the users and easy to use even for kids or novices.

This work is an extension of our previous work [3]. A major difference between this work and the previous work is that a position-based motion control scheme is introduced in this paper. The previous work [3] presented a velocity-based control which maps the pen velocity to the locomotion speed of a human figure directory in a similar way to existing gamepad-based interface. The interface was intuitive but did not fully exploit the advantages of the pen device. As a result, the users who were familiar with existing gamepad-based interfaces could not find the interface very useful. In this paper we present a position-based interface that also combines the benefits of velocity-based methods. Using this interface, the user can control the position of the figure precisely. In addition, the interface is still interactive and the controlled figure responds to the pen movements quickly.

In addition to the interface design and implementation, in this paper a motion generation engine is described, which produces various motions based on the parameters and motion trajectories that are given through the pen interface. We take a motion blending approach and construct motion blending modules with a set of small number of motion capture data for each type of motions: standing movements, locomotion, turn, step, and vertical and directional jumps. Spatial and orientational constraints for the root of a figure are applied to the blended motions so that the generating motion follows a given pen trajectory. We also introduce a motion transition scheme and foot constraints for generating continuous and natural motions. Finally, we introduce some experiments and comparison with a transitional gamepad-based interface. We then discuss the effectiveness of the interface.

This paper is organized as follows. In Section 2 we review some related works. In Section 3 the pen-based interface from the user's viewpoint is introduced. A system overview, interface implementation, and motion generation implementation are given in Sections 4, 5 and 6, respectively. Finally, we discuss our interface based on some user experiments in Section 7. Section 8 concludes this paper and shows future work.

2. Related work

In this section, we discuss related works from two points of view: motion control interface and locomotion generation.

2.1. Motion control interface

Many researchers have developed locomotion systems that generate a realistic walking or running motion along a user specified path [4–7]. On these systems, the user can draw a path using a mouse or pen device, a walking or running motion is then planned and executed based on the given trajectory. However, in most of the systems, the user can only specify the trajectory of locomotion. Other parameters such as speed or styles cannot be controlled. They focus on generating natural walking motion along a given curved path or/and on a curved terrain rather than sophisticated user interface or making various types of motions. Recently, Thorne et al. [8] developed an animation system that generates various types of motions based on "gestures" that are drawn by the user along with a locomotion path. In these systems, a motion is usually generated after an entire path is given. Therefore, the user cannot control the figure during a motion interactively.

Some researches employ input devices that have multiple DOF for animation. Oore et. al. [9] used two bamboo tubes in which a 6 DOF magnetic tracker is

embedded for each tube. Dontcheva et al. [10] used small widgets and a camera-based motion capture system. In these systems, the movements of a device are directly mapped to a part of the subject body. However, virtual figures have more DOF than such input devices. To solve this problem, they proposed layered editing techniques. By repeating specifying the movements for each body parts [9] or from an abstract motion to detailed motions [10], complex motions are composed interactively in their systems. These systems are motion editing tools rather than motion control interface. Since they need some iteration to generate one motion, they are not suitable for a motion control interface on an interactive application.

Laszlo et al. [11] proposed an animation system to control an articulated figure interactively through a mouse or keyboard by introducing a physics-based model. They mapped an input device to the key DOF of a figure, then the movements of all DOF are simulated using a physics-based model. As a result, continuous and natural-looking motions are generated interactively using a common input device. However, this method is not suitable for generating complex motions such as various styles of locomotion since in such motions many DOF of a figure should be controlled cooperatively rather than just follow the laws of physics.

While the above methods map acquired data from some input devices to a particular DOF of a target articulated figure, our method uses them as an abstract parameters (speed, angle, tilt, etc.) for motion generation modules that are constructed in advance. Our system is aimed at "motion control" rather than "motion editing". There are some devices that have more DOF than pen device. For example, a magnetic sensor has 6 DOF (position and rotation of sensor) and so does the 3D mouse [12]; however, we think that pen device is more suitable for motion control of biped figure because pen device gives the user some intuitive physical feedback. When a pen is pressed on the tablet, it gives back reactive forces to the user. When it is tilted, the user feel the moments caused by the vertical axis of the pen as discussed in [9]. These feedback forces help the user sense the state of the figure. In addition to those benefits, pen devices are more convenient and inexpensive than other devices.

There are some techniques which utilize much higher DOF from a device such as foot pressure sensor pad [13] or silhouette image of a human figure from camera vision [7]. These systems use input data to find an appropriate motion from a database using a similarity search rather than just use the input to control a figure's DOF directly. Therefore, it is difficult to control the resulting motion subtly. Moreover, such a system needs a large space as much as a motion capture. Monkey2 [14], an articulated figure device that has the same DOF with human body, is suitable for a key posture specification in an off-line animation editing but for interactive motion control.

Davis et al. [15] proposed a sketching animation system. This system extracts 3D keyframe postures of the skeleton from a sequence of 2D images that are drawn by the user for each desired keyframe. Although their work and ours have a similarity in that both use a pen device as interface, we aim at a totally different technique. Our system uses a pen as an input device and as a metaphor of a human figure rather than a drawing tool. Lately many researchers use pen devices as an intuitive interface for a graphic system [16] and a research for the effectiveness of the pen pressure in 2D GUI is also reported [17]. However, to the author's knowledge, no previous method utilizes multi-dimensional input that is acquired from a pen device such as pressure and tilt for interactive motion control.

2.2. Locomotion generation

Locomotion is a complex motion and many motion generation techniques that are targeted for locomotion have been developed by researches. However, many existing methods can change resulting motions through very few parameters and lack controllability. Most of them [4–6] generate a walking motion based on just a given path. Therefore, these methods cannot be used in our system because we attempt to change motions based on multi-dimensional parameters that are given from a pen device (speed, angle, bend, tilt, and duck). In addition, we expect to generate motions of various characters (e.g. man, woman, child, elder, fashion model, soldier, etc.) by replacing the motion data set.

To address these problems, we need to take an appropriate approach first. Existing locomotion generation techniques are categorized into four approaches:

(1) Procedural motion generation [18]
(2) Physics-based motion generation [19,20]
(3) Motion database or motion graph [5–7]
(4) Motion blending [4,21–24]

First, the procedural approach is to generate walking motion based on some experimentally designed functions. It is difficult to adapt to various types of characters because locomotion models are carefully tuned and designed for a particular character and difficult to be modified. Second, the physics-based approach is similar to the procedural method except that it combines physics-based dynamic simulation. By controlling joint torques based on walking motion trajectories and using a dynamic simulation, physically correct motion is generated. However, this method is more difficult to adapt to wide range of motions because the parameters of controllers should be tuned by hand.

