TURKISH SIGN LANGUAGE ANIMATION WITH ARTICULATED BODY MODEL

by

Turan Can Gürel

B.S, in Computer Engineering, Middle East Technical University, 2002

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University
2010

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Professor Lale Akarun, whose insistent encouragement was the driving force in the completion of this work.

I appreciate the contributions of Alp Kındıroğlu by acting in the motion capture of the signs and preparation of the images of the letter alphabet.

I also highly value the performance of Pınar Santemiz in reference sign videos.

Finally, I am thankful to Gökhan Uysal for his assistance in creation of the digital artwork used in this project.

ABSTRACT

TURKISH SIGN LANGUAGE ANIMATION WITH ARTICULATED BODY MODEL

Demonstration of sign languages with the computer is a potentially useful learning aid for sign language learners. If implemented as a part of a learning tool, one that includes sign recognition as well, it will invaluable for providing feedback to the learners, a most needed contribution.

Human body animation and motion capture technologies have reached a point where realistic virtual actors can perform plausible human movements in realtime. For this, the motion can be defined on a virtual human skeleton, either by design or by motion capture methods, and then displayed over the skeleton which drives a realistic skin model, visualizing the human body.

In this work we capture Turkish sign language finger spelling alphabet and semi-automatically translate it into a visually appealing model. For capturing the sign language we use a magnetic motion capture system. Then, a playback tool generates sign language demonstrations interactively and in realtime in 3D.

ÖZET

EKLEMLİ VÜCUT MODELİYLE TÜRK İŞARET DİLİ CANLANDIRMASI

Bilgisayar ile işaret dili canlandırması, işaret dili öğrenimi için önemli bir araç olma potansiyeli taşımaktadır. Özellikle işaret tanıma özellikleri de içeren bir paketin içerisine eklenirse, öğrencilere en önemli eksiklerini, geri beslemeyi gideren bir araç elde edilebilecektir.

İnsan vücudunun canlandırılması ve hareket yakalanması teknolojileri gerçekçi sanal aktörlerin, gerçek zamanda inandırıcı hareketleri yaptığı uygulamaları artık mümkün kılmaktadır. Bunun için hareketler sanal bir iskelet üzerinden tasarlanabilir ya da yakalanabilir. Bu iskelet doğru bükülmeleri yapabilen de gerçekçi bir deri modelini yürütmek için kullanılabilir.

Bu çalışmada Türk işaret dili harf alfabesini yakaladık ve yarı otomatik şekilde görsel olarak da çekici olan bir modele taşıdık. Hareketleri yakalamak için manyetik bir sistem kullandık. Daha sonra sunumlar bu iş için yazılmış bir uygulama ile etkileşimli ve gerçek zamanda, 3B ortamda oluşturuldu.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Motivation

Being social creatures, we humans require frequent communication with each other. There are many forms of communication, but the most common, and arguably preferred, method is speech. Speaking has a unique personal taste that, when supported by visual cues such as face and body gestures, helps transmit even the most complicated thoughts and feelings without trouble. It is direct, real time, bidirectional and once learned, straightforward to use.

However, speaking is not made available to everyone. Some people are born with disabilities and some lose their abilities further in life. What many take for granted may be a big issue that needs addressing for these people. Sign language communication can be the answer to this need.

While spoken languages reserve the face and body gestures for subtle details in meaning, sign languages use them to transfer the meaning itself, not employing sounds at all. Hence, they are useful for both speech and hearing impaired people. Sign languages are similar to speech in that they are direct and real time, and unlike lip reading, work bidirectionally. However, as do all communication media, sign languages require that both parties have prior knowledge. This is often not the case, since not all people depend on them. Hence, it would be beneficial to expand learning opportunities for sign languages.

When learning, language skills first start as imitations of the teacher and improve with practice and feedback. Sign languages are no different, but since they are not common in everyday life, such feedback is not readily available to the learners. With this point of view, we believe that the best sign language learning tool would be one that provides this much needed feedback to the learner. Obviously, such a tool would need to be able to produce visualizations of signs. Hence, this work attempts to generate synthetic but realistic sign language demonstrations using computer graphics and character animation

technologies, hopefully sparking some public interest toward this means of communication in the process.

1.2. Related Work

Visualizing sign language communication on a computer character is a complicated process requiring the collaboration of many fields of the computer science discipline.

