3D COMPUTER PUPPETRY ON VOLUMETRIC DISPLAYS

by

Naiqi Weng

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

3D computer puppetry on volumetric displays

Naiqi Weng

Master of Science

Graduate Department of Computer Science

University of Toronto

2005

In this thesis, interactive approaches for the creation and control of character animation on a three-dimensional volumetric display are explored. Drawing inspiration from puppetry a glove is selected as an input device which has a high degree of freedom for character control. Reflective markers on the glove are tracked in realtime using optical motion capture, and are then mapped into the character's virtual space, where the hand motion and several hand gestures are interpreted to control the virtual character on the display. Two forms of animation control are addressed: authoring animation by recording short clips which are blended together and performance-based animation where the character is in perpetual motion. The techniques are tailored to novice animators and hence we work with directing characters that already have a rich repertoire of motion. In particular we illustrate our techniques using a digital marionette controlled by a real time physical simulation and a human figure with a motion repertoire represented by the motion graph of a motion capture data set.

Acknowledgements

I would like to thank my supervisor, Karan Singh, for his patient guidance, encouragement and support. I will forever be grateful for his support and help. Without him, I would never have been able to complete this degree in such a remarkable program at this outstanding university. I would also like to thank Ravin Balakrishnan, for all of his help and feedback, and for taking the time to read this thesis.

I also wish to express my thanks to everyone at the DGP lab for being so friendly and helpful. I would like to thank Joe Laszlo for his valuable suggestions concerning my research; Tovi Grossman for his help with the volumetric display; John Hancock and Daniel Vogel for their time in assisting me with the motion capture system; and Abhishek Ranjan, Alexander Kolliopoulos and Mike Wu for their kindly support and help.

I would also like to thank Alex Hertel and Philipp Hertel for their informative introduction regarding the course project; Mary Ansell for her help with proofreading the thesis; Xin Gu for her help with capturing motion data; and all of my friends, who have been so supportive during the course of my research.

Finally, I wish to thank my family, Dad, Mom and Wei, for their love and support. It is they who have made this work worthwhile.

Contents

Chapter 1

Introduction

Animation, an art form used in the entertainment industry, has a history that can be traced back to the beginning of the last century. The traditional approach of hand drawing is impractical in terms of the time and expertise required, since numerous animators with sufficient expertise need to work for extensive periods of time in order to complete a complex project. Over the years, a set of techniques was developed to overcome the limitations of this traditional art form. By separating the drawings of objects into different layers on transparent sheets, *Cell* animation allows each of them to be animated separately. In *Keyframing* techniques, leading animators define an animation sequence by means of a few key frame drawings. The remaining interpolated frames are then drawn by a team of junior artists.

In the late 1960s, computers began to revolutionize the medium and to change the way in which animations are made. With the rapid evolution of computer graphics technology, the animation creation procedure and the task of the animator have changed fundamentally. The creation of motions has been separated from the representation of the character exhibiting the motions. A detailed three-dimensional (3D) description of the sets, props and characters can be created by modelers and rendered quickly from any angle. The animators need to focus only on the problem of how to create interesting

motions.

1.1 Motivation

Nowadays, an animator typically creates an animation by directly manipulating the virtual objects or by procedural approaches, giving rise to the term *computer puppetry*.

Puppetry is a traditional performance art, where inanimate objects are brought to life by direct human manipulation. Computer puppetry has become an effective approach to creating expressive character animations. In a computer puppetry system, the movements of an animator are translated by an input device so as to provide continuous control over the motion of a computer-generated graphical character in real time. Simultaneously, the animator is provided with ongoing visual feedback from the animation on the screen as it is being created.

The real-time puppetry approach represents an improvement over the conventional keyframe approach, because animations can be done much more quickly. Moreover, animators obtain immediate feedback regarding the quality of the animation, allowing on-the-spot corrections to be made(Figure 1.1).

Whereas real-time computer puppetry is a powerful and efficient way of creating expressive animations, animators are usually experts in the puppetry field, since controlling a virtual puppet requires extensive practice and long-term experience. A friendly interface is therefore important, to allow novice computer users to learn to manipulate a virtual puppet and create interesting animations within a reasonable period of time.

Since in computer puppetry the motion information is separated from the representation of the digital objects, the digital nature of the information can be fully exploited in all dimensions to facilitate generating new motions and synthesizing motions in a offline mode. Usually, a motion clip is created by a motion capture system, where an animator's actions are captured by cameras and processed to create motion data. The motion

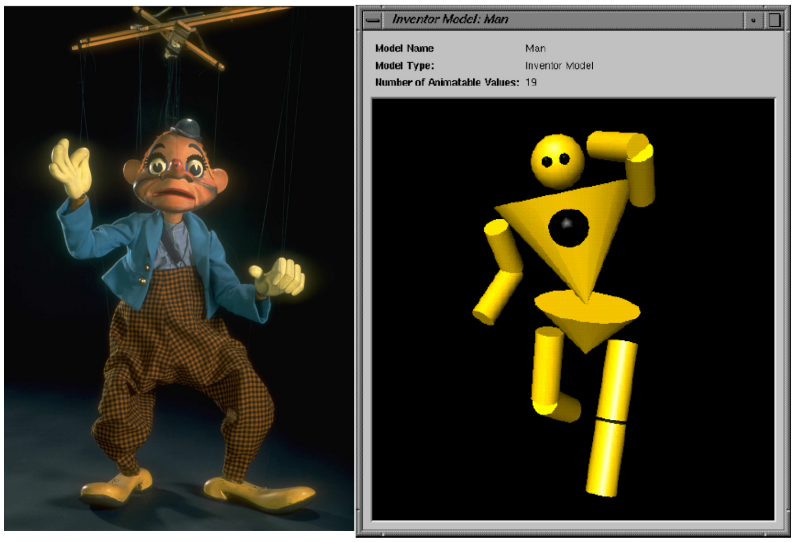


Figure 1.1: Left: marionette, one of the typical traditional puppet; Right: an articulated figure animation in a computer puppetry system(by Gildfind et al.[2])

information can then be stored, manipulated and re-used like any other piece of data. A large repertoire of motion clips thus offers the potential for creating new motions. It also provides new opportunities for enhancing the expressiveness of novice animators.

However, it should be kept in mind that the technology of motion re-use demands extensive professional skills from the animators who execute the motions before motion capture cameras. For instance, if it is desired that the virtual character shall execute a somersault, the animator must be good at performing the action accurately, in order to obtain high-quality motion data. In addition, the skeletal structure of the character must be homologous to that of the animator, since this technique does not work well with indirect mappings.

Motion graph[29] data structures are generally used in order to encapsulate information concerning motion data connections. A motion graph is a directed graph structure, where motion data are split into individual motion clips and two clips are connected if one can smoothly follow the other. Navigating a complex motion graph with a large repertoire of motion clips is also recognized as a problem, especially as the magnitude of

the graph increases.

The motivation of the present research is more closely related to traditional puppetry in the following aspects. In a traditional puppetry setting, the improvisation space of professional puppeteers is restricted to their hands. Since puppeteers have only two hands, they cannot use both hands to manipulate the strings and do something else at the same time. The virtual puppetry scenario has the advantage of providing more improvisation space and new possibilities for animations, since it is more flexible and is not restricted to the constraints of a real puppet. For instance, in the case of a virtual puppet, the weight of the puppet can be changed dynamically, and the strings can be detached during a performance. Virtual puppetry is also more suitable for novice puppeteers.

In conventional computer puppetry systems, although the animation data may be inherently 3D, users typically view it on a two-dimensional screen incapable of explicitly displaying the depth dimension. The lack of depth information leads to many drawbacks and limitations. For example, an animator may find it difficult to locate precisely a joint to be manipulated, or may estimate the length of an object incorrectly, due to its angle of orientation with respect to the camera. Also if two virtual objects do not overlap from the animator's viewpoint, information such as whether one is closer to the animator than the other is usually lacking. In order to solve this problem, the camera is often dollied back and forth so as to yield clearer depth dimension information. However, this greatly hinders the animator's interaction process. The user experience would be considerably enhanced by the ability to observe the 3D data directly. The volumetric display used in the present research generates 3D presentations of animation data. In addition to improving users' perceptions of the 3D data, it also provides a more immersive interaction environment. Unlike traditional 2D mouse/keystroke control interfaces, where users see the animation data on the screen and execute control actions in front of the screen, the volumetric display enables users to carry out control tasks within the same

framework as the animation, thus providing a more powerful 3D interaction space. The present research aims to explore 3D interaction techniques with a volumetric display, so as to improve the animator's experience in a computer puppetry system, thus developing an efficient, user-friendly 3D computer puppetry animation system for novice users.

1.2 Contributions

In this thesis, interactive approaches for the creation and control of character animation on a 3D volumetric display are explored. A 3D interaction interface and prototype computer puppetry system were constructed, where users wearing a specially designed glove can interactively create smooth 3D character animation on the volumetric display in real time. Two forms of animation control are addressed: a) authoring animation, where short clips are recorded and blended together, and b) performance-based animation, where the character is in perpetual motion. The animation control approaches were implemented in two applications. The first was a physically-based virtual marionette system, where users can manipulate the movement of a virtual marionette by controlling the strings connecting it to a control stick. Users can also detach a string from the control stick and reattach it to a point in the 3D virtual space. This provides greater improvisation space for animators and abundant real-time 3D interaction primitives for controlling the puppet. In the second application, a human figure with a motion repertoire represented by a motion graph was animated. A mechanism was developed where the hand actions and movements of the user directly guide the traversing of a motion graph. Commands are effected by simple gesture inputs. This mechanism permits the user to create a sequence of realistic character animations in real-time with ease.

The novelty of this work lies in the consideration of how to apply 3D interaction techniques to 3D animation, and in the investigation of the type of control mechanisms that should be employed in this 3D animation scenario, taking advantage of the new

3D volumetric display technology now available. Many investigations have been made of the control schemes and primitives[11],[16],[27],[40],[42],[53] in computer puppetry, but they do not take into account the influencing factor of the display of depth dimension information in data viewing. New interaction styles specifically suited to this volumetric display, as a comparatively new device, are investigated in [18]. In the present research, some of these techniques are used to experiment with computer animations, so as to implement an efficient, friendly interface for computer puppetry animation systems.

Another interesting idea implemented in the present research is to combine the traversing of motion graphs with real-time direct 3D user hand actions. In the present study, using the 3D animation on the volumetric display, a basic mechanism was designed so that during the animation in the current node, user hand input determines the selection criterion and the guiding vector for the next node. The system is then responsible for finding the next suitable node, the animation of which will follow that of the current node. The details of this process are discussed in later chapters.

1.3 Thesis Outline

The remainder of the thesis is organized as follows. Chapter 2 presents a more detailed background in computer puppetry, interactive character animation and 3D interaction technology, which are all closely related to the present research. Chapter 3 describes the details of the system settings for the prototype. The design framework of the idea of computer puppetry on volumetric display is explored in Chapter 4. The detailed interactive techniques applied and corresponding results in two examples(virtual physical marionette and motion graph exploration) are presented in Chapters 5 and 6, respectively. In Chapter 7, conclusions and future work are discussed.

Chapter 2

Background and Related Work

The present research is related to a number of fields, including computer puppetry, interactive control of computer animation, and 3D interaction technologies in the context of human-computer interaction. In considering computer puppetry within the more general context of animation tools and techniques, this chapter first presents a brief overview of other common animation techniques, and then focuses on the literature more specifically related to the present study.

2.1 Keyframing

A basic approach used in many computer animation systems is *keyframing*, where the position of the animated character is specified for a few discrete key frames, and the "in between" positions are then computed by interpolation. This is a natural extension of the most conventional and flexible of animation methods: drawing each frame by hand. In a keyframing computer animation system, the animator provides a sequence of specific image frames (key frames) together with the corresponding times at which they must occur, and the computer generates the intermediate frames by interpolation. Although tedious, this is still the method most commonly used to generate computer animation.

2.2 Motion Capture

Motion capture systems use various techniques to record an actor's movement, and construct a 3D representation of the actor's pose and position. In a typical optical motion capture system setting, the actor wears special reflective markers, the positions of which are captured by an array of cameras with high refresh rate. The cameras collect the marker location information and the associated software is then used to map the reconstructed marker positions onto an underlying skeleton model. This method can be used online to produce real-time animations, or can be employed for offline production, where the captured motion data is retrieved and modified for later use.

Considerable attention has been focused on the use of motion capture data to create realistic animation sequences. Most of the research investigates how motion capture data can be segmented and used to synthesize new motion. This method avoids not only the numerical complexity of simulating and controlling physics-based systems but also the tediousness of the traditional keyframing approach. Kovar and his colleagues describe how motion capture data can be managed by automatically constructing a directed graph, referred to as a *motion graph*[29]. A motion graph consists of clips of the original motion, together with automatically generated transitions between the motion clips. Realistic motion can thus be synthesized by finding a path on the graph that meets the user's specifications. In [31], a technique of enhancing keyframed animation with motion capture data is described. The technique is comprised of two processes: texturing and synthesizing. Texturing adds detail to the degrees of freedom (DOF) that have been keyframed, while synthesizing creates motions for degrees of freedom that were not keyframed.

Lee et al. [33] show that an extended unlabeled sequence of motion data can be preprocessed in order to obtain flexible behavior, efficient searching and the possibility of real-time avatar control. Flexibility is created by identifying plausible transitions between motion segments. An efficient search through the resulting graph structure is obtained by

means of clustering. Three interface techniques are demonstrated for controlling avatar motion using this data structure: the user selects from a set of available choices, sketches a path through an environment, or acts out a desired motion in front of a video camera.

With this motion data technique, as the quantity of motion data increases, two major research problems emerge: the automatic segmentation of the raw human motion data into distinct behaviors and actions, and the effective navigation of a complex motion graph with a large repertoire of motion clips. The present research explores the second problem, by focusing on how to synthesize interesting motion sequences by effectively navigating motion graphs. The problem is addressed by introducing real-time direct 3D user hand actions. A more detailed discussion is provided in later chapters.

2.3 Computer Puppetry

2.3.1 Brief History of Computer Puppetry Applications

Computer puppetry has become an important emerging area in computer animation. Advances in input technology and graphics hardware have enabled a new style of animation, where a puppeteer uses a range of specialized input devices to control the motions of an animated character in real time. The history of this technique can be traced back to the early 1960s.

Sturman's [25] survey presents a brief history of computer puppetry and describes some of the technical challenges that must overcome in this field. He points out that one of the first real-time interactive graphics systems was the SCANIMATE system, developed by Lee Harrison III [51]. It was used to create almost all of the flying television logos from the late 1960s to the early 1980s. The logos were generated by a human performer wearing a data suit made of potentiometers and Tinkertoy. Although it had the advantage of real-time control, this approach eventually became outmoded since it could not compete with the slicker imagery produced by using keyframing.

The first modern computer puppet, "Mike the Talking Head", [6, 43] which performed for a live audience at the Siggraph 89 electronic theater, was driven by a special controller built by deGraf/Wharman. A single puppeteer controlled all the head parameters including facial expressions, lip synching and head rotations, and demonstrated a live lip-synch to an operatic solo.