The third and fourth approach uses a set of motion capture data. Because they are based on actual human motion data, resulting motions are expected to be realistic. In addition, these methods can be adapted to various characters by changing the data set. The difference between two approaches is that motion blending methods use some motion data at the same time and generate a new motion by blending the multiple motion data while the other approach selects an appropriate motion segment from a data set and adjusts it. Lately, the motion graph approach is commonly used by many researches [5,6]. This approach seems to be easy to use since motion data do not have to be edited, aligned, and parameterized. However, if the parameter space is large, it is difficult to apply this approach because many motions are required. Moreover, using this method, it is difficult to satisfy a set of given parameters correctly. Therefore, we decided to take the motion blending approach.

The motion blending methods are further categorized into two methods: local and global blending. A local blending method [24,4,23] uses only $n+1$ example motions that are close to the given parameter in an n-dimensional parameter space. This method ensures the resulting motions are close to the original motions. However, this method needs many example motions especially when the dimension of the parameter space is large. A global blending method [22,21] uses the whole example motions to generate resulting motions. Although this approach does not need so much sample data, the sample data should be well scattered so that they represent the motion well over the parameter space. The local blending method allows only interpolation but exploration while the global blending method allows both. In addition, on the local blending method, it is difficult to change motion parameters during a motion since the motion set may be changed. Based on these features, we decided to use the global blending approach because we use a higher dimensional parameter space and smaller numbers of example motions are desired.

3. User interface

In this section, we describe the user interface design of our system from the user's view point. The algorithm for generating animation based on user input will be described in the following sections.

3.1. Tablet device

Among some commercial pen and tablet devices available, we used intuos 2 from Wacom [25]. Intuos 2 is one of the popular products and can detect various status of a pen as listed in Table 1. The position data are absolute values at where the pen is pointing in the tablet coordinates. When the pen touches the tablet, some positive pressure value is acquired. Otherwise the pressure value is zero. The altitude and azimuth show the tilt of pen. Unfortunately only 2 DOF of pen rotation are available in Intuos 2 (Fig. 2) and the direction that the figure is facing cannot be controlled by using the revolution of the pen. Therefore, we introduce a scheme to determine whether the figure faces the direction that the pen is moved or keeps its direction while backward or lateral step as described in Section 3.3.2. Although negative altitude values are acquired when the pen is inverted, our system only uses positive values of altitude. The positions, altitude, azimuth and button values are acquired even when the pen does not touch the tablet unless the pen moves more than about 1 cm above the tablet. If the pen moves far away from the tablet, no input data is captured. These input data from tablet device are easily captured in a user program using the Wacom tablet library [25].

Table 1
Input data available from Intuos 3

Input type	Resolutions	Main use in our interface
x-position	0.01 mm	Locomotion, turn, steps
y-position	0.01 mm	and jump
Pressure	1024	Duck and jump height
Altitude	−90 to 90 (64)	Bend, stretch, tilt
Azimuth	0–360 (64)	
Buttons	2	

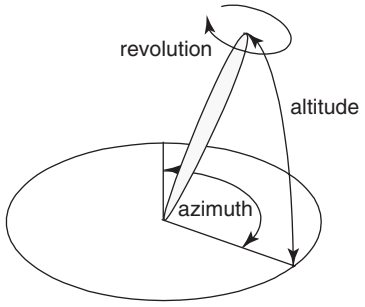

Fig. 2. The pen tilt information from intuos 2. Only altitude and azimuth are detectable but revolution.

3.2. Position-based vs. velocity-based control

There are two basic approaches for locomotion control interface: position-based and velocity-based. The first one takes absolute position values from an input device and uses them to control the position of a figure directory. On the other hand, the second one takes

relative values and uses them to control the motion velocities of a figure. A gamepad or joystick device is considered as an input device that gives relative values. Therefore, it is natural that we use them for a velocity-based control. In a velocity-based interface, a figure moves in response to the tilt of the stick. The user cannot specify a destination of a locomotion directory. Instead, we interactively control the speed and direction of locomotion so that the figure heads for the desired location. On the other hand, a mouse or pen device takes absolute positions. We can use them for position-based control. Using a position-based interface the user can directory specify a destination or a path of locomotion. A disadvantage of the position-based control was the lack of controllability and interactivity as described in Section 2.1.

In the following subsections, we present a position-based pen interface which also combines benefits of a velocity-based control. In our interface, the user can control the speed, direction, or styles of motions interactively in a position-based interface and the controlled figure responds to the pen movements very quickly.

3.3. Interface design

The design concept of our interface is that we use a pen as an approximation of a human figure and the figure intuitively responds to the movements of the pen manipulated by the user. We do not attempt to make the user input some gestures to specify the types of motions explicitly. The user just moves the pen in the same way that the user wants the figure to move around. Appropriate motions are then automatically generated in response to the pen trajectories or strokes.

3.3.1. Locomotion

The figure walks and runs along a given pen trajectory. In our interface, the figure starts walking after a short path enough for at least one step is given (Fig. 3(a)). The user does not have to input an entire trajectory before locomotion starts. The locomotion speed is controlled based on the velocity of the pen on each point of the path trajectory. If the pen is moved slowly, the figure also walks slowly (Fig. 3(b)). If the user draws a path quickly, the figure runs (Fig. 3(c)). When the figure catches up to the current pen position, it slows down in order to wait until further trajectory is given. There is also a speed limit of figure running in order to avoid unnatural animation. Therefore, if the pen is moved very quickly, the difference between the current pen position and the figure position become large and the figure takes some time to finish the given trajectory. Our system displays the pen trajectory on the screen in order to help the user draw a desired locomotion path.

The user can start path drawing from any point on the tablet. The start point on the tablet is mapped to the figure position in the world coordinates. If the pen is moved away from the tablet and touched on another position and started to be moved again, the figure keeps locomotion from the terminal position of the last trajectory. Therefore, by changing the pen position when the pen reaches the edge of the tablet, the user can make the figure walk far away from the original position without being limited by the dimension of the tablet.

3.3.2. Step and turn

Since the revolution of the pen is not undetectable as explained in Section 3.1, we need some rules in order to determine figure orientation. For example, when the pen is moved backward against the figure, it is difficult to decide whether the figure should step back while facing the front or should turn back and then walk straight.

Our system determines whether step or turn is executed based on the length of a pen stroke (Fig. 4). If the pen is moved shortly and stopped, a step motion is executed based on the distance and orientation of the stroke (Fig. 5(a)). If the pen is moved further than the

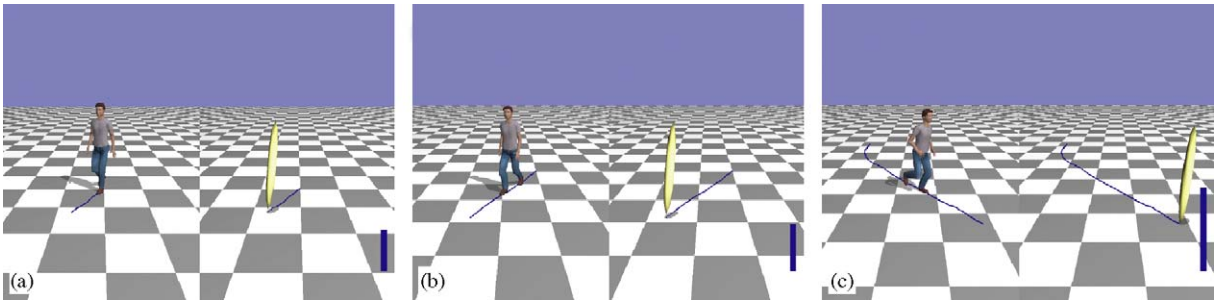

Fig. 3. Locomotion interface. The right view visualizes the movements of the pen that is manipulated by the user. The yellow stick shows the pan movements. The figure walks and runs along a given pen trajectory. The locomotion speed varies based on the pen speed at each point on the trajectory.

distance of single step, the figure turns toward the direction and then starts walking (Fig. 5(c)). If the user keeps repeating short pen strokes, the figure keeps stepping without turning around. This is also applied to front step and locomotion. If the pen is moved forward very briefly, the figure just steps forward instead of starting to walk.