1.2.1. Animation and Keyframing

At the most basic level, one should first understand the principles behind computer animation and how this animation is displayed. The way it is used in this context, the concept of animation is a specialization of the concept of movie, a display changing in some way over time. The principal method to achieve this effect has long been to change the displayed image quickly in succession, called a moving picture. If the images' replacement rate is high enough, the human brain's visual processing power falls short of interpreting them as separate images, but rather perceives what it sees as a continually changing display. The threshold rate for this illusion is a minimum of about 20 images (frames) per second.

With the large number of frames required to create even a short moving picture, it is quickly evident that an automated machine is required to create images. The cinematograph was invented in the 1890's to take photographs in succession and also display them as a moving picture. The first synthetic animations were created in the 1900's by manually drawing the images (on cels) and photographing them. Unlike the cinematograph, there was no machine that could simply draw the required images automatically. Hence, the process would have become prohibitively expensive if keyframing had not been developed. With keyframing, the more established and talented artists would draw the major images (the keyframes) in the image sequence and junior artists would take examples from those keyframes and draw the rest of the sequence (the inbetweens). This way, synthetic animation production was made more efficient (Figure 1.1).

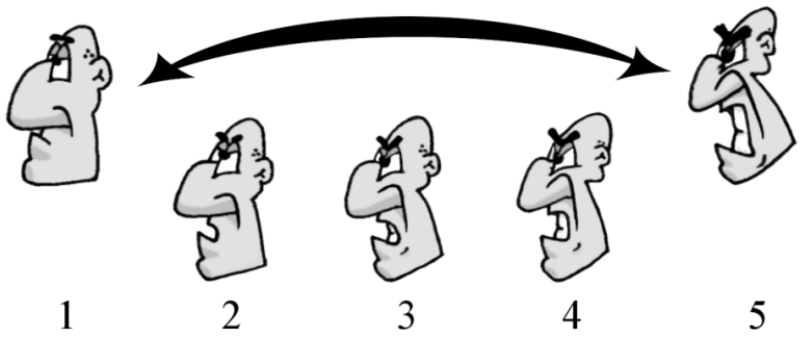

Figure 1.1. Traditional keyframing: Frames 1 and 5 are the keyframes, 2-4 are inbetweens.

While moving pictures were originally designed to be played at theaters; the invention of the cathode ray tube (CRT) displays proved a convenient alternative as they require the display to be refreshed many times in a second to prevent the phosphor from fading away. With the ability to update the display between the refreshes, CRT's lent themselves naturally for animation. However, displaying photographs or manually drawn images on a computer display can hardly be considered computer animation. Computer animation is more about the computer's ability to generate graphics for display in each frame. By arranging the image sequence correctly, a computer can create the illusion of moving graphics; that is, computer animation.

1.2.2. Computer Animation

The computer can use stochastic processes, an existing image or another data structure to generate a frame of an animation. This work concentrates on computer graphics that are generated from mesh representations of three dimensional (3D) models through a rendering pipeline. The animation process, then, is about how the mesh is updated through consecutive frames to create a credible sequence.

As told above, a computer can generate a frame of an animation automatically from a data structure. Often, however, that this data structure itself is very complex and cannot be defined manually for each frame. In some cases, the data can be constructed procedurally. For instance, the animation of an object falling under the influence of gravity will simply have its Cartesian position updated in each frame, according to a simple formula.

Not all types of animations can be calculated procedurally. For complicated scenarios, a computer animation artist has to define the frame data. In such cases, for the very same reasons as in traditional synthetic animation, animators rely on keyframing to generate most of the frames automatically from example frames provided manually.

The computers do not have artistic talent like people do. They require well-defined algorithms to calculate inbetweens from the keyframes. To create the inbetweens, the values of the keyframe parameters are interpolated. Interpolation methods take each variable that is to be interpolated and treat its value as a curve along time (or frames), estimating it at the missing points. The first keyframe systems used piecewise linear interpolation but other interpolation methods were later used in order to provide desirable qualities like plausible continuity [1][2].

The early approaches to computer keyframing mimicked the traditional practices by calculating inbetweens directly from keyframe images. A basic implementation of this approach is to simply interpolate the color values in the keyframe pixels, which usually does not produce the intended result. An informed approach is to define corresponding points in each keyframe, whose positions are interpolated to create the inbetweens [3][4].