In the same year, another computer puppet, "Waldo C. Graphic" [48] was created by Pacific Data Images (PDI) for production television. Waldo C. Graphic was controlled by a puppeteer using a customized mechanical arm with an upper and lower jaw attachment. The puppet animation was first rendered in low resolution. The puppeteer received feedback from a video monitor display where Waldo C. Graphic was overlaid by other real puppets which were performing and being taped at the same time. The animation was subsequently fully rendered and a composite made with the video source.

Medialab, Paris, produced a computer puppet, "Matt the Ghost," in 1991. Matt appeared daily on French cable TV for almost four years. An entire week's worth of animation (five animation minutes) was generated by a team of three puppeteers a single afternoon session. Medialab also produced "Donkey Kong Country", an animated production with 11 characters, which had 26 episodes totalling 9,000,000 images. The monkey character in this production was animated by two puppeteers [7]. One puppeteer controlled the facial animation and expressions, while the other, wearing 16 sensors on his limbs and joints, controlled the movement of the character's body. In this production, 70 percent of the animation was done by computer puppetry and 30 percent by keyframing. The fact that the time spent on each was the same demonstrates the obvious advantage of computer puppetry over traditional animation techniques.

The Protozoa company also introduced interesting and creative real-time computer puppetry. This company developed a "genetically mutated lizard cow" called "Cracker" [38]. Five electromagnetic trackers were placed on an actor's arms, feet and pelvis, ignoring the movement of other body parts. Specific mapping was employed to control

the legs, head and spine movements of the mutated lizard cow.

2.3.2 Interactive Character Animation

Computer puppetry combines the techniques of computer animation and interactive control. In addition to the commercial projects mentioned above, research-oriented puppetry systems have recently emerged, as described below.

Wilson [50] introduces a real-time performance animation system which extracts the 2D position and orientation of a puppeteer's hand from a video stream and maps this onto the head of a Luxo lamp in order to control its movements.

Modern animation systems provide many powerful tools for synthesizing complex motions. Three primary types of control mode are used in these animation systems [53]: guiding level real-time control, task level control and animator level control. The first is a low-level control and the others are high-level controls. Each control level involves different tradeoffs between automation and expressiveness. In guiding level control, every detail of the motion must be specified by the animator. Although this permits complete expressiveness, it requires a high level of expertise and extensive practical experience on the part of the animator. In addition, as model complexity increases, the volume of information which must be specified becomes overwhelming. In both task level and animator level control modes, programming tools and abstraction mechanisms are employed to generate motions which satisfy the animator's requirements. While high-level control modes are good at producing a high degree of motion fidelity, the level of abstraction restricts the animator's ability to exert fine control over the motions. Since computer puppetry is a guiding level control technique, the following discussion will focus on literature pertaining to this level of control. The more general issues associated with higher-level control, including procedural, scripted and autonomous character animation, are beyond the scope of the present study and will not be reviewed here.

In [2], Gildfind et. al introduce an interesting approach in which genetic algorithms

[42] are used to construct efficient low-level control systems for arbitrary combinations of input devices and classes of animated models. Instead of directly specifying the details of a control system, a search-based approach is used. Genetic programming is employed, and a fitness metric derived from subjective user feedback is used to optimize a control system to suit individual puppeteer preferences and dexterity.

For low-level real-time control, higher-level parameterization is necessary when the input has fewer dimensions than the output. For instance, a character posture output space may have M dimensions, while the input space has $N < M$ dimensions. One natural approach to the mapping is to find a parameterization of the posture state with fewer dimensions that can span the desired range of animation. This means collapsing some degrees of freedom (e.g. establishing certain interdependencies among the joint angles) so that entire sequences of joint angles over time can be represented by only a few parameters. Examples with different levels of parameterization are described below.

Donald and Henle [11] describe a haptic force feedback device interface for animation control (referred to as the "Phantom"), where trajectories in a 6-DOF input space are mapped to trajectories in the animation space. The user can modify the output trajectory by pulling or pushing the handle of the device. A key feature of this force feedback device is that the haptic vector field depends not only on the position, but also on the velocity of the input device. These velocities are used to determine the haptic force received by the user to assist in navigation when two trajectories cross. This system is suitable for a few input motions and may not scale well as the desired range of output motions increases.

Perlin and Goldberg [40] present the "Improv" system, a real-time interactive framework that allows animators of various abilities to create remarkably lifelike, responsively animated character interactions. The system combines low-level and high-level control. Its development is based on the real-time responsive animated characters introduced in [30]. In [30], Perlin creates basic motions for an articulated character by specifying the maximum and minimum values for each joint parameter, together with an associated

function for interpolating between these two extremes. Once a set of basic motions has been defined, each basic motion is assigned a scalar weighting parameter. This defines its relative contribution to the blending process which converts discrete actions into smooth, continuous transitions over time. In addition, a layered approach can be used to generate complex natural motions by combining basic actions such as "scratching head" while "walking". Keystrokes or mouse clicks constitute the input interface that allows users to trigger basic motions. The "Improv" system [40] generalizes the previous framework for layering and composing motions and provides a system for scripting multiple interactive and autonomous characters.

Another fundamental issue in computer puppetry is the remapping of motions from one character to another in the case of characters which have the same skeleton structure but different limb lengths. Gleicher's paper [17] is one of the first to address this issue. A non-linear optimization problem is set up, with key features of the input motion such as footfalls or interactions with external objects serving as the constraints. The solution of the problem is the new motion. Shin et al. [24] present a comprehensive solution to the problem of adapting motion capture data to a virtual character which differs from the performer in terms of size and proportions. These researchers use the concept of the dynamic importance of an end-effector. This serves as the criterion determining which aspects of the performance must be retained in the resulting motion. The system operates in real time.

2.3.3 Dynamics Simulation with Interactive Control

Dynamics simulation has been successful in generating realistic motion for various types of physical systems, including rigid body animation, passive systems such as hair and cloth, and natural phenomena such as water and smoke. In this approach, the classic laws of physics and the corresponding formulae are used to compute accelerations according to the forces acting upon a system for which the masses are known. Issues such as

motion smoothness and maintaining proper ground contact are not a concern since these inherently obey the physical laws. The Siggraph 1997 course notes package prepared by Baraff and Witkin [4] gives a good general introduction to the concepts underlying numerical simulation. These authors first review differential equations, and then deal with rigid body dynamics and constraint dynamics, two critical aspects of physical character animation.

Although animation generated by dynamics simulation is realistic, it exhibits several inevitable limitations. A primary limitation results from the computational demands arising from small timesteps or from the computational complexity of each step. In addition, it is difficult to define appropriate control functions to generate a particular end-effector trajectory, so as to achieve a desired motion. Extensive research has been focused on how to design and incorporate appropriate control mechanisms into dynamics simulations in order to produce natural but controllable computer animations. This aspect is examined below.

Troy's work [27] is one of the earliest examples of interactive control of the locomotion of a dynamic simulation of a graphical biped character. The user operates a 7-link planar biped model with a feedback-based 2D balancing regulator and a state machine proportional derivative (PD) controller for a walking motion in a virtual environment.

Laszlo, van de Panne and Fiume [16] have demonstrated a well-designed system for the interactive control of 2D physically-based animated characters. Various combinations of continuous and discrete input actions provided by a standard mouse and keyboard permit the creation of a broad range of motions for three characters: a Luxo desk lamp, a bounding cat and a human character. For example, continuous input of mouse horizontal and vertical translations are mapped to the two actuated joints of a Luxo lamp, driving the lamp by means of PD controllers. A set of 6 keys corresponding to 6 different desired states is assigned to the front and back legs of a planar bounding cat. The user can easily create smooth animation sequences by using the appropriate keystrokes to manipulate

the legs of the cat. Continuous and discrete control methods are combined to control a biped human character capable of motions such as walking, running and long jumping.

Oore [39] presents a novel computer puppetry system which enables a user to control complex human motion using two 6-DOF Polhemus input devices. Local physical models are introduced and developed to augment the existing kinematic mapping in order to facilitate the animation task and enhance the naturalness of the motion. This system thus integrates kinematics and dynamic principles in order to achieve more effective real-time animation control.

Another example of combining controllers with physics-based simulation is found in the work of Yang, Laszlo and Singh [52]. These authors propose a layered strategy for controlling a physically simulated human swimmer in a dynamically simulated fluid environment. A further example is the work of Hodgins et al. [28], which deals with animating human athletics. Specific control algorithms are designed and developed for three dynamic athletic behaviors: running, bicycling and vaulting. Faloutsos et al. [14] present a framework of composing controllers which provides a broader repertoire of lifelike motor skills for a dynamic anthropomorphic figure. This idea has been further explored in [15]. Shapiro et al. [45] combine kinematic animation with physical simulation. A supervisory controller is implemented, and a transition between the two animation control methods can be effected depending upon the circumstances in the scene. In the work by Zordan and Hodgins [10], the problem of motion synthesis for interactive, humanlike characters is addressed by combining dynamics simulation and human motion capture data. Their control system uses trajectory tracking to follow motion capture data and a balance controller to keep the character upright. The control is loosened to allow dynamics to exert a greater influence on overall motion in such situations as a boxer being punched.

In [32], Laszlo, Neff and Singh propose a new technique, referred to as "predictive feedback", where a prediction of the near future for a few sample inputs is continuously presented to the animator. A visual presentation of the predictive feedback is illustrated:

Control input samples are selected in the vicinity of the user's current input, and the predicted results are co-located with the position of the input necessary to achieve them. These researchers illustrate how the technique contributes to a user's interactive control of the dynamically simulated motion of a planar character, using mouse and keyboard input. A similar implementation was developed in our prototype system, which will be discussed in later chapters.

Igarashi et al. [47] introduce spatial keyframing techniques for performance-driven animation, where key poses are defined at specific key 3D positions in the same space as that of the character. Mapping from the 3D position space and the character's configuration space is defined by interpolation. The user controls the character by adjusting the position of a control cursor in the 3D space. The pose of the character is generated as a blend of nearby key poses. The authors also present several applications and examples demonstrating the expressiveness of this approach.

2.4 3D Interaction Technology

Over the past 20 years or so, a large number of 3D systems has been developed. Around the middle of the 1980s it became increasingly common to experiment with interactive 3D systems, resulting in numerous reports on the use of new techniques and input devices. Chris Hand [20] provides an informative overview of the interaction techniques for object manipulation, navigation and application control in a 3D virtual environment. Direct manipulation of graphical objects in a 3D virtual environment is a typical task that must be performed. It involves actions such as selecting, scaling, rotating and translating.

An obvious method of object selection is to use the position of a 3D cursor [22, 36, 37, 41] in the virtual environment. The main alternative to using a point cursor is the use of a ray-casting selection cursor [34], where a virtual ray originates from the user's hand position. The user controls the starting point and orientation of the ray. Typically,

the first object it intersects is selected.

In many virtual environments, (e.g. [34]), objects are manipulated using a 6-DOF tracker. In this approach, the tracker movement is directly mapped to the position and orientation of virtual objects. Direct gestural interactions have also been used [36], where hand movements are mapped directly to object movement. The techniques of ray-casting selection and subsequent direct manipulation are combined in the work of HOMER [9]. Charade [5] describes the use of freehand gestures to manipulate 2D virtual objects in an augmented reality system.

Since the present research primarily utilizes this type of interaction technique, the following review will focus mainly on object manipulation. The introduction of true 3D input devices has led to many investigations of the possibilities of this new technology.

2.4.1 Polhemus Device

Early work at the Massachusetts Institute of Technology (MIT) such as gesture- and speech-based "put-that-there" studies [8] used a Polhemus electromagnetic 6-DOF tracker. The Polhemus device became widely used due to its small size, ease of use and ability to measure both positions and orientations. For example, a Polhemus device was used to track the position and orientation of the hand in the "VPL Dataglove" [19] project. "3-Draw" [44] was another MIT project, in which a clipboard and stylus were tracked by two Polhemus devices to create 3D curves and thus 3D shapes, either by free-form drawing or by using constraints. Galyean and Hughes [1] developed a technique referred to as "sculpting", which permitted direct manipulation of a volumetric representation by chiselling away parts of the volume with a Polhemus-based tool.

2.4.2 Glove-based Input

The introduction of instrumented gloves was significant, as this technology enables computers to "read" users' hands directly, thus removing the limitations of intermediary

devices such as keyboards, mice and joysticks. A virtual hand controlled by a glove rather than by a cursor has the potential to be very direct, expressive and natural, since the user can apply more manual dexterity to the interaction tasks.

In their survey of this technique, Sturman and Zeltzer [26] discuss many of the hand-tracking technologies and applications which use glove-based input. The following discussion refers to findings relevant to the present study.

In the VPL Dataglove [19] project developed by Thomas Zimmerman and others, a lightweight cotton glove was used to monitor 10 finger joints and information relating to the position and orientation of the hand in real time. In this system, analog flex sensors mounted on the glove measure finger bending and transmit a signal to a host computer via a small cable. A 3D magnetic tracker attached to the back of the hand determines the position and orientation of the palm. In addition, piezoceramic benders provide the wearer of the glove with tactile feedback. In the VPL system, users wearing the DataGlove can see a graphic hand that follows the motions of the hand wearing the glove. By pantomiming reaches and grabs, the user causes the graphic hand to reach and grab objects in the simulated environment. The user can also move through the virtual space by pointing in the desired direction and "flying" to the destination. The VPL system uses look-up tables to recognize finger postures representing grab and flight behaviors, in order to trigger corresponding events.

More advanced use of the glove takes advantage of the extra capabilities of the hand over a 3D joystick. As well as using the techniques described above, AT&T Bell Laboratories used a DataGlove with two additional thumb-based gesture controls, referred to as the "clutch" and "throttle" [12]. "Clutching" effected incremental transforms such as rotation. For instance, the virtual object followed the rotation of the hand only when the thumb was brought against the index finger. This allowed object manipulations to be carried out incrementally, without twisting the hand uncomfortably. "Throttling" was a variation of the clutch mechanism, where the angle of the thumb was used to scale the

effect of a hand motion.

Glove-based interaction still constitutes an under-explored area in the field of computer animation. This thesis incorporates some of the ideas in the above research in developing a prototype system.

2.4.3 Interaction with Volumetric Display

A volumetric display [13, 35] is a newly emerging display device which generates true volumetric 3D images by illuminating points in 3D space. Relatively little research has been done on how to use volumetric displays effectively, since they have not been readily available until recently. A speculative paper [3] by Balaskrishnan et al. uses mock-up prototypes as illustrations in an exploration of possible interaction scenarios for volumetric displays. Various techniques for selecting objects, displaying text and menus, and manipulating objects are also discussed.