3.3.3. Jump

A jump motion is executed based on the pen movements as well as locomotion. If the pen is lifted up from the tablet and touched on a nearby position on the tablet, the figure jumps toward the touched position (Fig. 6(a) and (b)). If the pen is touched in a position very close to the pen left, the figure performs a vertical jump (Fig. 6(c)). If the pen is touched on a point out of jump range, the system does not start a jump motion and recognizes it as a start of new input as described in Section 3.3.1. To help the user control the landing position, the system displays the available jump range and the cursor corresponding to the current pen position while the pen is floating on the tablet (Fig. 6(a) and (b)). The jump direction and distance are determined based on the difference between the landing point and the take-off point. The jump height is determined based on the pen pressure just before the pen left the tablet. The more the user presses the pen onto the tablet, the higher the figure jumps. The jump height is currently controllable only for vertical jumps, but for directional jumps in our implementation.

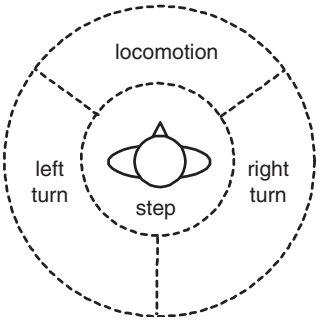

Fig. 4. The relationship between the orientation and distance of the pen stroke and executed motion. Given a short stroke, the figure makes a step in response to the orientation and distance. Given a long stroke or trajectory, the figure turns and/or starts walking.

3.3.4. Posture control (bend, stretch, tilt, and duck)

The posture of the figure is controlled through additional DOF that the pen device provides. The tilt and pressure of pen are used for this purpose. The figure bends, stretches, and tilts its upper body in response to the angle and direction of the pen tilt (Fig. 7(d)–(f)). The figure also ducks in response to the pen pressure (Fig. 7(c)). The height of duck depends on the size of pressure. As long as the pressure is being applied, the figure keeps ducking. These posture controls work

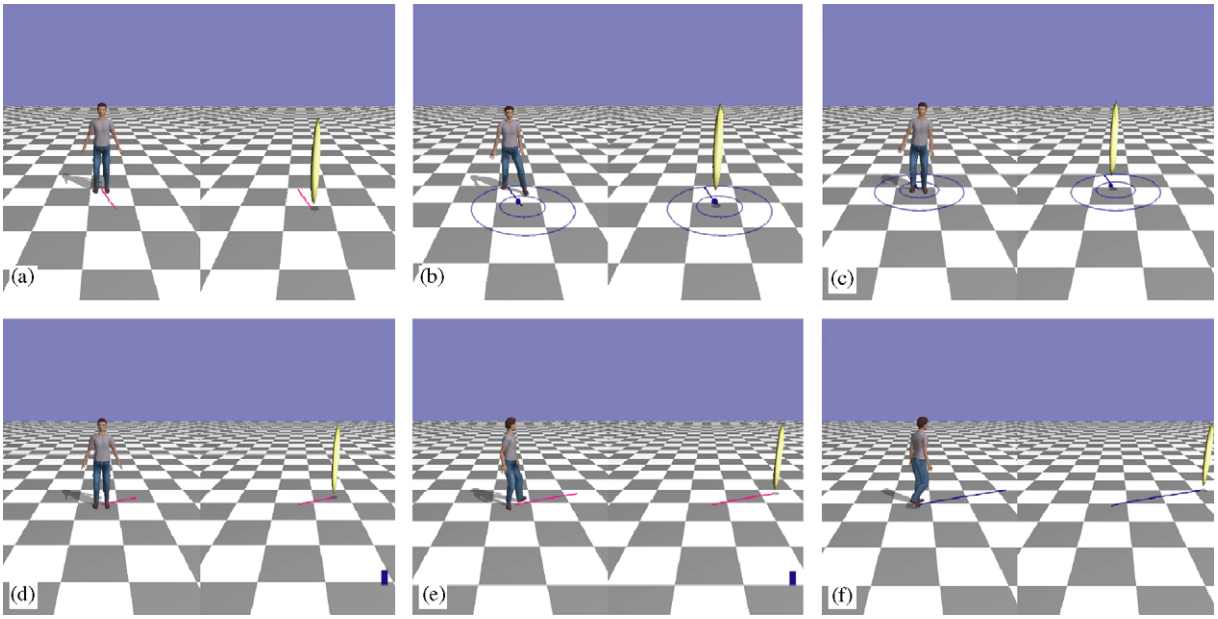

Fig. 5. Step and turn interface. (a)–(c) If a short pen stroke is given, the figure steps along the stroke without it orientation. (d)–(f) If a long stroke or trajectory is given, the figure turns first and starts walking.

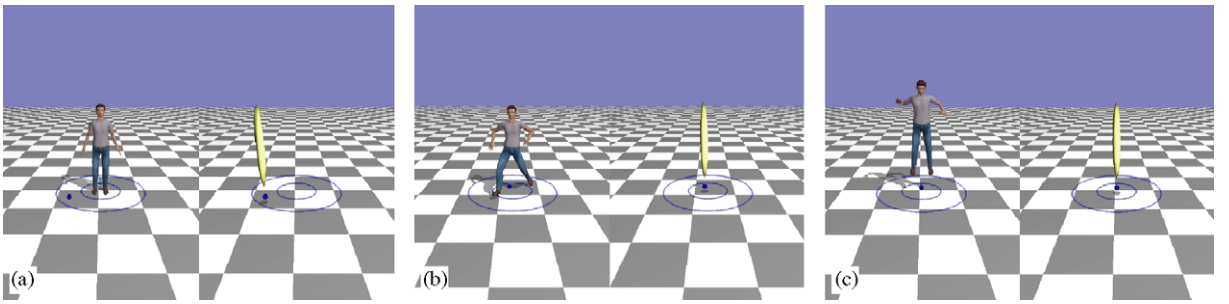

Fig. 6. Jump interface. (a) If the pen is lifted up and touched on a nearby point quickly, (b) the figure jump toward the touched position. If the pen is touched on the same point, (c) a vertical jump is executed. The jump height is determined based on the pen pressure before the pen is released. Guide circles which show the range of vertical jump and directional jump are displayed for the user.