With the improvements in vector computer graphics and, in particular, the 3D rendering pipeline; using parametric keyframing became common [2]. Since graphical data structures are simply organized collections of numerical data, it is possible to operate directly on the numbers and have the rendering pipeline create the frame from the resulting structure. For a 3D scene, possible interpolation targets may be the positions of vertices, camera, or projection parameters. For instance, if interpolation is done on the positions of matching mesh vertices for each keyframe, animations for mesh movement, rigid transforms (translation, rotation etc.) and even deformations can be obtained.

1.2.3. Animation Control

So far, procedural calculations and keyframing have been described as the most useful techniques to generate frame data in computer animation. However, in practice,

procedural methods are only possible with the simplest animations. Complex interactions between objects and influences of numerous affecters require more than an animator's skills for describing the motion. Similarly reducing frame definition costs by keyframing may not always suffice, because the frame data may be too complicated to define even only for the keyframes.

Animation control is the umbrella term for the aids that animators have for high level management of complex frame data and description of animations [5][6]. By using controlling tools, the animators are able to specify the motion or scene configuration at a higher level of abstraction and leave the computer to do the detailed scene arrangements. For instance, the animator might define a bending animation for a mesh and the computer will arrange the positions of the vertices in each frame by the correct transformation. Note that when animation control is employed, keyframing is often applied to the control parameters and not directly to the parameters of the resulting mesh or image. In a bending animation, for instance, often the bending angle is interpolated, not the resulting vertex positions (which would result in unrealistic, skewed mesh configurations) (Figure 1.2). A few examples of animation control mechanisms are discussed below.

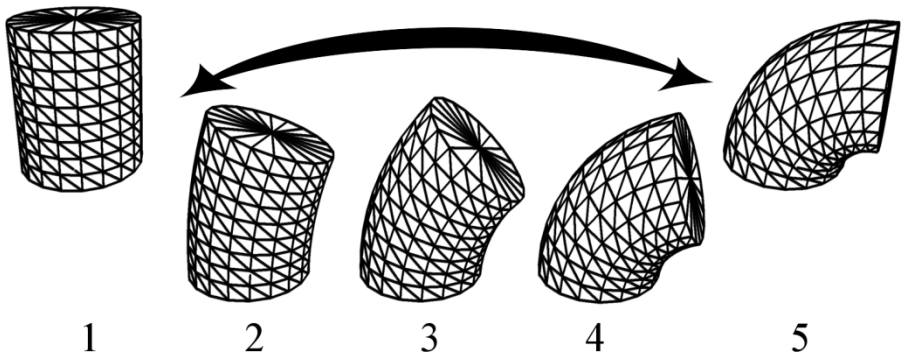

Figure 1.2. Bending of a mesh: The animator supplies the keyframes (1 and 5) and the computer generates the inbetweens (2-4) controlled through parametric interpolation. The interpolation parameter is the bending angle, not vertex positions.

1.2.3.1. Soft Object Deformation. Deforming an object, particularly a mesh, is difficult to do by hand, because the number of variables that describe the object (that would need to be individually specified for the deformation) is typically too large. It is usually not required

either. Most realistic deformations require the entire object to be deformed in a more or less similar way.

For instance, bending a mesh would require all the vertices to rotate around the same center by an amount according to (i.e. a function of) their position in the object. Hence a lot of the brute work can be left to the computer simply by asking the animator to provide the axis of bending and the angle. The nonlinear global deformation approach transforms the vertices of the object by a matrix that is a function of their positions [7]. This approach provides easy specifications of tapering, twisting and bending transformations for objects.

In [8], a local space around the target object is defined. The animator can then deform the space itself in a completely general way, called free form deformations (FFD), and the computer calculates the new positions of the vertices in the global space by substituting their local positions in the deformation.

1.2.3.2. Linked Hierarchies. Consider modeling the front wheel complex of a bicycle mesh. If modeled completely as a single solid mesh, animating the mesh would require the artist to reconfigure all the vertices in the body, the handlebar and the front wheel separately for each keyframe. Instead, it is possible to model the handlebar and the steering column as a separate submesh linked to the body (chassis) mesh at a particular position. Similarly, the front wheel itself can be modeled separately and linked to the handle column (Figure 1.3). The links can be completely defined with their positions and orientations along with a list of the movements they allow (in this case, the axes of rotation). In this example, the internal configuration of the complex is defined by only two rotation angles (one allowed rotation axis for each link) and the positions of all the vertices can be calculated, greatly reducing the effort required of the artist. Hence, such a model is said to have two degrees of freedom (DOF).