Grossman, Wigdor and Balaskrishnan [18] investigated interaction with the volumetric display interface. These researchers designed and implemented techniques for the direct interactive manipulation of objects with a 3D volumetric display. A Vicon motion tracking system (www.vicon.com) was used to track the positions of markers on the user's finger in 3D space. An enhanced touch-sensitive display was simulated, given the marker data information in conjunction with knowledge of the precise topology and 3D spatial location of the display enclosure in the tracking volume. The system categorizes the precise positional information of the fingertips into three discrete states: down, hovering and up. By examining the relative distance between the markers, a set of static hand postures and dynamic hand gestures was defined. Two techniques were developed to facilitate command input: a surface menu, and the set of postures and gestures. For interaction in 3D space, a simple file/object management system referred to as a "SurfaceBrowser" was created to display various objects by organizing them into the cells of a 2D array. Three-dimensional object manipulations were performed by the users' hands,

by means of different finger states and hand postures and gestures. All of the above techniques were demonstrated through the development of an interactive 3D geometric model-building application.

Chapter 3

System Setup

In this chapter, the details of the system setup are described. A motion tracking system, a user input glove and a volumetric display were used. The system is illustrated in Figure 3.1.

3.1 Tracking System

A commercial Vicon motion tracking system (http://www.vicon.com) was used to track the positions of markers attached to the user's input glove. The Vicon system has high-resolution cameras to record the 3D location of multiple passive reflective markers in real 3D space, as well as a software package which can uniquely identify and label each marker according to its position on a user-defined virtual model so as to reconstruct the motions in virtual 3D space. For instance, in order to track the motion of the markers attached to a user's hand, a virtual model resembling the topology of the hand must be created to allow the system to identify and label the individual markers accurately. The reconstruction can be carried out by means of offline post-processing as well as in real time. Since the prototype for the present research required a real-time approach, a Vicon sdk was used to stream the reconstructed 3D coordinates of labeled markers to the application software in real time. In the prototype, four cameras were used for tracking.

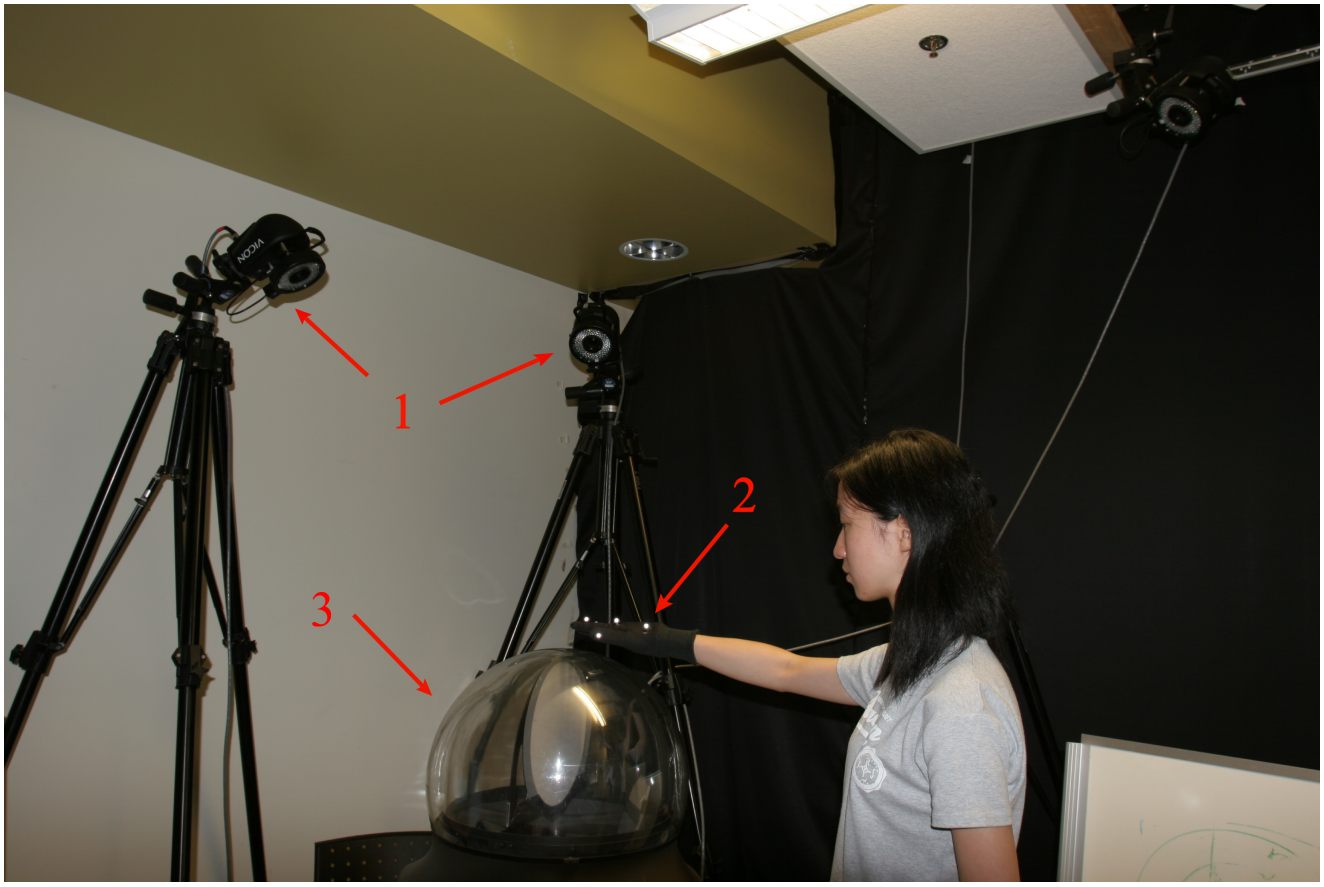


Figure 3.1: System setup: 1. Vicon motion tracking cameras 2. User input glove 3. Volumetric display

3.2 User Input Glove

In Section 2.4.2 it is pointed out that a glove-based input interface gives users additional space and flexibility for applying manual dexterity to the interaction task. In the prototype system, a light-weight fabric glove was used. Seven spherical reflective markers 5 mm in diameter were attached to the glove, as illustrated below. One marker was attached to the tip of the thumb, one to the tip of the index finger and one to the tip of the ring finger. Four additional markers were attached at the base of the index finger, at the base of the little finger, and at the left and right sides of the hand just above the wrist. It is presumed that only the right hand is used in interacting with the animation.

Figure 3.2: User input glove

The glove was selected because it is comfortable to wear and unobtrusive to the user. The fabric construction of the glove allowed it to stretch to accommodate different hand sizes. The disadvantage of the glove is that it does not provide tactile or force feedback, with the result that users are unable to feel the presence of virtual objects.

3.3 Volumetric Display

The constraints of current display technologies limit the ability of users to view and interact with 3D computer graphics. Although the data may be inherently 3D, users typically view it on a 2D system incapable of explicitly displaying the depth dimension. Volumetric display, a spatial display designed exclusively for 3D viewing applications, enables true 3D image visualization and enhances the realism of rendered 3D graphics

by providing all of the depth cues humans require. Moreover, the volumetric display allows users to view 3D scenes from almost any direction, and permits multiple users to view them simultaneously. There are various volumetric displays, which differ according to the underlying technology of the display. For the present research, a dome-shaped swept-volume display was used. When the volumetric display sweeps a periodically time-varying 2D image through a 3D spatial volume at a frequency higher than the human eye can resolve, it generates an image that the viewer perceives as a 3D volumetric image.

3.3.1 Volumetric Display Hardware Structure

A 3D volumetric display from Actuality Systems (http://www.actuality-system.com) was used. It generates a 10" spherical 3D volumetric image by sweeping a semi-transparent 2D image plane about the vertical axis (Figure 3.3). This display has a refresh rate of 24 Hz and can be viewed from any position around the hemispherical dome, without requiring the user to wear hardware. Thus, multiple users can view the imagery simultaneously, each from a different viewpoint, while maintaining the context of their shared physical surroundings.

3.3.2 Software Interface of the Volumetric Display

Like a computer monitor, the volumetric display is a display operated from a host computer. The software interface, named HMESA, provides drawing methods similar to those of the industry standard OpenGL API. Most of the OpenGL drawing methods can be used without modification, and migrating an application already written in standard OpenGL to HMESA is generally quite straightforward.

The prototypes of the methods used to create, destroy and manage rendering contexts (RCs) in HMESA are defined in the file hmesa.h. Normally, OpenGL calls are dispatched to a shared library (e.g. openGL32.dll in Windows or libGL.so in Linux). For HMESA to function properly, HMESA must be able to masquerade as this library. When the

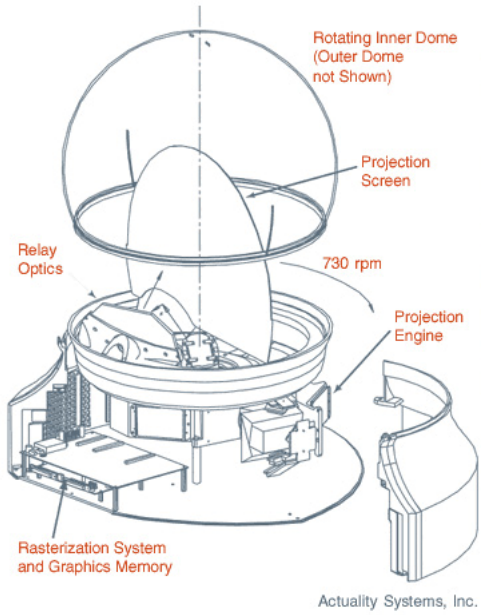


Figure 3.3: volumetric display structure

application searches for OpenGL methods, it must find the HMESA library before the OpenGL library. Therefore, adapting OpenGL applications for use with HMESA requires the following steps: adding the HMESA library to the application path, including hmesa.h in the code, creating a HMESA rendering context, and making the handle to this rendering context the current handle for rendering.

Due to some fundamental differences between flat and volumetric realizations of graphics primitives and the limitations of current volumetric rendering technology, there are several OpenGL method calls which cannot operate correctly with the volumetric display. For example, since volumetric rendering with the volumetric display is inherently view-independent, there is no perspective transformation associated with the view volume, and view-dependent aspects of the OpenGL API such as hidden surface removal and operations on the depth buffer do not function as expected. Moreover, images cannot be rendered with full opacity on the volumetric display; thus, all of the images shown on this device are wire-framed images. This will be discussed in detail later.

Chapter 4

Computer Puppetry with a Volumetric Display

4.1 Design Principle

The initial design motivation for the present study was as follows: If a motion repertoire of a character was available on a volumetric display, how could novice users affect and control the motion of the virtual character by their own actions, so as to create an animation on the volumetric display in real time?

As explained in Section 2.3.2, computer puppetry is a guiding level control technique, where the animator provides specified descriptions of the character's parameter configurations. The control tasks become difficult to accomplish when the virtual character has several joints with more than three degrees of freedom. As mentioned in Section 2.3.2, although low-level control gives animators more opportunity for expressiveness, it demands considerable proficiency and expertise. In the present research, the key issue considered in the interaction technique design was how to enable the user's hand to exert effective control over the large volume of information in the output space without increasing the complexity of use.

Direct mapping from the joints of the hand to those of the animated character seems to be ineffective, since it is difficult to devise suitable mapping from a hand joint to a body joint, if they are different types of joint. In addition, this type of mapping would introduce excessive complexity to the user interface, sometimes demanding awkward hand postures for the user to achieve control over the body joints.

In the present research, the interaction technique design inspiration is drawn from a traditional marionette, also known as a string puppet. A marionette usually consists of a puppet, a control stick, and several strings which attach the limbs of the puppet to the control stick. In a traditional marionette performance, experienced puppeteers control the movement of the puppet by manipulating the control stick and the strings from which the puppet is suspended. The puppeteer can pull the strings to lift the legs or arms of the puppet, can move the puppet head so that it appears as if it is looking around, or can even make the puppet perform somersaults such as a backflip. All of these actions are controlled by combined manipulation of the control stick and the strings. In the case of the 3D animation shown on the volumetric display, the volumetric display can be regarded as an acting stage. Thus, the concepts of a traditional marionette setting can be intuitively applied to the volumetric display by designing the interaction interface so that it is as if the puppet is suspended from an imaginary control stick, with imaginary strings attached to its limbs.

4.2 Animation Control Mode

The prototype design is used to explore two forms of animation control: authoring animation and performance-based animation. In the case of authoring animation, the animated character remains in some existing pose until interaction input is provided by the user. If the user is satisfied with the motion of the character resulting from the interaction process, the animation can be recorded. After several trials and recordings have been

made, the motions can be blended together to create an interesting motion sequence for the character. In the case of performance-based animation, the virtual character is in random motion. The user interacts with the character with the aim of causing the character to perform a specific desired motion, thus generating an interesting animation sequence.

4.3 Interaction with a Volumetric Display

As described in 3.2, a glove with several attached optical markers was used as an interaction medium for conveying the user's intentions to the animation system.

Due to the specific features of the volumetric display interface, new interaction styles specifically suited to volumetric displays were investigated in the work by Grossman et al. [18]. Most of the interactions were carried out on or around the display device. This thesis explores the basic interactions with a volumetric display (i.e. the direct touch design and gesture-based input) which may be useful for our particular control tasks and applies them on the prototype system.

In the implementation, the position information for an individual marker is recorded and stored. The state with respect to the volumetric display is determined. The state is classified as "down" if the marker is less than a certain distance from the spherical dome, and "up" if the marker is more than a certain distance from the dome. The program precisely calculates the range of the dome, with the exact topology and the 3D spatial location of the display in the tracking volume. A small set of configurations was developed based on the positions and relative location of the markers. Three gestures were used: "pinch", "release" and "tapping". The "pinch" gesture is recognized when the distance between the marker on the tip of the index finger and the marker on the tip of the thumb is less than a certain threshold value. This is tested in advance, when the user performs an assured "pinch" action. The "release" gesture is recognized when

the above distance is more than the predetermined threshold. When a previous "up" state and a current "down" state of the index fingertip marker are detected, a "tapping" gesture is recorded.

Since the target users are novice animators, complex gestures could be expected to hinder the user's intuitive understanding of the interaction process and thus impair the usability of the system. For this reason, the number of possible gestures was limited to a very small set. All of the gestures are designed to exploit rather than conflict with the natural movement ranges of the fingers of the hand, so as to avoid any awkward gestures or movements.

In addition to the set of gestures described above, a mechanism referred to as an "object selector" was also implemented, where a user's hand movement is tracked by linearly mapping its translation to the translation of a spherical 3D virtual object in the virtual scene. The object closest to this moving "object selector" is highlighted in green. By providing users with visual feedback concerning their own hand movements (e.g. with respect to position and velocity), this mechanism enables users to conveniently explore the virtual space and to select the desired virtual objects by gestures.

4.4 Design Framework

Following the development of a marionette concept with an imaginary stick and strings, and the definition of the basic interactions with the volumetric display, two animation modes were implemented. The basic framework will first be described in general, and then the detailed design for the two modes will be presented.

4.4.1 Manipulation with a Control Stick

In the case of a real marionette, the puppeteer usually moves the control stick in order to move the whole body of the puppet. In the case of the virtual puppet used in the

present study, the control process can be implemented in such a way that when the user performs a "pinch" gesture, the location of the imaginary stick is set to be the position of the hand at that moment. If the user's hand moves while maintaining this gesture, the movement of the puppet will follow the position and rotation of the hand.