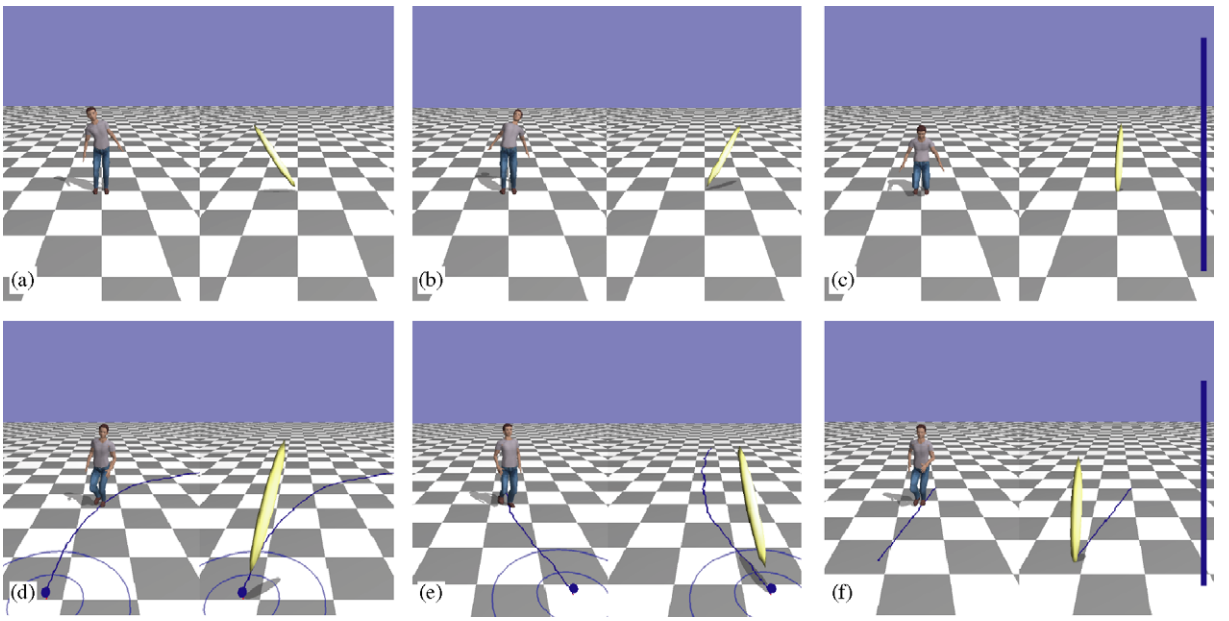

Fig. 7. Posture control interface. (a), (b) The figure tilts, bend sand stretches in response to the pen tilt. (b) The figure also ducks in response to the pen pressure. (d)–(f) The posture control also works during locomotion.

not only when the figure is standing but also during locomotion (Fig. 7(d)–(f)).

4. System overview

The structure of our system is shown in Fig. 8. The system consists of the interface module and the motion generation module. The interface module interprets input data from a tablet device and sends motion parameters, root constraints and motion transition command to the motion generator. The motion generation module is composed of submodules for each type of motion. Based on a current motion, one of submodules takes charge of generating motion. When a motion transition signal is sent from the interface module, the main submodule switches one to another.

4.1. Motion generation modules

The motion submodules are developed based on a global motion blending method as discussed is Section 2.2 with a small number of example motion data. To ensure smooth blending over the examples, every example in a motion module has similar patterns and the same sequence of keyframes, for example, left foot up or right foot down. For this reason, we separate a vertical jump and four orientational jumps, right turn and left turn, and right step and left step, respectively.

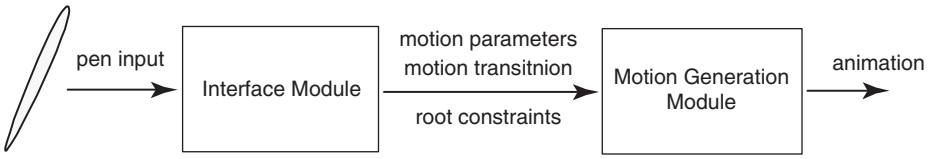

Fig. 8. System overview.

The parameters that each motion module takes are shown in Table 2. Some motion modules such as locomotion are designed to generate one cycle of a continuous motion, e.g. moving the right leg first and the left leg. By repeating the same motion with varying parameters, continuous motions are generated. The parameters are allowed to be changed during motion. Currently we do not terminate a motion before a cycle of motion has finished in order to avoid discontinuous between motions. A motion transition command from the interface module is queued in the motion generation module and is executed after the previous motion has finished.

4.2. Example motions

We need motion parameters of each example motion in order to blend them. The motion parameters are computed from the motion data using simple approximations. Locomotion speed and angle are computed by using a method similar to that of Park et al. [21]. Jump, step, and turn parameters are simply computed from the difference between the first frame and last frame of the example motion. Jump height and tilt, bend, and stretch angle are calculated from the maximum height and spine angle during the motion. To apply motion blending, a sequence of keyframe is manually assigned to all motions in each motion set in advance.

Table 2
Motion modules

Motion	Parameters	# Examples
Locomotion	Speed, angle, tilt, bend-stretch, duck	12
Standing	Tilt, bend-stretch, duck	6
Right step	Angle, distance	10
Left step	Angle, distance	10
Directional jump (front right)	Angle, distance	6
Directional jump (right)	Angle, distance	4
Directional jump (front left)	Angle, distance	6
Directional jump (left)	Angle, distance	4
Vertical jump	Height	2
Right turn	Angle	4
Left turn	Angle	4

5. User interface implementation

This section describes the implementations of the user interface described in Section 3. Since there is some time difference between the current input data from the pen and device and the motion that is currently being executed, input data are recorded in a queue as a series of point data, and then a block of them is interpreted to generate an appropriate motion from the top of the queue. The motion parameters are also computed from the corresponding input data.