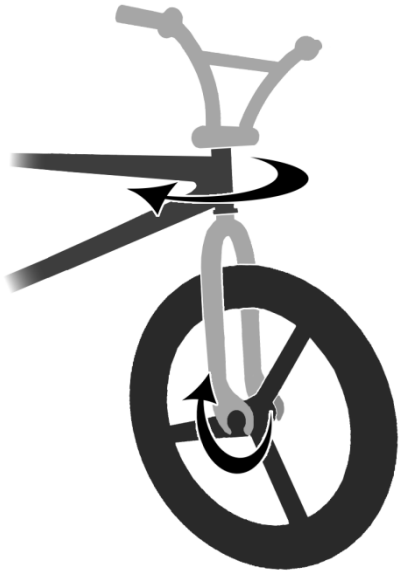

Figure 1.3. Part of a bicycle modeled as a linked hierarchy: The link positions and allowed movements (rotations in this case) are marked.

Modeling an object as a collection of linked submeshes instead of a single mesh provides another level of animation control to the artist by simplifying the way poses and animations can be specified. In such models, the submeshes are often hierarchically related so that moving or rotating a submesh would similarly affect all its descendants, preserving the integrity of the model. For instance, rotating the handlebar and the steering column in the above example would also rotate the front wheel along the same axis. If the submeshes in a linked hierarchy are physically connected, the model is also said to be articulated.

A well known use for articulated models is the human body. Typically, the animator would model a torso and each limb separately and link them at the appropriate joint positions. The upper arm would be linked to the torso, the lower arm to the upper arm, the hand to the lower arm and so on. The submesh hierarchy in a human model has a tree structure called the skeleton. A skeleton is comprised of a number of nodes (bones) connected by links (joints). A convenient bone is selected as the root bone and is used to position or orient the entire model. The bones away from the root in the hierarchy are called the child bones and those toward the root are called their parents. The root bone does not have a parent. Conversely, any child bone that does not itself have children is called an end affecter. The movements allowed by each joint are defined by a number of

transformation parameters, the number of which is called the DOF of the joint. The sum of the DOF's of all the joints in a skeleton is the DOF of the skeleton. A human model requires over two hundred DOF's to represent the capabilities of the joints of a real human. A vector containing a value for each of the DOF's of the skeleton completely describes a particular skeleton configuration, called a pose. Given a vector of joint parameters, the parameter vector, one can calculate the exact position of every vertex in the model.

Past work using skeletons include [4], which uses skeletons for transforming 2D images. Controlling articulated human body models with kinematics methods is described in [9] and [5]. The PODA animation system described in [10] and [11] also employs kinematics. In [12], [13], [14], [15], [16], [17] and [18], the inverse kinematics method is extended with features such as multiple constraints, interactive editing and analytical hybrid solutions.

1.2.3.3. Forward Kinematics. An animation artist can specify the joint parameters of an articulated object to fully describe its pose. From the joint parameters, the computer can generate joint transformations and calculate the position of each vertex by applying the transformations at each joint towards the vertex incrementally. This process is called forward kinematics method.

1.2.3.4. Inverse Kinematics. While forward kinematics is easy to calculate, it is not always practical to use. In most cases, artists are interested in directing the articulated body in an abstract, goal oriented manner. For instance, the artist may want the virtual human to reach for a door knob but not care about the actual joint parameters as long as they are physically plausible. However, the computer would still need these parameters to apply keyframing on and create the animation. Since by adding requirements such as reaching for a door knob, the artist effectively constrains the pose of the body, with enough such requirements, it may possible to solve for the parameter vector. This process is called inverse kinematics (IK). Of course with IK, a solution cannot always be guaranteed (e.g. a human model may not be able to reach the flag on a tall pole from the ground, however he positions himself).

1.2.3.5. Physics Based Methods. Since the laws of physics are relatively known, it is also possible to control animations with high level physics simulations. This approach not only lifts some of the work from the animation artist but also produces more realistic results. Physics based methods essentially rely on procedural animation techniques, governed by the descriptions of physical laws of the virtual world [19] [20]. These laws are used to simulate the object behaviors automatically. Such simulations are especially effective on simple rigid objects that are static (e.g. stones) and even work on those that are internally motivated (e.g. rockets).