4.4.2 Manipulation with Strings

A key element of a traditional marionette is its strings. It is the strings that make expressive puppet actions possible. String manipulation is thus an essential feature of a puppetry performance.

In the case of real marionettes, the puppeteer usually pulls or loosens the strings to create interesting puppet motions. This notion is incorporated into the present research. The interface was designed so that when the user performs a "pinch" gesture, the system receives the command that from this time on the hand movement shall act as a guiding element to animate the character, and the hand position is recorded. If the user changes the gesture to "release", the system records the current hand position. A 3D guiding vector is then formed from the point where the "pinch" gesture was executed to the point where the "release" gesture was executed. The guiding vector indicates the direction and extent of the user-desired movement of the limbs of the character. In this context, the imaginary strings can be regarded as rigid rods.

While the impulsive control described above can be seen as an extension of real marionette string manipulation, a smoother slow control mode was designed for kinematic control of the character. In this mode, the user can control the character's limbs by using an inverse kinematics (IK) solution for the manipulated end-effector.

In real marionette performances, it can be observed that puppeteers sometimes execute amazing maneuvers by abruptly changing the effective length of the active string. Similarly, in the case of the virtual puppet, an interaction mechanism can be designed so that when the user performs a particular gesture, the length of a selected string instantly

increases or decreases by a certain amount.

4.4.3 Animation Mode Implementation

The interaction interface was applied to the two animation control modes.

In the authoring animation mode, the user first employs the "object selector" mechanism to explore the virtual scene. The user then uses the "pinch" gesture to select the desired object, and the "pinch" and "release" gestures to form a guiding vector to move the selected object. The "tapping" gesture can then be used to record the animation.

In the performance-based animation control mode, instead of using an "object selector" to select the desired object, the user can use a 2D selection interface, since it is usually difficult for users to select moving virtual objects. In order to use the 2D selection interface, the user performs a "tapping" action on a surface menu, as described in [18]. A detailed discussion will be provided later. "Pinch" and "release" gestures are used to create a guiding vector to guide the character to the next motion, following the motion currently being performed.

In the next two chapters, we demonstrate the detailed implementation of our techniques in two examples: a digital marionette controlled by a real time physical simulation and a human figure with a motion repertoire represented by the motion graph of a motion capture data set. Experiment results are provided respectively.

Chapter 5

Physics-based Marionette Application and Results

Based on the general framework of the interaction concepts described in the last chapter, in this chapter a detailed implementation of a physics-based puppet application is introduced. Section 5.1 states the design objective. The interaction tasks were first realized with a 2D keyboard/mouse interface, as described in Section 5.2. The implementation details are presented in Section 5.3. First "Breve", the software package used to generate physics-based animation is introduced, then the technical details of interactions with the Breve simulation on a volumetric display are explained. The 3D interaction experiments are described in Section 5.4 and the results are presented in Section 5.5.

5.1 Real Marionette Scenario

As discussed in 4.4, during a real marionette performance, the puppeteer executes all kinds of maneuvers by manipulating the control stick and the strings. The control repertoire includes moving the stick back and forth and up and down, pulling the strings in all directions, and loosening or plucking the strings. In the present research, a similar interaction task repertoire was designed, since these are the actions with which professional

puppeteers are familiar and the ones they would feel most comfortable using to control a virtual puppet.

5.2 2D Interaction Experiments

Prior to exploring the interactive control of 3D animation on a volumetric display, basic mouse/keyboard interface interaction experiments were first carried out with animation on an ordinary screen.

A physics-based simulated 3D virtual puppet was created, as shown in Figure 5.1. Interactions with the virtual model were explored using a basic mouse interface and keyboard control.

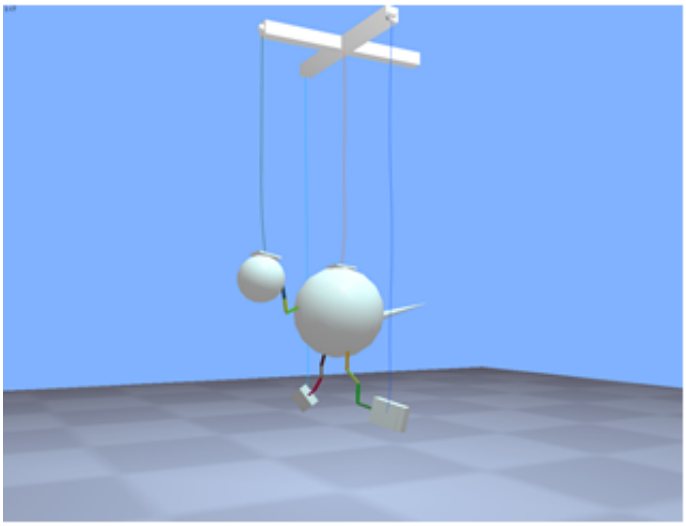


Figure 5.1: Virtual ostrich marionette

In this scenario, the puppet is a physics-based model composed of multiple linked parts connected by joints with limited DOFs. With this design, the puppet is capable of many interesting actions, but is not so complex as to make the whole system unstable or the simulation speed too slow. The control stick above the puppet is not physics-based, since the objective was to simulate a real-world performance, where the puppeteer can

maneuver a marionette's control stick at will, and the strings and the puppet will follow its movement.

Creating reasonable physics-based strings was critical for this system, because the strings act as bridges during the interactions between the puppet body and the stick controlled by the animator. An effective, practical approach described by Jakobsen [23] was employed, where each string is modeled as a collection of particles and the distance between each pair of particles is constrained to a fixed value. This permits an individual string to be considered as a particle system with external forces (such as gravity) acting on it. The string has to satisfy both internal constraints (e.g. a fixed distance between two particles) and external constraints (e.g. attachment of a particle in the string to a point in 3D space) simultaneously.

Three aspects of interactions with the 3D virtual puppet are examined below: stick control, string manipulation and puppet control.

Stick Control

The user is able to shift-click on the stick and move it to any desired location in the 3D virtual space. The user can also use various individual keystrokes to move the control stick an arbitrary number of Breve units in the x, y and z directions, and can rotate the stick by a user-specified number of degrees about the x, y and z axes.

String Manipulation

The length of the string can be increased or decreased a certain amount by means of a keystroke. In addition, the user can detach a string from the control stick and attach it to any other object in the scene. The user can also shift-click on one end of the string and move it randomly in the 3D virtual space, or can select one particle in the string, pin the particle into the simulated world space and swing the remaining string at will.

Puppet Control

Created as a physics-based MultiBody in Breve, the puppet can also be user-selected by shift-clicking, and can follow mouse movement.

5.3 Implementation Details

5.3.1 Breve - a 3D Simulation Engine

Breve is the open-source 3D software package used to construct the physics-based animation for the present research. It is a 3D simulation environment designed for simulating decentralized systems and artificial life. The simulation environment provides a clean programmable framework for creating artificial life and simulating behaviors in a 3D virtual environment. Within the overall framework, several individual components address individual aspects of the simulation, such as graphics display, physical simulation and collision detection. All of the components communicate with and are managed by the Breve engine. The physics-based engine was used to simulate the 3D virtual puppet in our work and the OpenGL graphics engine was used to visualize the simulated world.

The "steve" Language

Simulations in Breve are written using a simple interpreted object-oriented language called "steve". The steve language, especially designed for the implementation of 3D simulations, handles garbage collection, supports lists and 3D vectors as native types, and includes a set of classes which interface with the simulation features of the Breve engine. The steve language is a procedure language, with many features similar to those of C. The fact that steve is an interpreted language which delays several type checks and method lookups until runtime can give rise to speed concerns. However, in practice, the limiting factor for Breve simulations is usually the physics-based simulation and graphical

rendering rather than the execution of the steve code.

Physics-based Simulation

Every simulation is controlled by a single controller object, which is an instance of the built-in Control class (or one of its subclasses). The controller instance is defined in the simulation source code file and is the only instance which is automatically created when a simulation is run. Other instances are all created by the controller object and communicate with it. The controller object controls the settings of the simulation and serves as the main interaction interface between the user and the Breve environment. It can thus be considered to be the main element of the simulation.

The physical simulation engine uses the instantaneous state of objects in the simulated world, together with the internal force (such as joint torque) and external forces (such as gravity) acting upon them to compute the state of the objects in the following time step. The Breve physical simulation engine handles rigid body simulation, collision detection, collision responses, as well as integration aspects and the general state of the simulated world.

5.3.2 Interaction with Breve Simulation using the Volumetric Display

Since only wire-framed images can be rendered on the volumetric display, the "outline" drawing mode in Breve was employed to generate the animation on the display. Figure 5.2) above shows the virtual marionette used in the experiments. In the case of Breve animation shown on the volumetric display, several modifications to the original system were required in order to extend the original 2D mouse/keyboard interaction to 3D hand manipulation. First, the Vicon real-time SDK was used in order to obtain cleaned motion data from the Vicon real-time server. Secondly, due to the discrepancies between the Vicon coordinate system and that of the volumetric display, calibration data

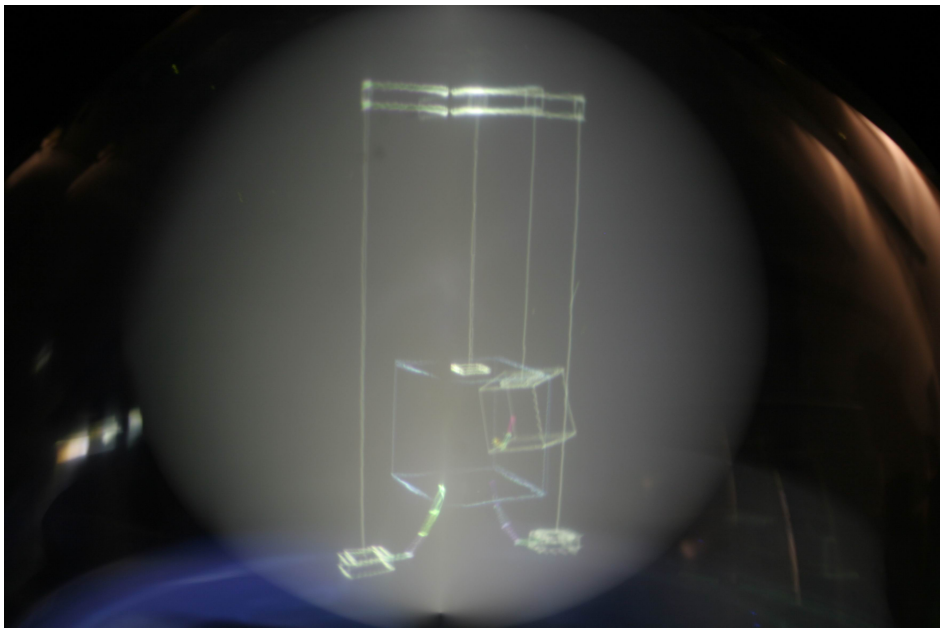


Figure 5.2: Virtual marionette on the volumetric display

were necessary in order to calculate the required conversions between the two systems. Thirdly, it was necessary to embed the Vicon code into the Breve source code so as to obtain real-time Vicon data and interaction interface information directly in Breve.

Vicon Real-time SDK Interface

The Vicon real-time system communicates all motion capture data in packages, using a client/server paradigm with TCP/IP (stream sockets). After connecting to the real-time server, a client should first send a request for an information packet, to which the server will reply with an array of the names of the various channels that are available. When the client makes subsequent requests for a data packet, the server replies with a sequence of real numbers, where the position of each value indicates the corresponding channel. In the present research, the client end code was implemented in C.

Mapping Between the two Coordinate Systems

The origin and units of the Vicon coordinate system differ from those of the volumetric display. The Vicon data are recorded in millimeters, whereas the volumetric display coordinates have their own units. In addition, the Vicon origin (determined by the Vicon calibration process) differs from the origin in the volumetric display (which is positioned in the center of the display).

The following steps were used to obtain the transformation matrix for the two coordinate systems: In the program to initialize the hmesa library, the system was set up so that only images in a canonical view volume, a cube extending from -1 to 1 in each direction, are shown on the volumetric display. This means that a vertex at point (1,0,0) is right at the edge of the display. Given a display radius of 5 inches and Vicon units in millimeters, then

1 unit in the volumetric display = 5 inches

1 Vicon unit = 1 mm = 0.03937 inches

The above equations define the scale coefficient for the two coordinate units. To calculate the offset, the equation

volumetric_display_coordinates = vicon_scale*vicon_coordinates + vicon_offset.

was used, where vicon_scale is the scale coefficient. In order to calculate vicon_offset, the Vicon coordinates for a marker at the center of the top of the dome were obtained. Since the corresponding volumetric display coordinates of this marker are known, the vicon_offset can be calculated. Thus, conversions can readily be made between the coordinates of these two coordinate systems.

Program Design and Structure

Breve source code is written in C, whereas the Vicon SDK interface is written in C++. In addition, the Breve simulation file is written in the steve language. Wrapper function files were created for Vicon SDK interface to be imported into the Breve source code.

In order to construct the prototype, interaction classes (i.e. the Hand class and the Fingersegment class) were also developed, based on the Vicon real-time SDK interface.

The following diagram illustrates the interrelationships among the Breve engine, the Vicon SDK and Vicon real-time motion data (Figure 5.3).

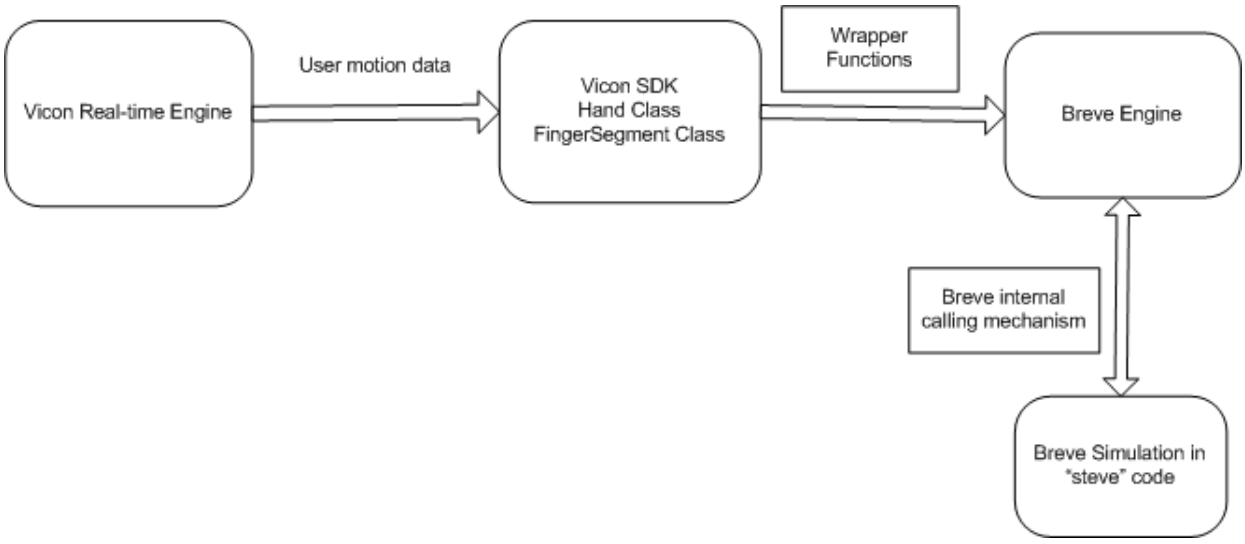


Figure 5.3: Breve/Vicon communication structure

5.4 Basic 3D Interactions

5.4.1 Virtual Control Stick Manipulation

By performing a simple "pinch" gesture, the user triggers the mode in which the virtual control stick dynamically tracks the movement of the user's hand as it moves in the 3D control space above the volumetric display. As is the case with real puppetry, the virtual puppet follows the motion of the virtual stick and thus follows the movement of the user's hand.