5.1. Recording pen trajectory

Input data from the pen device are recorded as a series of point data (Fig. 9). Input data come from the pen device asynchronously while the pen is moving on the tablet. If the pen is not moving, no input data comes. Based on the spatial and temporal difference from the previous and next point data, we categorize each point data into one of four types: moving, stop, jump landing, and reset (Fig. 10). When the pen keeps moving, the point data are recognized as moving data. If the pen is stopped or lifted up from the tablet, it is recognized as a stop data. If the pen is lifted up and positioned on a point in a short distance from the previous position quickly, the data is treated as a landing point of jump. If the distance or time from the previous point is large, the point data is treated as a reset data. In case the pen is moved very quickly on the tablet, we treat the input data as moving data instead of jump landing data in order to avoid unexpected jump motions. These rules are summarized in Fig. 10. As explained in Section 3.3.1, the position of a reset data is assigned to the same position with the last data and the user can continue to extend the previous trajectory (Fig. 9(d)). Based on these other types of the point data a series of point data in the queue is recognized and interpreted as one motion. The pen pressure and tilt at the moment are also recorded on each point data. They are not used to determine the type

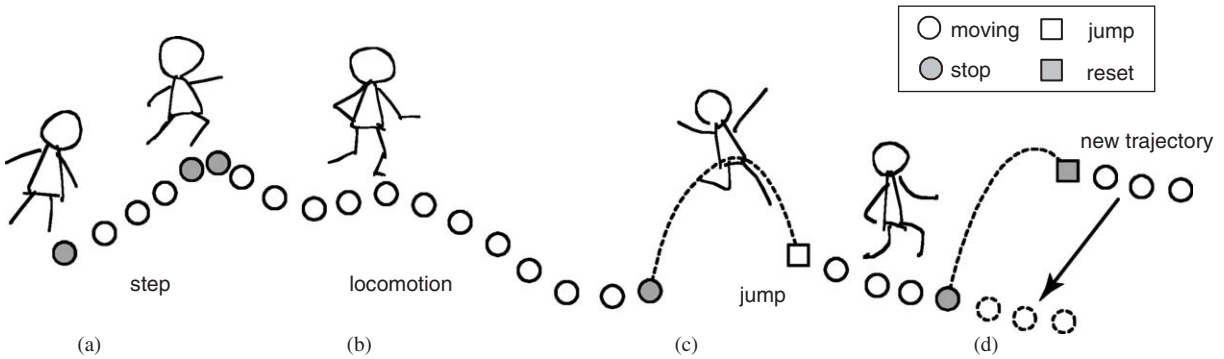

Fig. 9. An example of a series of point data that is stored in queue. Each point data are categorized into one of four types.

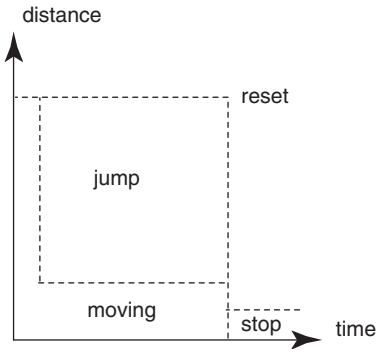

Fig. 10. Rules for categorization of point data. Based on the distance and temporal difference from the previous and next input data, each input point data is categorized into one of four types.

of executed motion but for computing motion parameters.

5.2. Starting a motion

If the pen starts moving and then stops in a short time, the pen stroke from the beginning of the series of moving data to the stop data is interpreted as a step motion (Fig. 9(a)). If the pen moves over a certain distance, locomotion or turn-and-locomotion is started (Fig. 9(b)). If a jump landing data is recorded, a vertical jump or directional jump is generated based on the distance between the landing position data and the previous stop data (Fig. 9(c)).

Locomotion is initiated after enough length of point data that are categorized as moving data are recorded. Other motions are executed after the all point data for the motion are given. We could predict the user's intension and start a motion before the entire stroke is inputted. However, we did not use such an approach for now, because it is difficult to decide the correct motion from a very short input data. In addition, the duration of input is shorter than the duration of the motion executed.

5.3. Motion parameter computation

Motion parameters are computed from a pen stroke (series of position data). All motion except locomotion starts after the entire stroke is given (stop or jump landing data is recorded). Therefore, the distance and angle parameters are computed using the difference between the first and last point data. The parameters of locomotion are computed from the difference between the current position data on the locomotion trajectory (series of position data) and a position data in a certain distance from the current data. The current position on the trajectory is updated based on the speed parameter on each frame.

The generated motion that is blended based on the computed motion parameters may not always satisfy the given trajectory or target position and direction. Therefore, we apply spatial and orientational constraints to the root (pelvis) of the figure. These constraints are computed from the input trajectory. During locomotion, the horizontal position and horizontal orientation of the root is always constrained so that the root moves above a locomotion trajectory. The figure orientation on each point of the pen trajectory is computed from tangent vector of the trajectory. In other motions, only the terminal position and direction is constrained. The weight of constraints becomes large when the motion reaches the end.

Other locomotion parameters such as speed, duck, bend-stretch, tilt are computed by averaging the recorded values of point data nearby the current position on the trajectory. As explained in Section 4.1, a long locomotion is generated by repeating a motion of one locomotion cycle. However, the last locomotion step does not always finish at the end of trajectory. In such case, the figure must keep walking motion without moving forward and this looks like an unnatural

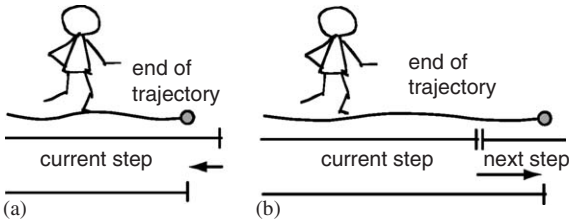

Fig. 11. (a), (b) Control of the last step of locomotion so that a locomotion step finishes at the end of the trajectory.

animation. Therefore, we adjust the locomotion speed parameter during the terminal part of locomotion so that the last locomotion step stops just at the end of the trajectory. The distance of the current step is computed by multiplying the speed parameter and the motion duration. If the speed parameter is changed, motion duration also changes since the blending weights are recomputed to satisfy the changed parameters. Therefore, the relationship between speed parameter and distance of step is nonlinear. However, we can assume that the more the locomotion speed is increased, the longer the step distance is extended. Using this assumption, we repeat adjusting speed parameter until the step distance fits to the desired length. We apply two rules here. First, if the current step is going to exceed the end of the trajectory (Fig. 11(a)), the speed is reduced to fit the step distance to the rest of trajectory. Second, if the next step is going to be very short and we cannot handle it even if the minimum speed is applied (Fig. 11(b)), the speed of the current step is increased so that it finishes at the end of the trajectory and the next step becomes unnecessary. As explained in Section 3.3.1, if the figure catches up the current pen position but the trajectory is not finished yet, we simply reduce the speed parameters.

6. Motion generation

As discussed in Section 2.2, we decided to take a global motion blending approach with small modification. When a set of parameters is given, weights of all example motions are determined first. By blending example motions based on the weights, a desired motion is then generated. In addition to this motion blending, we use motion transition and constraints methods for generating continuous motions.

6.1. Motion blending

Our implementation is based on the algorithm by Park et al. [21] and Rose et al. [22]. A typical method for weigh computation is to combine linear coefficients and nonlinear coefficients for a radial basis function.

$$\mathbf{w} = \mathbf{Ap} + \mathbf{BR(p)}, \tag{1}$$

where $\mathbf{w}$ is the n-dimensional vector that represents the weights for example motions and n is the total number of them; $\mathbf{p}$ the m-dimensional parameters and m is the dimension of the parameter space. First term is for blending example motions linearly with coefficients matrix $\mathbf{A}$ and the second term is for reducing error using a radial basis function.