It is also possible to apply physical simulations to the internal configurations of nonrigid or articulated models. For instance, the vertex positions can be determined by physically simulating the deformations of soft and elastic bodies and their collisions, as explored in [21] and [22]. For articulated bodies, physical simulations can be used to drive the joint parameters through methods collectively called dynamics, as explained in [23], [24] and [25]. Dynamics performance can be improved as explained in [26], which also extends the inverse dynamics method.

In dynamic simulations, the parameters are no longer the actual angles or positions of the joints but rather, the forces and torques affecting the joints. These parameters can be used to calculate the key skeleton poses, which can be subsequently interpolated for animation, or directly control the animation itself. Dynamics can also be regarded as yet another level of abstraction, for the object is described in terms of its physical attributes. Since simulation laws are based on observed real world laws and the models are defined with all the physical attributes they would have if they were real, the animations produced are often more realistic compared to pure kinematic designs.

1.2.3.6. Forward Dynamics. In forward dynamics, the poses of the articulated body over a period of time are constructed from the specifications of its internal and external forces and torques, effectively replacing the parameter vector of angles. This can produce very realistic animations but its usefulness is limited due to its lack of control. In other words, the animator can specify the entire model and its physical attributes, but cannot control its motion directly.

The lack of control may not be an issue for inanimate articulated bodies, such as chains, that are intended only to obey the physical laws [27]. However, automotive characters, like humans or robots, exert forces internal to their bodies and on their surroundings to accomplish any tasks they may have. For such models, the artist often has a planned motion the model needs to follow, and has to specify the internal forces exerted by muscles or motors of the model to generate it. This can prove to be labor intensive, because with forward dynamics, obtaining the desired motion is mostly trial and error.

1.2.3.7. Inverse Dynamics. Inverse dynamics is the arguably more useful dynamics approach. Similar to IK, in inverse dynamics, the animation artist describes the desired motion and the dynamics engine calculates the forces and torques necessary to produce the motion, which, in turn, can be used to produce animation.

1.2.3.8. Constrained Dynamics. Using inverse dynamics alone to figure out the forces necessary to generate a given motion is convenient, but relies on the animation artist's talent a lot while reducing the benefits of physical simulation. A hybrid method is to introduce kinematic constraints to regular dynamics simulations. The approach is loosely attempted in the PODA animation system of [10] and [11], and improved in [28] and [20]. The constraints would be used to solve for unknown forces in the dynamic simulation and then used in the simulation to generate a realistic motion.

The method is generalized in [29] and [27] to solving for entire trajectories instead of frames.

1.2.4. Human Body Animation

Modeling, displaying and animating the human body is a very old objective of computer graphics. As discussed above, control of human animation is commonly achieved through employing an articulated body, a skeleton, which can be directed through kinematics or dynamics methods.

For visualizing a human body, however, simply producing the rigid limbs of an articulated body is not enough. Humans, like all automotive animate entities, are covered with an elastic skin. Without such a skin, the displayed object would always look robotic (inanimate). This would suffice for a robot, but a realistic human model must incorporate skin deformations as well.

Historically, the first structured human models were more valuable for their use in practical simulations than their realistic look. In fact, during the early 1970's, human body animation attempts merely used stick figure representations. Only later, realistic human displays were added to the simulations. For instance, the work described in [30] explores how car crash victims and parachuters can be simulated on computers with somewhat realistic body models.

The more modern human body representation methods are largely categorized in two groups. With volume models, the body is divided into (sometimes numerous) 3D primitives such as cylinders, ellipsoids or spheres [30][31][32]. Rendering the final image, then, consists of obtaining the contours of the primitives; but shading is relatively difficult. An alternative approach is using surface models employing patches or meshes [33]. With these methods, rendering can be made more realistic but more time consuming, not to mention the undesirable artifacts with some approaches. The available methods of the time are summarized in [34]. As computer and graphics hardware grew in power and solutions were proposed to the problems, mesh surface representations gradually dominated along with layered models extending the surface approaches, rendering the volume models virtually obsolete.

1.2.4.1. Layered and Physically Based Methods. Layered methods usually employ some sort of representation for the internal structure of the body and use it to drive the surface forms. While more realistic, these methods typically require more numerical processing and hence, are often not interactive. First muscle simulations were added to existing skin models as explained in [35] and [36]. The work described in [37] is a system for creating human body models by easily specifying volume features (muscles) as well as a surface skin model. In [38], [39] and [40], the authors investigate the effects of anatomically accurate muscle models on skin deformations, with the latter also providing such a generic

human model. [41] describes implicit surfaces for applying coating layers to a model, such as skin over muscles. The work discussed in [42] adds a new fat layer to the anatomical models and links to it an inelastic skin surface, instead of using an elastic model.