5.4.2 Virtual String Manipulation

Given the implementation of the interaction classes, a convenient approach was explored for manipulating the virtual marionette by detaching the strings and attaching them to points in 3D virtual space. When the user performs a "tapping" gesture near one end of the virtual control stick, the string attached to it becomes detached from the control stick and follows the movement of the user's finger. The user can reattach the string to a 3D point by "tapping" again on the display surface.

A further mechanism was developed for controlling the marionette. As discussed in 4.3, the "object selector" follows the movement of the user's hand. When the user performs a "pinch" gesture, a body part which is attached to a string and which is closest to the "object selector" is selected. Instantly, the string force acting on the marionette body part is replaced by a spring model. The equilibrium position for the spring model is the body part position at the time when the user executed the "pinch" gesture. An invisible object is created at the same position. When the user's hand moves, the invisible object follows the movement of the hand, and the distance between the final and initial positions of the invisible object becomes the displacement for the spring model.

With all of the above approaches, the user can manipulate the marionette by removing the string from the control stick, swaying it, and pinning it in 3D space. The user can also animate the marionette by detaching its body from the strings, moving it around and posing it at will.

5.5 Results

The disadvantage of the 2D display and the mouse/keystroke control interface is the lack of control primitives for a large number of degrees of freedom. The mouse input can control at most only two degrees of freedom, and keystrokes are even more limited. This type of interface therefore cannot provide a sufficient control space for interactions with

3D characters.

In the 3D prototype system, the physically simulated virtual puppet was displayed on the volumetric display and it was controlled as expected, using the methods described above. Figure 5.4 below illustrates a sequence of the resulting animations, shown on a normal screen.

One drawback of this virtual marionette system is that the refresh rate of the animation is too slow for genuinely interactive work. In the virtual marionette setup there are four strings, each consisting of 20 particles. Because all of the particles are associated with 3D position, velocity and force information, the calculation of all of the necessary data and the transmission between the host machine and the volumetric display greatly reduce the speed of the animation. The Breve simulation speed is also a contributing factor with regard to the slow animation speed. It is expected that improvement in the API of the volumetric display and the Breve simulation will alleviate this problem.

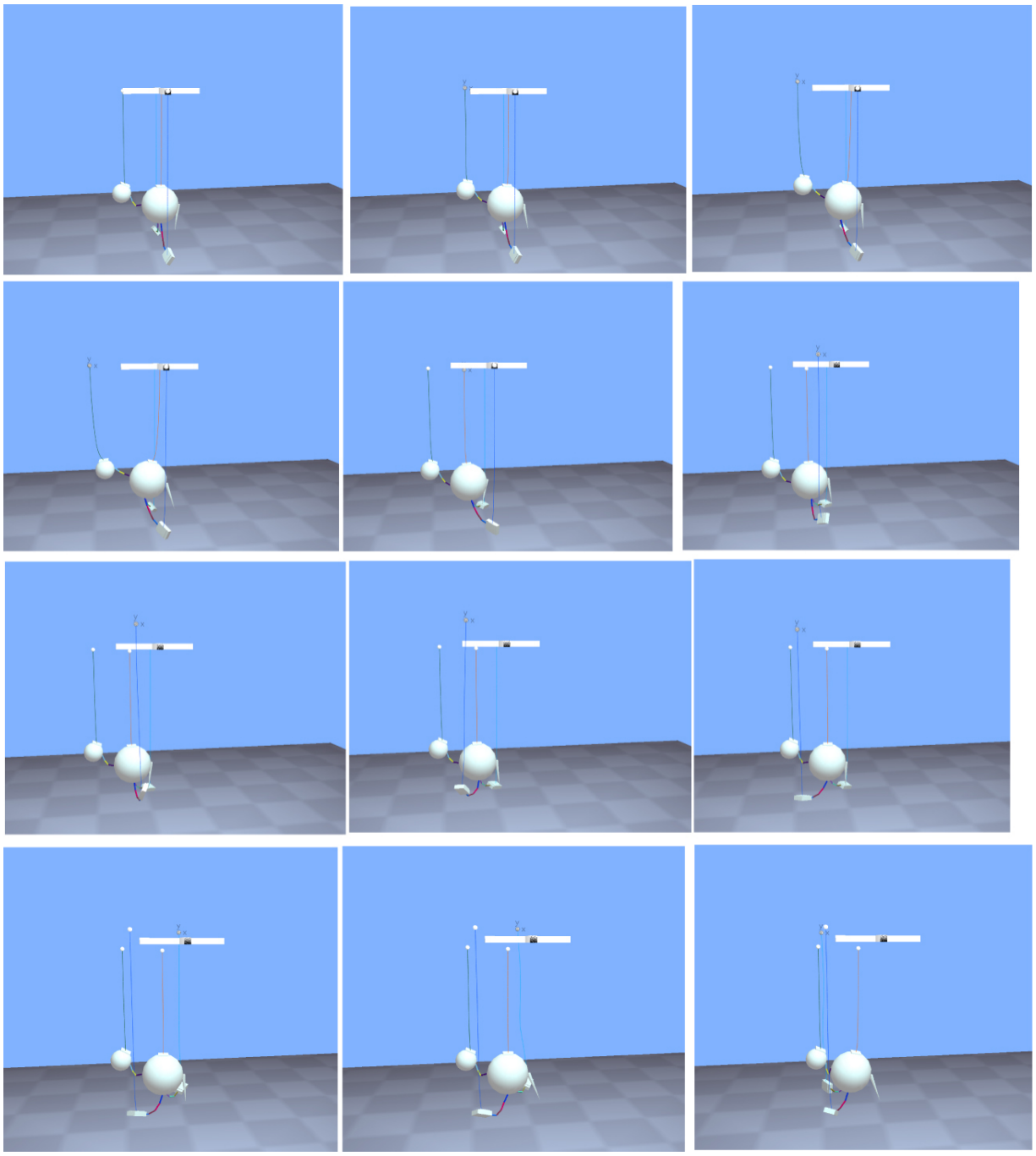


Figure 5.4: interaction results

Chapter 6

Motion Graph Exploration Application and Results

In addition to applying the design concept and the 3D interaction techniques to physics-based simulations, they were further explored by being incorporated into a motion graph application based on a course project by Alex Hertel et al. [21]. A human figure with a motion repertoire represented by the motion graph of a motion capture data set was shown on the volumetric display. A prototype system was developed to enable users to guide the character's actions to flexibly create sequences of animations based on a motion graph search algorithm.

6.1 Technical Overview

The prototype system was constructed based on the course project "Collaborative Motion Graph" [21]. First a general overview of this project will be provided and then the interaction design and implementations of the present research will be presented.

6.1.1 Project Overview

The input of the project is a set of .mov files containing 3D position information for all of the markers on an actor's body, obtained in a motion capture session. After the motion capture data set is imported into the project, a data structure is created to store the 3D positions of each marker for every recorded frame. The data set can also be grouped based on the segmentation of the skeleton. Motion graphs are created for each data set group.

6.1.2 Application Interfaces

The main component of the project is an application, CollaborativeMotionGraphProject.exe, which is dialog-based. The process for constructing the collaborative motion graph data involves six steps, which correspond to six selection tabs in the main dialog box. The main windows corresponding to the six selection tabs are as follows.

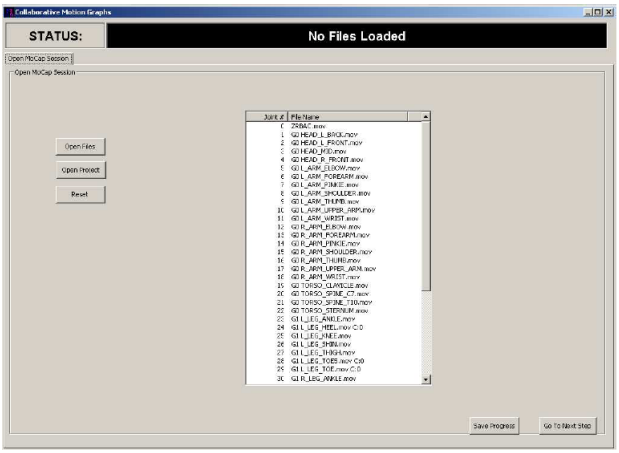


Figure 6.1: The 'Open Files' Tab

Figure 6.1 illustrates the initial program interface, where a motion capture session is loaded into the program. In step two (Figure 6.2), the user views the motion capture data in its original form. Users can view the raw motion capture data by pressing the "Play" button. In step three (Figure 6.3), the motion data for the whole body is divided

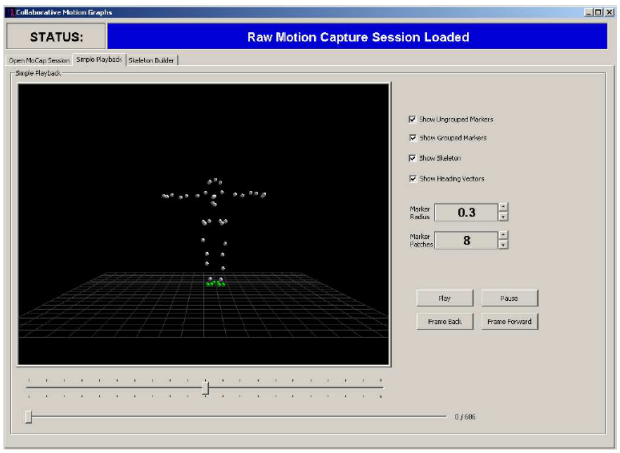


Figure 6.2: The 'Simple Playback' Tab

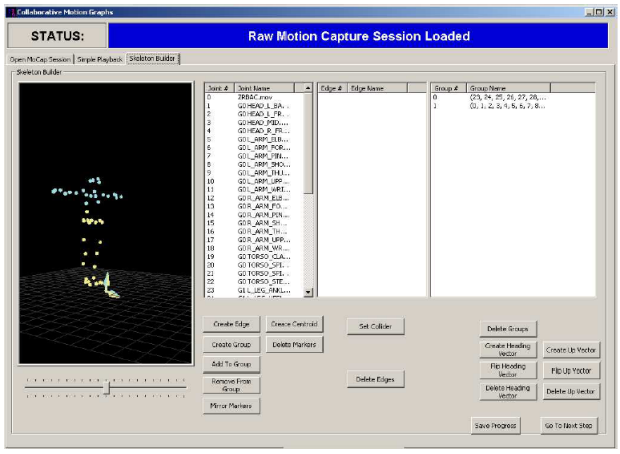


Figure 6.3: The 'Skeleton Builder' Tab

into groups according to the user's specifications. An inner data structure to store the motion data is constructed and several feature variables (such as the heading vector and up vector) are created for each group. In the "Animation Dissection" step (Figure 6.4), the animation is dissected manually by the user or automatically by the program, to create a motion graph for each group. In step five (Figure 6.5), the user inspects the motion graphs which have been generated and modifies them if necessary. Users can review the animation for each node of a group motion graph by selecting the node and pressing the "Play" button. The last step, "Collaborative Motion Graph Playback"

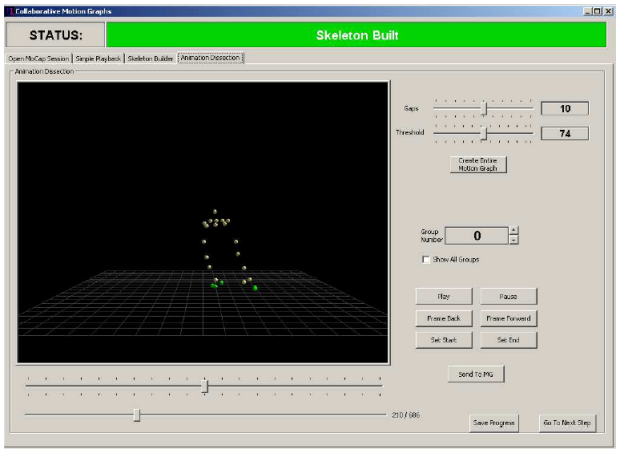


Figure 6.4: The 'Animation Dissection' Tab

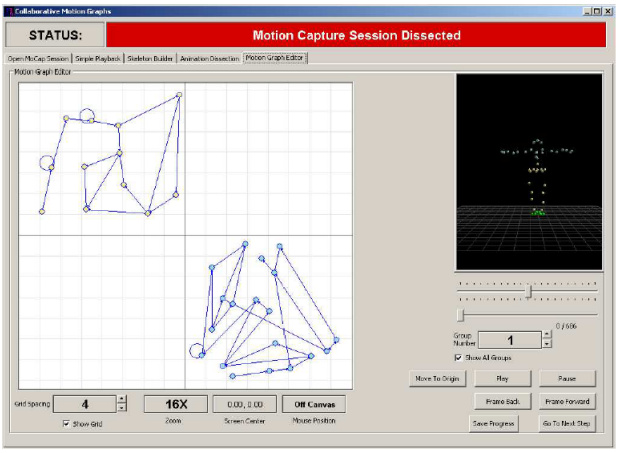


Figure 6.5: The 'Motion Graph Editor' Tab

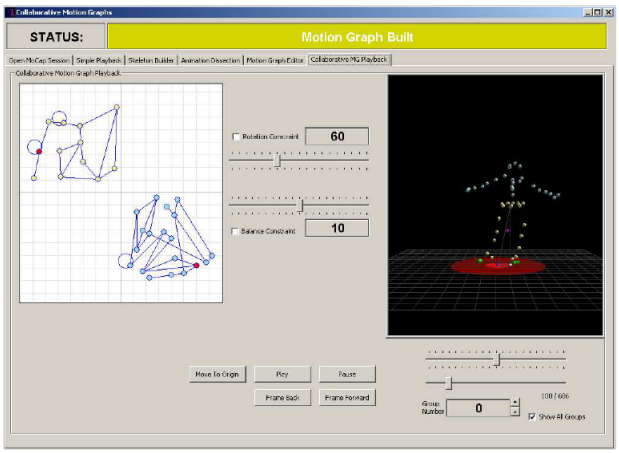


Figure 6.6: The 'Collaborative Motion Graph Playback' Tab

(Figure 6.6), shows the user how the different motion graphs interact with one another. As the character moves, the relevant nodes from each group are highlighted in the graph as they come into play. Two types of constraint were also implemented to control the interactions between different graphs.