Ignoring the second term of Eq. (1) and considering the conditions that $\mathbf{w}_i = 1$ is satisfied when $\mathbf{p} = \mathbf{p}_i$ where $\mathbf{p}_i$ is the parameter vector of ith motion, each row of $\boldsymbol{A}$ is computed using a least-square method.

A radial basis function $\mathbf{R(p)}$ is used for the second term. The ith column of $\mathbf{R(p)}$ is computed using a faction and distance between $|\mathbf{p} - \mathbf{p}_i|$. For a radial basis function, we use the cubic B-spline function in the same way as Park et al. [21]. Using the same conditions for the first term, each row of the radial basis coefficients matrix $\mathbf{B}$ to reduce the error between the actual weights and computed weights calculated using the linear term is determined by solving the linear problem:

$$\mathbf{BR}(\mathbf{p}_i) = \mathbf{w} - \mathbf{Ap}_i \tag{2}$$

Using Eq. (1) and coefficient matrices $\mathbf{A}$ and $\mathbf{B}$, weights of example motions are determined based on the given parameters.

Once the weights are determined, a blended motion is computed by timewarping the example motions based on the sequential keyframes that are given in advance. For smooth blending we employ an incremental timewarping and sphere-vector space orientation blending technique that are presented by Park et. al. [21].

The major differences between the original method [21] and our method are as follows. We use an incremental technique also for the horizontal position of the root instead of simply blending the positions of example motions, because blending of long step motion and short step motion may causes reverse movements of the root position. Therefore, we blend the velocities of example motion instead of positions of example motions. The height of the root is computed based on the example motions and the ground height. In addition to this, we use the initial posture of the following motion as $(n+1)$th example for motion blending in order to realize smooth transition as discussed next (Fig. 12).

6.2. Motion transition

We need to make a transition from one motion to the next motion. This includes repeating the same motion,

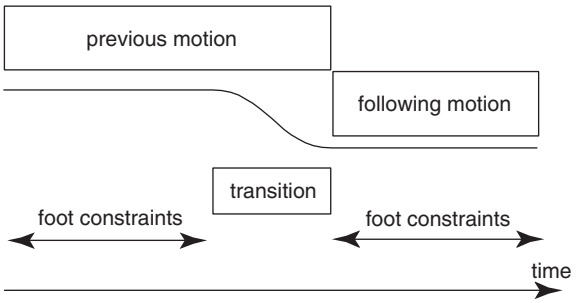

Fig. 12. Motion transition and foot constraints. The initial posture of the following motion is blended with the previous motion as the $(n+1)$th example motion.

such as locomotion cycle. For smooth transitions, we simply blend the terminal segment of the previous motion and the initial posture of the following motion (Fig. 12). We employ this simple technique since the types of motions in our system are limited and the differences between motions are small. Although there is a more complete approach that we prepare all transition motion data from one motion to another in advance [26], it is difficult to be adopted in our system since we use many types of motions and the combination is quite large.

6.3. Root and foot constraints

During a motion root constraints are given from the interface module based on an input pen stroke or trajectory. The constraints include horizontal spatial constraints and horizontal directional constraints. These constraints are directory applied to the root position and rotation.

The motion blending scheme does not ensure a foot position is always fixed while the foot is contacting the ground. In addition, the root constraints also change the foot positions. To avoid these problems, the constraints between the figure feet and the ground are also applied. We introduced a kind of importance-based end-effectors constraints [27]. Once a foot contacts the ground, a filter tries to keep it on the same position on the ground. The leg postures are computed based on the filtered end-effectors position using an analytical inverse kinematics. However, we do not apply the constraints during a motion transition phase, because the following motions sometimes become unnatural when two transition postures are different and the initial posture of the following motion is not established because of the constraints. Although our system currently generates motions on an even terrain, it is easy to retarget them on a curved terrain by controlling the heights of the foot constraints.

7. User experiments and discussion

We have implemented the proposed interface as a Windows application with MS Visual C++ and Wacom Tablet Library [25]. The program works on over 60 fps on a standard computer. Fig. 13 shows an example animation. In the animation, the user was making the figure walk along the path in many styles. The camera is automatically moved along the figure so that the figure was displayed on the center of the view. While the use is drawing a motion trajectory, if the pen trajectory that the user is drawing reaches the edge of screen, the camera moves along the end of the trajectory so that the end of trajectory remains inside the screen. The user needs to see the trajectory rather than the animating figure. The orientation of the camera is fixed in order to keep the relationship between the pen direction and figure direction constant.

7.1. A gamepad-based interface for comparison

We also developed a gamepad-based interface based on a velocity-based control for comparison with our pen-based interface (Fig. 14). This interface uses the same motion generation engine with the pen-based interface. We used a gamepad for PlayStation 2 in our experiments with a USB adopter for PC. The gamepad has two sticks (4 DOFs) and 14 buttons. The user can control locomotion using the left stick. The speed and orientation of locomotion are directly computed from the tilt of the left stick. In addition, by combining with a button with the left stick, the user can initiate a step and jump motion. The orientation and distance of a step or jump are computed from the tilt of the right stick. The right stick is for posture control of the figure. There is also a duck button. The duck height is computed based on the duration that the user keeps pressing the button.

7.2. Experiments

To evaluate the controllability of the pen-based interface and to compare it with a standard gamepad-based interface, we developed a kind of game application. Five undergraduate students participated in our experiments. The subjects were requested to control the figure to the goal (blue tile) without going over the given course (cyan tiles) or be hit by moving obstacles. The obstacles moved between two fixed points in a fixed speed. We designed four stages to test various control abilities (Fig. 15).

First, we explained the subjects how to use the pen and gamepad interface and then let them try one of two interfaces freely on the stages without obstacles. After about 15 min training, the subjects tried each stage three times and the scores are recorded. A score record includes the goal time, times of course out,