6.1.3 Motion Data Acquisition

The motion data used in the present study was captured by twelve Vicon cameras with a refresh rate of 120 Hz. This motion set was designed to enable the character to perform relatively clear, distinct motions, including the following:

Action Number	Action Description
Action 1	left arm moves straight up from front of body
Action 2	left arm moves straight up from side of body
Action 3	right arm moves straight up from front of body
Action 4	right arm moves straight up from side of body
Action 5	left leg kicks forward from front of body
Action 6	left leg kicks up from side of body
Action 7	right leg kicks forward from front of body
Action 8	right leg kicks up from side of body
Action 9	jump one step to the left
Action 10	jump one step to the right
Action 11	walk one step forward
Action 12	walk one step backward
Action 13	walk forward 45 degrees to the left
Action 14	walk forward 45 degrees to the right
Action 15	walk backward 45 degrees to the left
Action 16	walk backward 45 degrees to the right
Action 17	turn around

The entire body motion contains data corresponding to 40 markers, captured in 4850 frames. The data are then exported to a V-file. The V-file is imported into the Maya software and all of the 3D data information are exported to .mov files via a built-in feature of Maya.

The motion data of a dancing figure was also used and experimented in the prototype. The snapshots in Figure 6.12 illustrate some interesting sequence of the motion.

6.2 Design Framework

As discussed in Section 4.1, the intuitive design concept of the interaction interface is that the puppet is suspended from an imaginary control stick, with imaginary strings attached to its limbs. A prototype is thus proposed where the user moves the character's limbs as if there are strings connecting the user's fingers to the character's body parts. Since motion capture data rather than physics-based simulation is used to animate the character in the present study, it is necessary to construct a correlation between the user hand input and the motion data. In the original motion graph exploration project, the animation is generated by randomly traversing the motion graph. Therefore, a construction of a correlation between the user hand input and the traversing of the motion graph becomes a natural implementation, where the user hand input can communicate the intention for the next motion by controlling the selection of the next node in the graph. For example, when the user's hand performs a gesture aiming to raise the character's left arm, the program should search the motion graph data to find the graph node with the most similar motion and make it the next node to be traversed.

In order to implement the interaction techniques prototype, several new elements were incorporated into the original system, as described below.

6.2.1 User Input

So as to capture the user's hand and finger motion information in real time, the Vicon SDK and all of the interface code (Hand and FingerSegment classes) were incorporated into the original project. The project is programmed in C++, which makes it convenient to add the Vicon SDK classes and the extended interface classes.

6.2.2 Limb Class

As discussed above, the essential aim of the prototype is to permit the user to guide the traversing of the motion graph by means of hand and finger motions. During the course of an animation, different parts of a character's body may move in different directions to differing extents. This raises the question as to which motion should be chosen as the criterion for evaluation of the next node? An intermediate "limb" class was developed to allow the user to specify which part of the body's motion should be used as a criterion. Four limbs ("left arm", "left leg", "right arm" and "right leg") are specified in the prototype. The "object selector" mechanism (see Section 4.3) is implemented to allow the user to select the desired limb dynamically in real time.

6.2.3 Traversing the Motion Graph

The user confirms the limb selection by executing a "pinch" gesture, and the selected limb is then highlighted, with its color being changed from white to red. The movement of the selected limb thus becomes the criterion for choosing the next node in the motion graph.

The method used to find the next appropriate node is as follows. Suppose the current node is n_0, and there are N potential next nodes $(n_1, n_2, \ldots, n_n)$ for n_0, as illustrated below in Figure 6.7. For each node in the next nodes set, the selected limb movement is represented as a vector in 3D space, referred to as the "candidate motion vector". The vector is formed from the limb's position at the initial frame to the limb's position at the final frame of the node. Thus, for the N next nodes $(n_1, n_2, \ldots, n_n)$, there are N corresponding candidate motion vectors $(v_1, v_2, \ldots, v_n)$. Another 3D vector v_0, referred to as the "user guiding vector", is used to represent the user hand movement between the execution of the "pinch" and "release" gestures. The similarity between vector v_0 and vector $v_r, r \in [1, n]$ is evaluated by using a combination of two criteria: the angle between

the two vectors and the similarity in length of the vectors. The angle is calculated in accordance with the following equation.

$$\cos\theta_r = \frac{v_0 \cdot v_r}{|v_0| \cdot |v_r|} \tag{6.1}$$

The similarity in length is calculated in accordance with the following equation.

$$s_r = \frac{||v_r| - |v_0||}{|v_0|} \tag{6.2}$$

Different weighting parameters can be applied to the two criteria, depending upon which aspect of the motion users wish to control. By default, the angle between the vectors has greater weight. The node corresponding to the smallest angle and to a length most similar to the length of the "user guiding vector" is chosen as the next node.

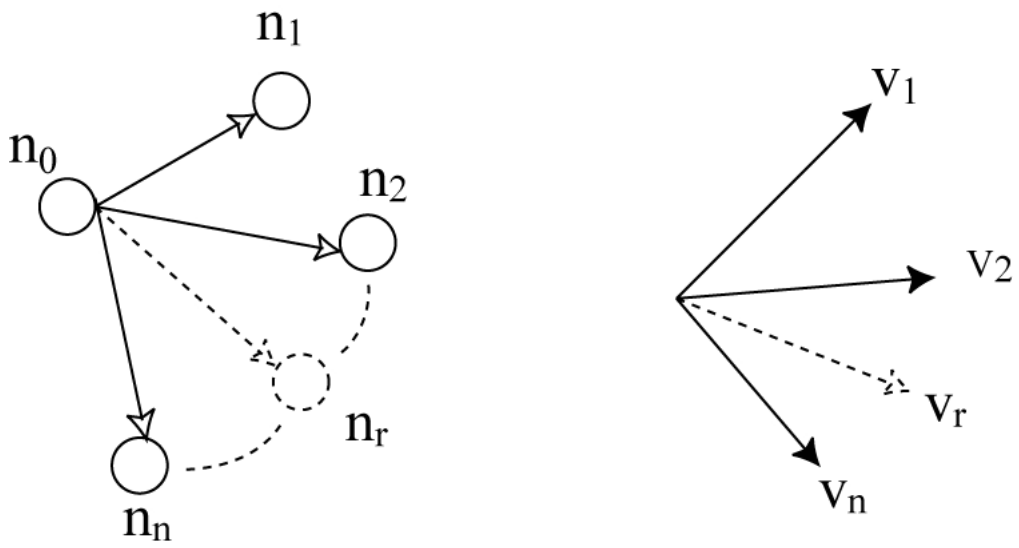


Figure 6.7: Algorithm for Traversing the Motion Graph

6.3 Basic Interactions

Once the above-described implementations have all been incorporated, a prototype system is constructed where the user can freely select limbs and guide the character to

achieve desired motions. The whole process runs in "performance-based animation" mode (4.2), where the motion graph is randomly traversed and the character is in perpetual motion. The "object selector" (4.3) gives users visual feedback regarding their own hand movements by tracking the hand position. When the user's hand moves, the limb closest to the "object selector" is highlighted in green. The user can confirm the selection by executing the "pinch" gesture. The selected limb will then be displayed in red, indicating that the movement of this limb has become the evaluation criterion for the selection of the next node. The user can then specify the desired direction of limb movement by moving the hand while maintaining the "pinch" gesture, and then executing the "release" gesture to set the user guiding vector. The next node will automatically be calculated and searched, and the animation for the next node will then follow the final frame of the current node.

A "repeat animation" mode is implemented so that the user can cause the program to execute the motions in a given node repeatedly. During the animation sequence for a node, if the user "taps" the spherical dome of the display, the current node will be recorded, and after the final frame of the sequence, the animation will recommence from the initial frame of the current node. When the user's "tapping" action is detected a second time, the "repeat animation" will stop, and after the final frame of the current node, the character will be restored to random motion.

6.4 User Feedback and Preview Mode

User feedback is a very important aspect in designing interaction interfaces. Real-time visual feedback can help users understand the interaction process and accelerate learning, especially in cases such as the present study, where responses from other channels are not available. The highlighting of the selected limb merely represents basic visual feedback. The primary problem that must be solved is how the user can learn which

hand actions can guide the performance of the character and, more concretely, how the hand movements affect the traversing of the motion graph and thus affect the character animation. In order to instruct the user in guiding the character, the visual feedback was enhanced by developing a preview mode. In preview mode, the most similar node in the motion graph is found dynamically, and the animation for that node is simultaneously displayed. When the user executes the "pinch" gesture, the user guiding vector is shown by a blue line with an arrow indicating the direction. At the same time, the limb movement vectors for all of the next nodes are calculated. A threshold is set, so as to select a set of the most similar potential limb movement vectors. These vectors are shown in the same manner, except that they differ in color (purple) and size from the user guiding vector. In addition, the most similar limb movement vector is drawn in a color (green) which differs from other potential movement vectors in the set, as shown in Figure 6.8. In the meantime, the next node corresponding to the most similar limb movement vector is found in the graph, and the animation for that node is played back repeatedly, for as long as this limb movement vector is the most similar to the user guiding vector. The whole process is dynamic, which means that once the user's hand movement results in a change in the most similar node, the animation will change accordingly. The current animation is always the motion for that node.

With this dynamic mechanism, by moving the user guiding vector, users obtain a clearer idea of the candidate motions for the selected limb. Once the user decides to guide the character to a certain movement (for example, motion 1), the user needs to move the guiding vector so that the current previewed animation is motion 1. Selection of this node is then confirmed by execution of the "release" gesture.

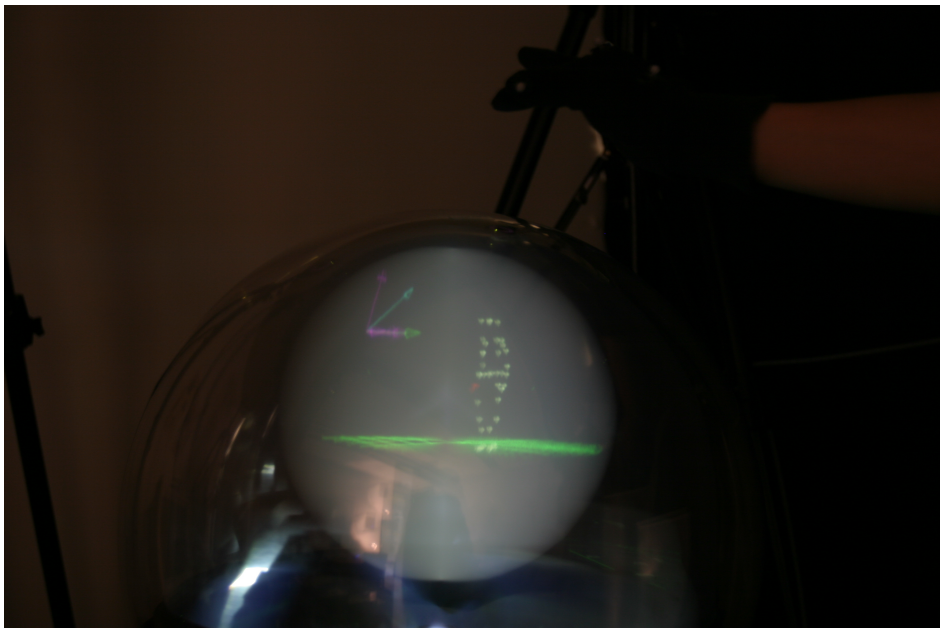


Figure 6.8: The Set of Candidate Vectors

6.5 3D Selection versus 2D Selection

Although the 3D interaction space provides considerably more control primitives than is the case for 2D input devices, it also introduces much greater complexity than is the case with traditional 2D interfaces. In the present experiment, it was found that users sometimes have difficulty in moving the "object selector" close to the desired limb and selecting it, especially in the performance-based animation mode, where the motion graph is traversed randomly and the character is constantly moving, with users unable to predict subsequent motions. Like the interface for a 2D touch screen, a "surface menu" [18] is an approach for displaying frequently used commands as buttons on the surface of a display. As shown in Figure 6.9, four pre-printed acetate overlays were taped onto the surface of the display, to represent commands for the selection of each of the four limbs.

In the case of the 2D selection surface menu, when the user performs a "tapping" action on one of the menu items with the index finger, the limb corresponding to the name on the menu is selected and is highlighted in red. This greatly increases the

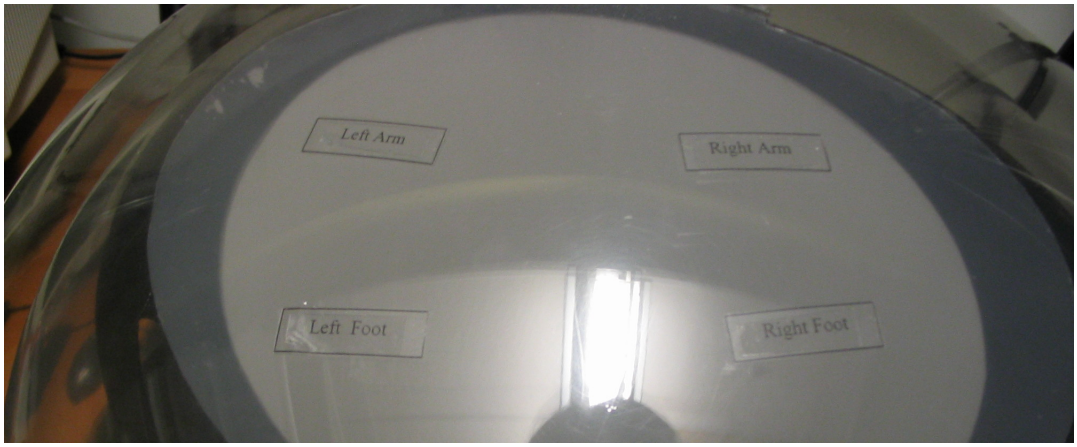


Figure 6.9: 2D Selection Menu

convenience and ease-of-use for users who try to manipulate a limb while the character is performing random, unexpected movements.

6.6 Primitive Kinematics Control

In addition to implementing impulsive user interaction, basic continuous kinematic control of the character's movement was also tested. A hierarchical kinematic model was constructed for the character. The 3D position of each joint is represented as a transformation with respect to its parent position. In this way, kinematic control can be achieved by changing the transformation matrix or the parent position. In the present study, the character's limbs are controlled by an inverse kinematics solution for the manipulated end-effector. Figure 6.10 shows an animation in which the character is bending its waist to the right side of the body in response to user hand manipulation.