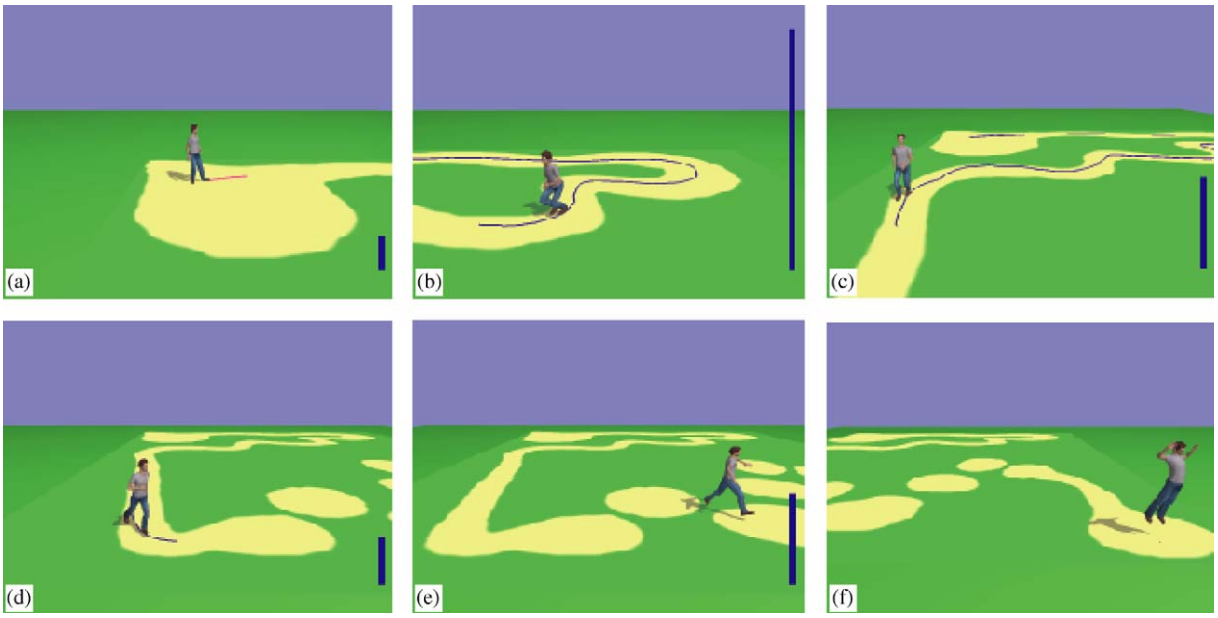

Fig. 13. Images from an example animation.

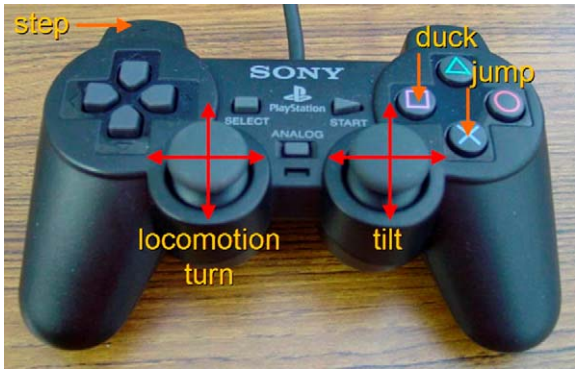

Fig. 14. The gamepad-based interface with a gamepad for PlayStation 2.

and times of collision with an obstacle. Then the subjects switched to the other interface, and did the same thing with another 15 min training. After that, the subjects tried each stage with obstacles after about 10 min training for each interface. Finally, we summarized the average scores for each stage and each interface from all the records.

7.3. Results and discussion

The result of the experiments is shown in Table 3. It leads us to the flowing discussion.

- The number of course out times is smaller with the pen-based interface in all the stages. This result strongly supports our expectation that the pen-based interface is suitable for precise locomotion trajectory.
- The goal time is shorter with the pen-based interface in the stages that have narrow paths while it is longer in the stages that have wide paths. From an observation of the experiments, we found that the users do not tend to control speed with the gamepad-based interface. Instead, they just use the maximum speed or the minimum speed. They basically have been trying to run through a narrow path at a maximum speed. As a result, they reach the goal in a shorter time, although the number of course out times increases. On the other hand, the users move their pens slowly even involuntarily in a curved or narrow path with the pen-based interface. As a result, locomotion speed naturally changes in response to the path and the control of the users.
- The number of collision times with obstacles is basically smaller with the gamepad-based interface. The reason considered is that the delay before a desired jump or step motion is executed is smaller in the gamepad-based interface. The problem seemed to be the duration of the input time of a jump stroke rather than the duration from when the stroke was inputted to when the motion was executed. To initiate a jump motion the user has to move the pen within a certain distance and timing. Because of this, it seems to be difficult for the user to execute a motion on an appropriate timing to avoid moving obstacles. We think the controllability will be improved by allowing more margins for motion input.

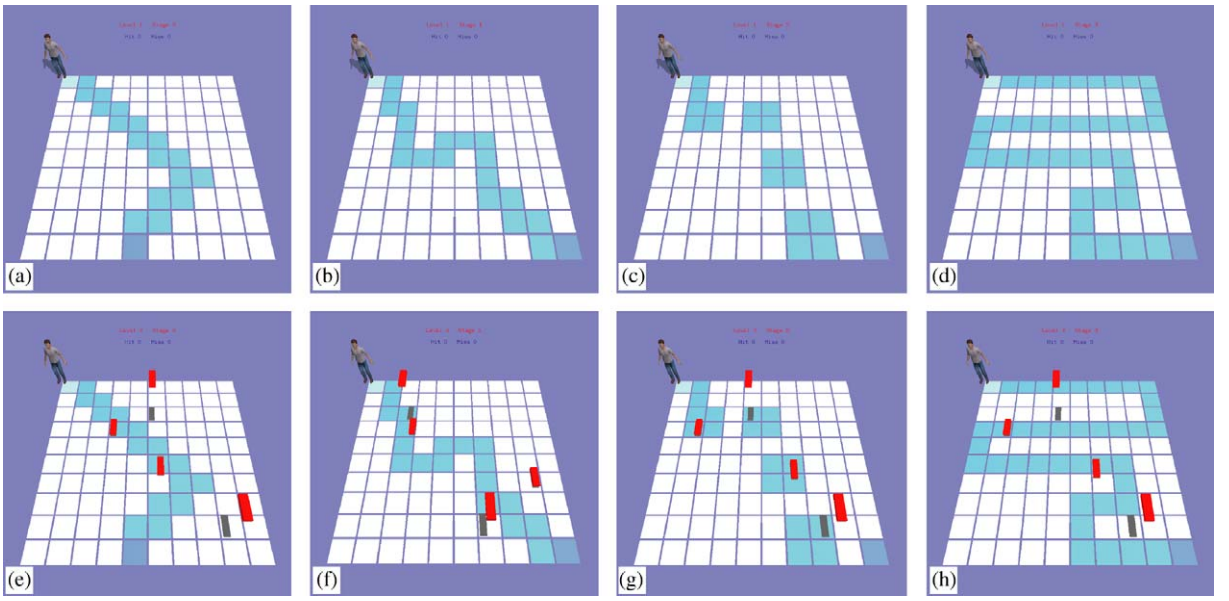

Fig. 15. The stages used in our experiments. The subjects are requested to control the figure to the goal.

Table 3
Results of the experiments

Stage	Pen			Gamepad		
	# Course out	# Collision	Goal time	# Course out	# Collision	Goal time
Stage 1 (no obstacle)	0.50	—	5.60	0.89	—	3.68
Stage 2 (no obstacle)	0.75	—	7.94	1.33	—	8.87
Stage 3 (no obstacle)	2.25	—	23.06	2.42	—	19.17
Stage 4 (no obstacle)	0.33	—	11.93	2.00	—	11.93
Stage 1 (with obstacles)	0.50	1.08	6.85	0.67	0.83	6.14
Stage 2 (with obstacles)	0.67	0.67	10.00	1.42	0.92	9.68
Stage 3 (with obstacles)	2.17	2.92	17.75	3.25	2.33	24.58
Stage 4 (with obstacles)	0.83	2.17	15.82	2.35	1.33	16.29

In summary, the pen-based interface has advantages especially when precise control is required. It also has the comparable controllability with the traditional gamepad-based interface. Further improvement of the input recognition will help the user control not only positions but also timings of motions precisely. We also could improve the controllability by introducing some kind of prediction of user input and by executing an anticipated motion before the user has finished the input. However, this may cause detection errors and unnatural resulting motion when an incorrect motion is executed and is transformed to the correct motion. There is a trade-off between controllability and natural animation.