6.7 Results

Using the prototype, the techniques developed were applied to animate a human figure with a motion repertoire represented by a motion graph on the volumetric display. Fig-

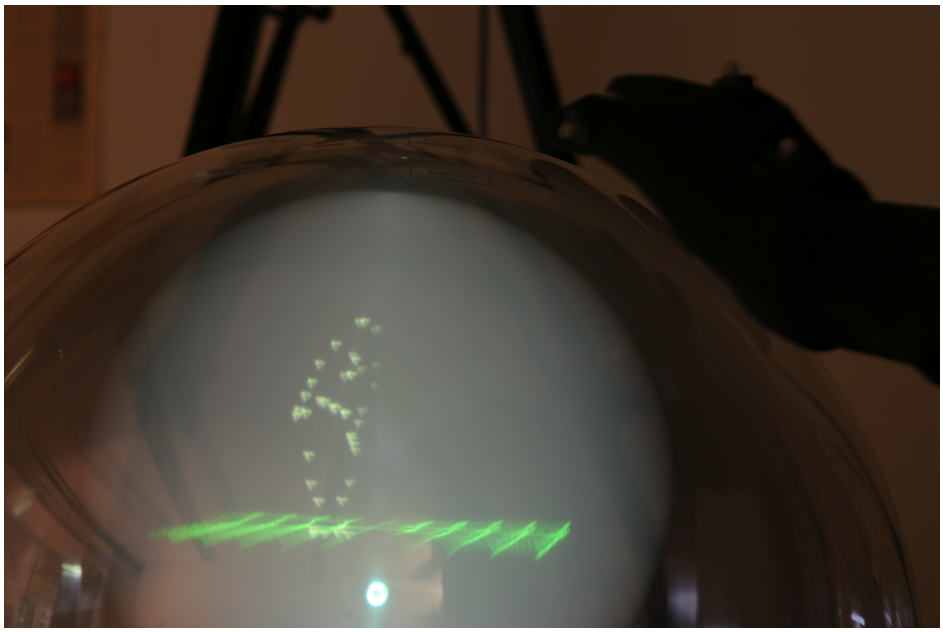
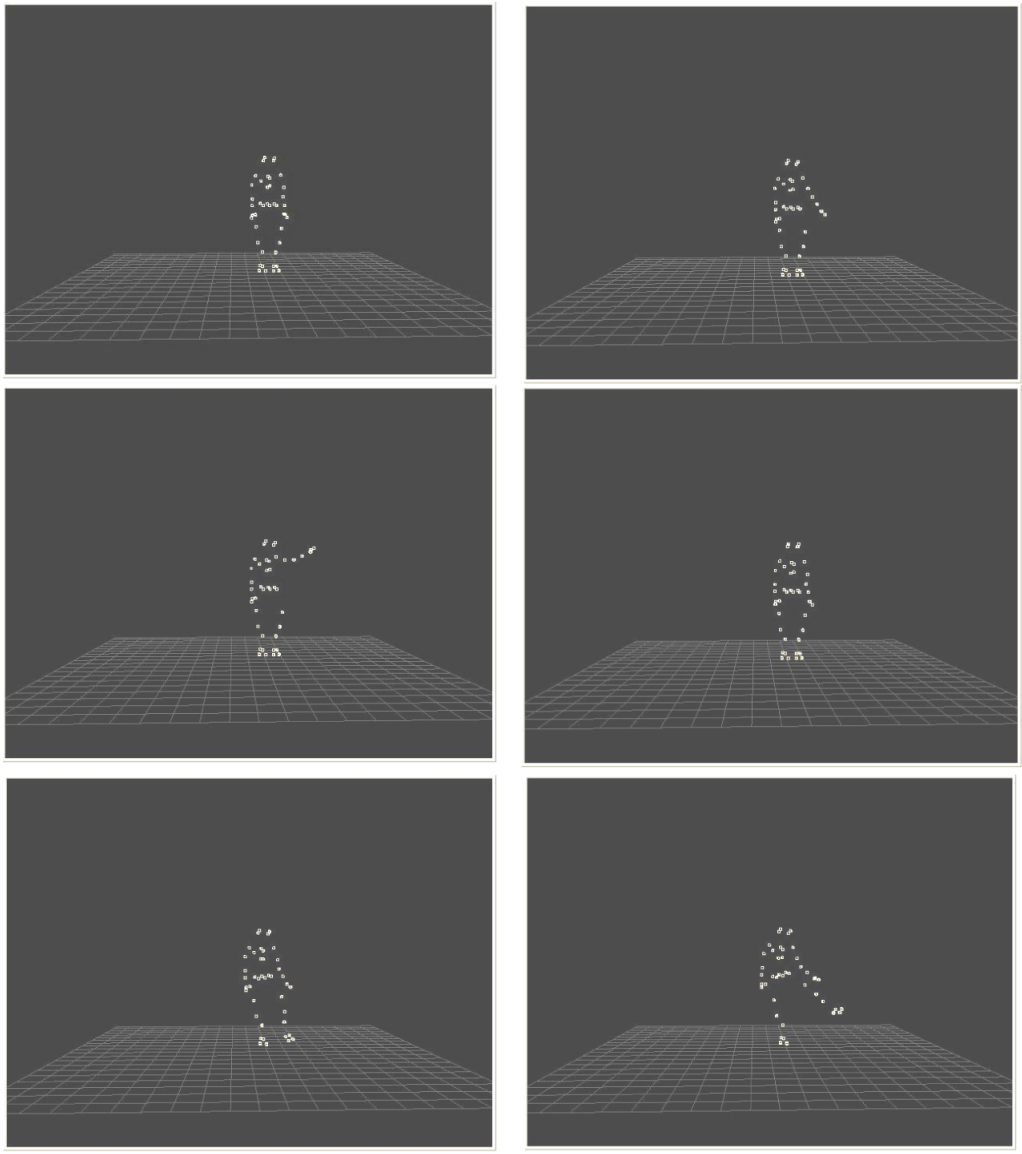


Figure 6.10: Primitive Kinematic Control

ure 6.11 below illustrates a sequence of the resulting animations, shown on a normal screen.



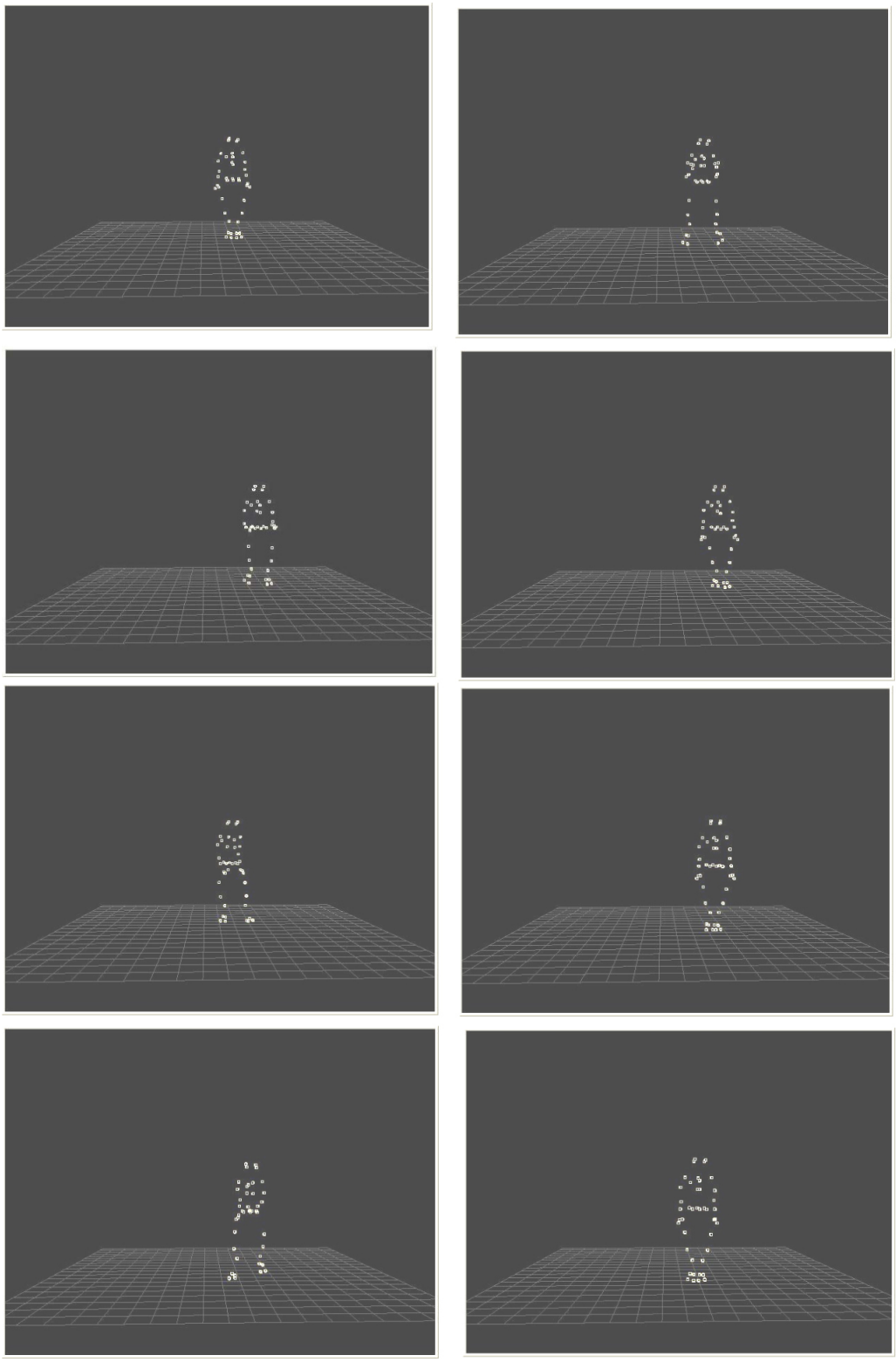
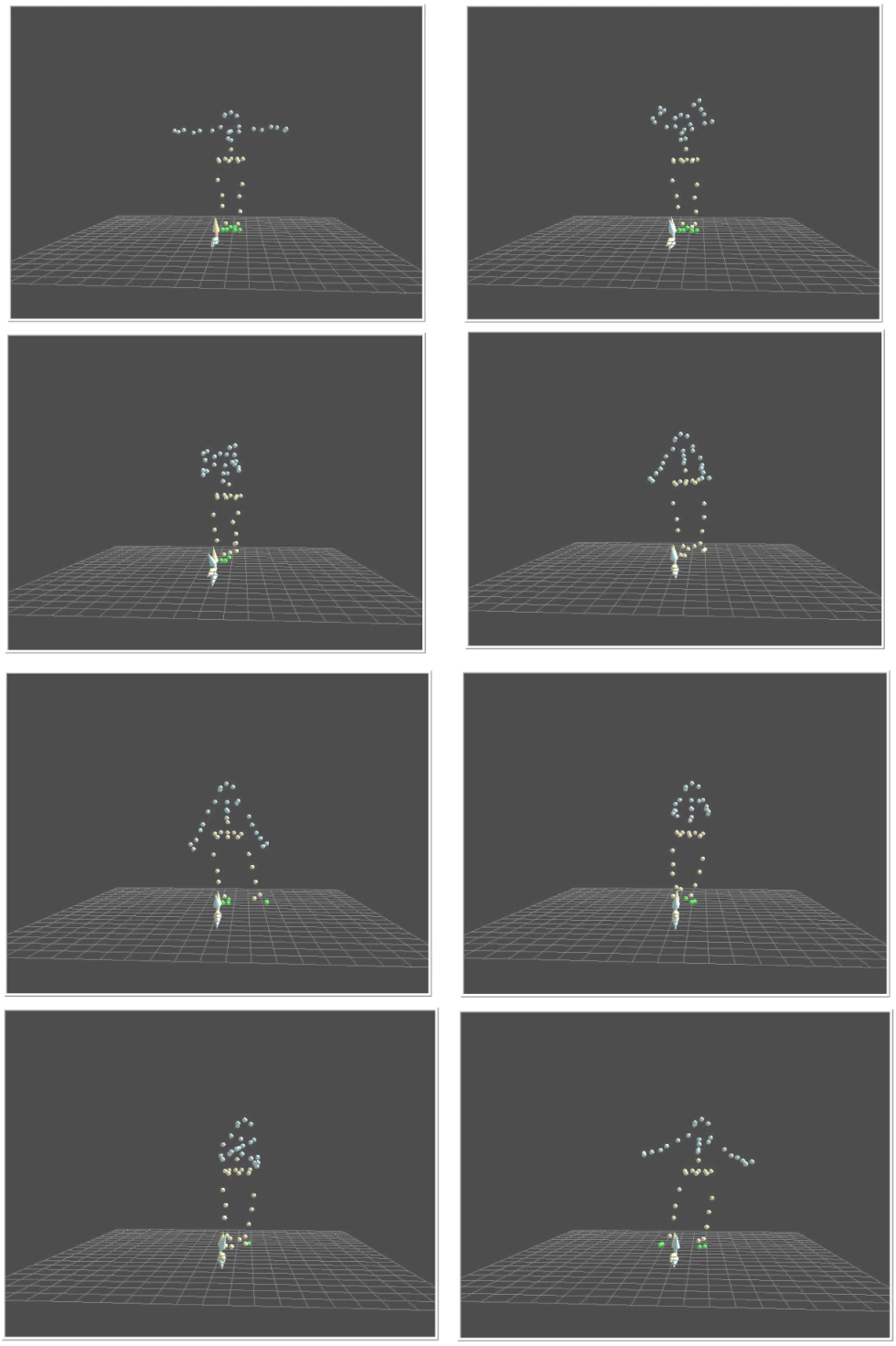


Figure 6.11: A sequence of animation (from left to right, up to down): a).character lifts up his left arm b).character lifts his left leg c).character jumps one step to the left d).character walks forward 45 degrees to the left e).character walks forward 45 degrees to the right



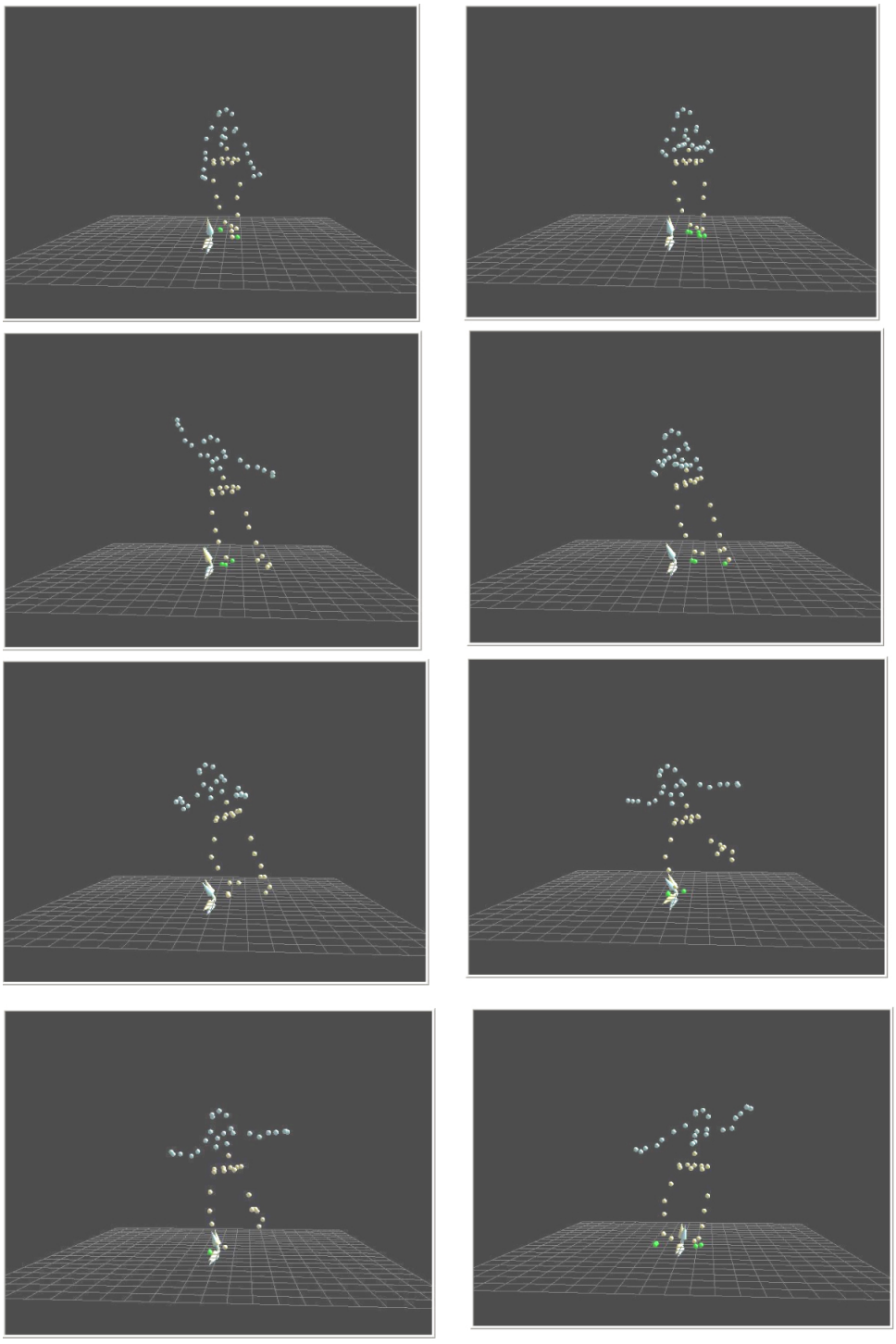


Figure 6.12: A sequence of dancing animation (from left to right, up to down)