As mentioned above, the pen-based interface also seems to be suitable for natural looking animation since even subtle changes of input are reflected on the resulting motion. Therefore, it may be useful not only for on-line applications but also in animation editing for off-line animations. However, unexpected noises of pen position also may be reflected since the figure tracks the input trajectory precisely. Moreover, a hand trajectory and a human locomotion trajectory should have different kinds of features in a detailed level. To realize more natural looking animation, we might need to introduce a kind of filter that converts an input trajectory into a natural locomotion trajectory based on some knowledge of locomotion trajectory [28].

8. Conclusion

In this paper we presented a pen-based interface for interactive motion control. We introduce a position-based pen interface which also has controllability and interactivity. The user can control locomotion speeds and styles during motion in addition to the precise control of motion trajectory and positions. We think that our interface is also suitable for novice users and kids. We are going to experiment the pen-based interface with more various types of people. Pen is a very common tool in our life and we think they have potential for an interactive motion control interface.

In addition to the direct control based on the pen movements, more intelligent movements such as avoiding obstacles or working with other character will be required on some applications. We also intend to develop reaction models that generate believable reactions based on the environments in addition to input from the pen. For example, a dance application in which two figures dance cooperatively guided by two separate pens that are manipulated by two users is an interesting application.

Acknowledgments

The author would like to thank Naoko Ikesumi for her help in motion capture and experiments. The author also would like to thank the participants of the 1st Eurographics Sketch-Based Interfaces and Modeling 2004, who gave a lot of thoughtful comments for the earlier version of this work. This research was partially supported by a Grant-in-Aid for Scientific Research (16650022) from the Ministry of Education, Culture, Sports, Science and Technology.

References

[1] Tachi S. Telecommunication, teleimmersion and telexistence. Ohmusha: IOS Press; 2003.
[2] Perlin K, Goldberg A. Improv: a system for scripting interactive actors in virtual worlds. In: Proceedings of SIGGRAPH, 1996, p. 205–16.
[3] Oshita M. Pen-to-mime: pen based interface for interactive control of a human figure. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling, 2004, p. 43–53.
[4] Sun HC, Metaxas DN. Automating gait generation. In: Proceedings of SIGGRAPH, 2001, p. 261–70.
[5] Gleicher M. Motion path editing. In: Proceedings of the ACM SIGGRAPH symposium on interactive 3D graphics, 2001, p. 195–202.
[6] Kovar L, Gleicher M, Pighin F. Motion graphs. ACM Transactions of Graphics (Proceedings of SIGGRAPH 2002) 2002;31(3):473–82.
[7] Lee J, Chai J, Reistma P, Hodgins J, Pollard N. Interactive control of avatars animated with human motion data. ACM Transactions of Graphics (Proceedings of SIGGRAPH 2002) 2002;22(3):491–500.
[8] Thorne M, Burke D, van de Panne M. Motion doodles: an interface for sketching character motion. ACM Transactions of Graphics (Proceedings of SIGGRAPH) 2004; 23(3):2004.
[9] Oore S, Terzopoulos D, Hinton G. A desktop input device and interface for interactive 3D character animation. In: Proceedings of Graphics Interface, 2002, p. 133–40.
[10] Dontcheva M, Yngve G, Povović Z. Layered acting for character animation. ACM Transactions of Graphics (Proceedings of SIGGRAPH 2003) 2003;22(3): 409–16.
[11] Laszlo J, van de Panne M, Fiume E. Interactive control for physically based animation. In: Proceedings of SIGGRAPH, 2000, p. 201–8.
[12] Spaceball. 3Dconnexion, http://www.3dconnexion.com.
[13] Yin KK, Pai DK. FootSee: an interactive animation system. In: Proceedings of the ACM SIGGRAPH/ Eurapraphics symposium on computer animation, 2003, p. 329–38.
[14] Monkey2, Digital Image Design Inc. http://www.didi.com/www/areas/products/monkey2/.
[15] Davis J, Agrawala M, Chuang E, Povović Z, Salesin D. A sketching interface for articulated figure animation. In: Proceedings of the ACM SIGGRAPH/Eurapraphics symposium on computer animation, 2003, p. 320–8.
[16] Igarashi T. Freeform User Interfaces for Graphical Computing. In: Proc. of 3rd International Symposium on Smart Graphics, 39–48, 2003.
[17] Ramos G, Boulos M, Balakrishnan R. Pressure Widgets. ACM CHI Conference on Human Factors in Computing Systems 2004. ACM CHI Letters 2004;6(1): 487–94.
[18] Boulic R, Thalmann NM, Thanlmann D. A global human walking model with real-time kinematic personification. The Visual Computer 1990;6:344–58.
[19] Bruderlin A, Calvert TW. Goad-directed, dynamic animation of human walking. Computer Graphics (Proceedings of SIGGRAPH 1989) 1989;23(3):233–42.
[20] Laszlo J, van de Panne M, Fiume E. Limit cycle control and its application to the animation of balancing and walking. In: Proceedings of SIGGRAPH, 1996, p. 155–62.
[21] Park SI, Shin HJ, Shin SY. On-line locomotion generation based on motion blending. In: Proceedings of the ACM SIGGRAPH symposium on computer animation, 2002, p. 113–20.
[22] Rose C, Cohen MF, Bodenheimer B. Verbs and adverbs: multidimensional motion interpolation. IEEE Computer Graphics and Applications 1998;18(5):32–40.
[23] Tsumura T, Yoshizuka T, Nojirino T, Noma T. T4: a motion-capture-based goal-directed realtime responsive locomotion engine. In: Proceedings of computer animation, 2001, p. 52–60.
[24] Wiley DJ, Hahn JK. Interpolation synthesis of articulated figure motion. IEEE Computer Graphics and Applications 1999;17(6):39–45.
[25] WACOM Technology Co. http://www.wacom.com.

[26] Park SL, Shin HJ, Kim TH, Shin SY. On-line motion blending for real-time locomotion generation. Computer Animation and Virtual Worlds 2004;15(4):125–8.
[27] Shin HJ, Lee J, Gleicher M, Shin SY. Computer puppetry: an importance-based approach. ACM Transactions on Graphics 2001;20(2):67–94.
[28] Brogan DC, Johnson NL. Realistic human walking paths. In: Proceedings of the 16th international conference on computer animation and social agents (CASA 2003), 2003, p. 94–104.