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In the present research, a design framework for real-time 3D computer puppetry with a volumetric display is proposed, and a prototype interface constructed to allow novice users to control and create computer animations on the volumetric display. Since the target users are novice animators, a straightforward intuitive interface is designed with a small set of gestures, and characters are used which already have a large repertoire of motions. A glove with attached markers is employed as the medium for users to express their control intentions regarding the computer animation on the 3D display. The key design idea of the interaction interface is derived from traditional marionette performances in the real world, where puppeteers create interesting motions by manipulating the strings from which a puppet is suspended. Several interaction interface elements are implemented: an object selector, a 2D selection surface menu and a set of hand and finger gestures.

Two forms of animation control are explored: authoring animation and performance-based animation. In the case of authoring animation, users can explore the virtual scene by means of the "object selector" mechanism, and can use gestures to select objects, control the character's movement, and record the animation. In the performance-based

animation control mode, users can employ a 2D selection interface to select the character's limbs and can direct the character's future movements by means of gestures. In the present research, the techniques developed are applied to two applications. One application is interaction with a physics-based virtual puppet simulation. Users can control the movement of the puppet by manipulating the virtual strings. Since the refresh rate of the system is too slow for genuinely interactive work, this application will not be well-suited for future work until the implementation can be made efficient enough to allow real-time performance. The other application is the motion graph exploration project. In this experiment, the character is in performance-based animation mode. An individual part of the character is selected using a 2D selection interface, and simple hand gestures are detected and used for determining a "user guiding vector". In accordance with the guiding vector and the selected limb's movement in candidate nodes, the system dynamically calculates the next node that is to be traversed. The animation in that node is simultaneously played back to the user. Users can also cause the motion graph to traverse a particular node repeatedly by "tapping" the display dome.

7.2 Future Work

Only basic aspects of the design framework have been explored in the present study. Much remains to be done in order to extend the design concept and to construct a comprehensive, versatile 3D computer puppetry system.

In the present research, the search algorithm looks for the next node where the limb motion most closely approximates the user guiding vector in the motion graph in real time. A direct extension of this approach would be that rather than looking forward one step for the next node, the algorithm would look forward several steps for a path in the motion graph that would cause the limb motion in the nodes along this path to achieve the closest approximation to the user guiding vector. Based on the user's hand movement,

different paths would be dynamically found in real time. An additional refinement would be to image the user guiding vector not as a straight vector, but as a 3D directed curve. The algorithm would then search for a path in the motion graph where the limb motion is most similar to the user guiding curve. This concept would employ an approach similar to that of the work on "Motion Doodles" by Thorne, Burke and van de Panne [46], in which an animated motion is created for an character by drawing a continuous of lines, arcs and loops in a 2D sketching interface.

In the prototype, only one glove is used, with the dominant hand. A natural future research direction would be to extend the system to two-handed interactions. The simultaneous use of two hands for input has long been recognized as beneficial, as described in the work of Buxton and Myers [49]. With two-handed input, more interesting interactions can be achieved. For example, one hand could pull the character's arm while the other pushes the character's leg at the same time so as to perform interesting character animations. In this situation, the search algorithm would search the motion graph in accordance with a guiding vector from each hand, or guiding vectors simultaneously from both hands with priority set for dominant hand. More control primitives can also be added to the system. For example, the dominant hand could guide the selection of the next node, while the non-dominant hand could control the position or orientation of the whole body. This could enrich the motion repertoire originally acquired from the motion capture data.

Another interesting extension of this work would be to use a virtual hand drawn in the display space, in examining the user's interaction with the character. It is noted that in the VPL Dataglove[19] project, a 3D virtual hand model is constructed and drawn on the screen in real time during the user's manipulation of the object. The drawing of a virtual hand in the image space enables users to observe the relationship between the virtual hand and their own actions, thus enhancing users' intuitive understanding of the interaction process, so as to accelerate learning. A virtual hand model could be drawn

in point line format, where the movement of each point corresponds to the movement of individual markers on the glove; and where two points are connected with a line when the points on the hand beneath the respective glove markers are connected by bone segments. Users could move the virtual hand to the desired limb of the character and could then push or pull the limb to achieve interesting fine control of the character. Users would also be able to apply the inverse kinematics techniques and pose the desired end-effector to animate the character in different motions.

In the course of research with this system, observations during informal user tests indicated that users tend to place their hands on the volumetric display and to touch the display to interact with it. An interesting future research direction could be an experiment with an interaction interface in which motion graph is drawn on the upper half of the dome. With this configuration, a mechanism could be developed such that the character animation in one node could be previewed in the center of the display, triggered by a user's fingertip hovering over the display close to that node. Once the user has studied the animation associated with the nodes, a customized sequence of animation could be created by touching the spot corresponding to the desired node on the display. This system would not work well if there were too many nodes in one graph. The problem can be solved by drawing a subgraph of candidate nodes for the current node in real time.

Bibliography

[1] Tinsley A. Galyean and John F. Hughes. Sculpting: an interactive volumetric modeling technique. *Proceedings of SIGGRAPH'91*, pages 267–274, July 1991.

[2] Michael A. Gigante Andrew Gildfind and Ghassan al Qaimari. Evolving performance control systems for digital puppetry. *Journal of Visualisation and Computer Animation*, 2000.

[3] R. Balakrishnan and G. Fitzmaurice, G.and Kurtenbach. User interfaces for volumetric display. *IEEE Computer*, pages 37–45, 2001.

[4] David Baraff and Andrew Witkin. Physically based modeling: principles and practice. *SIGGRAPH'97 Course Notes*, 1997.

[5] M. Baudel, T.and Beaudoin-Lafon. Charade: remote control of objects using freehand gestures. *Communications of the ACM*, 36(7):28–35, 1993.

[6] B.deGraf. Notes on human facial animation. *SIGGRAPH 89: Course Notes(Boston, Massachusetts)*, vol.22:10–11, 1989.

[7] Veronique Benquey and Laurent Juppe. The use of real-time performance animation in the production process. *ACM SIGGRAPH 97: Course Notes(Los Angeles, California)*, vol.1, 1997.

[8] Richard A. Bolt. "put-that-there": Voice and gesture at the graphics interface. *Proceedings of SIGGRAPH'80*, pages 262–270, July 1980.

[9] D. Bowman and L. Hodges. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. *ACM Symposium on Interactive 3D Graphics*, 1997.

[10] Victor Brian Zordan and Jessica K. Hodgins. Motion capture-driven simulations that hit and react. *SCA'02:Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 89–96, 2002.

[11] Bruce Randall Donald and Frederick Henle. Using haptic vector fields for animation motion control. *Proc. IEEE International Conference on Robotics and Automation(ICRA)(San Francisco, CA)*, April 2000.

[12] D.Weimer and S.K.Ganapathy. A synthetic visual environment with hand gesturing and voice input. *Proceedings of CHI'89*, pages 235–240, May 1989.

[13] D. Ebert, E. Bedwell, S. Maher, Smoliar L., and E. Downing. Realizing 3d visualization using crossed-beam volumetric displays. *Communications of the ACM*, vo.42(no.8):101–107, 1999.

[14] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. *SIGGRAPH'01:Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 251–260, 2001.

[15] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. The virtual stuntman: dynamic character with a repertoire of autonomous motor skills. *Computer and Graphics*, vo.25(no.6):933–953, 2001.

[16] Joseph F.Laszlo, M.van de Panne, and E.Fiume. Interactive control for physically-based animation. *Proceedings of SIGGRAPH 2000, ACM SIGGRAPH*, 2000.

[17] Michael Gleicher. Retargetting motion to new character. *SIGGRAPH'98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 33–42, 1998.

[18] Tovi Grossman, Daniel Wigdor, and Ravin Balakrishnan. Multi-finger gestural interaction with 3d volumetric displays. *Proceedings of UIST- the ACM Symposium on User Interface Software and Technology*, pages 61–70, 2004.

[19] Thomas G.Zimmerman, Jaron Lanier, Chuck Blanchard, Steve Bryson, and Young Harvill. A hand gesture interface device. *Proceedings of CHI + GI'87*, pages 189–192, 1987.

[20] Chris Hand. A survey of 3d interaction techniques. *Computer Graphics Forum*, vol.16(no.5):269–281, 1997.

[21] Alex Hertel and Philipp Hertel. Collaborative motion graphs. 2003.

[22] K. Hinckley, R. Pausch, J.C. Goble, and N. Kassell. A survey of design issues in spatial input. *ACM UIST 1994 Symposium on User Interface Software and Technology*, pages 213–222, 1994.

[23] Thomas Jakobsen. Advanced Character Physics. 2001.

[24] Hyun Joon Shin, Jehee Lee, Sung Yong Shin, and Michael Gleicher. Computer puppetry: an importance-based approach. *ACM Trans. Graph.*, pages 67–94, 2001.

[25] David J.Sturman. Computer Puppetry. *IEEE Computer Graphics and Applications*, vol.18(no.1):38–45, 1998.

[26] David J.Sturman and David Zeltzer. A survey of glove-based input. *IEEE Computer Graphics and Applications*, vol.14(no.1):30–39, 1994.

[27] James J.Troy. Real-time dynamic balancing and walking control of 7-link biped. *Proceedings of ASME Design Engineering Technical Conference*, 1998.

[28] Jessica K.Hodgins, Wayne L.Wooten, David C.Brogan, and James F.O'Brien. Animating human athletics. *In SIGGRAPH'95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 71–78, 1995.

[29] Lucas Kovar, Michael Gleicher, and Frederic Pighin. Motion graphs. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 473–482, 2002.

[30] K.Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, (no.1):5–15, 1995.

[31] K.Pullen and C.Bregler. Motion capture assisted animation:texturing and synthesis. *In SIGGRAPH02:Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 501–508, 2002.

[32] Joe Laszlo, Michael Neff, and Karan Singh. Predictive feedback for interactive control of physics-based characters. *Proceedings of Eurographics 2005*, vol.24(no.3), 2005.

[33] Jehee Lee, Jinxiang Chai, Paul S.A.Reitsma, Jessica K.Hodgins, and Nancy S.Pollard. Interactive control of avatars animated with human motion data. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 491–500, 2002.

[34] J. Liang and M. Green. Jdcad: A highly interactive 3d modeling system. *Computers and Graphics*, 18(4):499–506, 1994.

[35] M. Lucente. Interactive three-dimensional holographic displays: seeing the future in depth. *Computer Graphics issue on Current, New, and Emerging Display Systems*, 1997.

[36] M. Mine. Isaac: A virtual environment tool for the interactive construction of virtual worlds. *UNC Chapel Hill Computer Science Technical Report TR95-020*, 1995.

[37] M. Mine. Virtual environment interaction techniques. *UNC Chapel Hill Computer Science Technical Report TR95-018*, 1995.

[38] Mike Morasky. Wiring cracker: The mechanics of a non-anthropomorphic real-time, performance animation system. *Siggraph 98: Conference Abstracts and Applications(New York), Computer Graphics Annual Conference Series, ACM SIGGRAPH*, page 310, 1998.

[39] Sageev Oore. Digital marionette: Augmenting kinematics with physics for multi-track desktop performance animation. *Ph.D. Thesis*, 2002.

[40] Ken Perlin and Athomas Goldberg. Improv: A system for scripting interactive actors in virtual worlds. *Proceedings of the 23rd annual conference on Computer Graphics and Interactive Techniques*, pages 205–216, 1996.

[41] M. Poupyrev, I.and Billinghurst and T. Weghorst, S.and Ichikawa. The go-go interaction technique: non-linear mapping for direct manipulation in vr. *ACM UIST Symposium on User Interface Software and Technology*, 1996.

[42] John R.Koza. Genetic Programming. *MIT Press*, 1992.

[43] Barbara Robertson. Mike, the Talking Head. *Computer Graphics World*, pages 15–17, 1988.

[44] Emanuel Sachs, Andrew Roberts, and David Stoops. 3-draw: A tool for designing 3d shapes. *IEEE Computer Graphics and Applications*, pages 18–26, November 1991.

[45] Ari Shapiro, Fred Pighin, and Petros Faloutsos. Hybrid control for interactive character animation. *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 455, 2003.

[46] Matthew Thorne, David Burke, and Michiel van de Panne. Motion doodles: an interface for sketching character motion. *ACM Transactions on Graphics, Proceedings of SIGGRAPH 2004*, 23(3), 2004.

[47] T.Igarashi, T.Moscovich, and J.F.Hughes. Spatial keyframing for performance-driven animation. *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2005.

[48] Graham Walters. The story of waldo c.graphic. *SIGGRAPH 89 : Course Notes*, vol.4:65–79, 1989.

[49] W.Buxton and B.A.Myers. A study in two-handed input. *Proceedings of the CHI'86 Conference on Human Factors in Computer Systems*, pages 321–326, 1986.

[50] Andy Wilson. Luxomatic: Computer Vision for Puppeteering. *http://research.microsoft.com/ awilson/papers/luxomatic.pdf*, 1997.

[51] W.Vasulka and Lee harrison III. Pioneers of electronic art. *exhibition catalog for Ars Electronica 1992*, pages 92–95, 1992. Linz, Austria.

[52] Po-Feng Yang, Joe Laszlo, and Karan Singh. Layered dynamic control for interactive character swimming. *In SCA'04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 39–47, 2004.

[53] David Zeltzer. Towards an integrated view of 3d computer animation. *The Visual Computer*, (no.1):249–259, 1985.