1.2.4.2. Skinning Based Models. These models define the human body to be an empty shell implemented as patches or a mesh and use the skeleton to drive the deformations of the shell. With such a model, the shape of the body (shell) is completely artist designed and then deformed according to the position specified by the skeleton pose inside (Figure 1.4). This approach is more common with interactive applications, but it is also known for its production of characteristic skin deformation artifacts when handling particular joint configurations. In this work, we follow common practice by using a skinning based surface model driven by an internal skeleton.

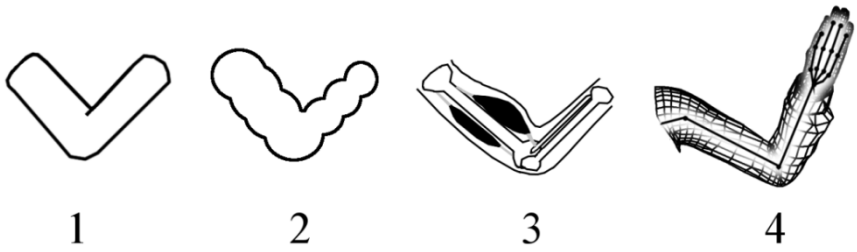

Figure 1.4. Demonstration of the human body representation methods on a human arm: Contours of a volume model with cylindrical (1) and spherical primitives (2), a layered model (3) and a skin model (4).

[43] describes how a mesh surface model (virtual skin) can be used in conjunction with a controlling skeleton. In [44], a variety of human movements are studied on anatomically realistic mesh bodies with kinematic and dynamic control. The work described in [45] uses the character's skeleton to drive an FFD model which, in turn, drives the deformation of the actual skin. The common technique is named skeleton subspace deformation (SSD) in [46], where its shortcomings are studied, and to solve them, using examples is suggested. The example approach is extended in the work described in [47], which also suggests ways to reduce the interpolation space. The method employed in [48] uses principal component analysis (PCA) to reduce the pose space and makes interpolation

and SSD suitable for hardware implementations. In addition, to fix the SSD artifacts, using 2D limbs (medials) is proposed in [49], and multiple weights (for matrix elements) in [50].

According to [51], dissimilarity of consecutive rotations are to blame for the SSD artifacts and to smooth the transitions, new joints can be inserted. In [52], it is proposed to interpolate the bones themselves and transform the vertices without further blending. Arguing that matrices cannot be directly interpolated, in [53], interpolation on joint rotations is suggested, using spherical linear interpolation (SLERP). In [54], it is shown that common interpolation methods for SSD effectively reduce to direct interpolation of matrix elements, which causes the artifacts. This verifies SLERP as a better interpolation method, for which an optimization is also proposed, called linear interpolation of quaternions (QLERP). Alternatively, in [55], it is suggested to use dual quaternions [53] with two components per element to solve artifacts and achieve better performance.

1.2.5. Motion Capture

As it was previously stated, keyframing in computer animation, as in traditional animation, is mostly necessary because of the excessive amount of work required to define even a single frame of an animation sequence. However, there is always a tradeoff with keyframing; while the more frequent they are defined, the more credible the interpolated animation looks. This is due to the fact that interpolation is basically an uninformed approximation to the motion the artists intend to create. Moreover, even with animation control mechanisms, such as using skeletons, defining the parameter vector (e.g. for a keyframe) is still a tedious task. Often, the animator can only specify part of the parameter vector and the rest need to be discovered through a solution method like IK. Finally, the credibility of keyframe animation is ultimately limited by realism the artists can provide to the keyframes themselves, regardless of the performance of the interpolation or solution method applied. Fortunately, if the animation is supposed to represent the behavior of real objects or characters, an alternative method is available.

Motion capture (MOCAP) is an approach that can be used to discover the control parameters for a frame of an animation automatically, bypassing manual definition of

keyframes and IK. Moreover, modern MOCAP equipment is fast enough to capture control parameters in realtime for every frame, making interpolation unnecessary as well. In addition, since the parameters are captured from a real subject, the derived motion data are guaranteed to be realistic. When applied with care, MOCAP promises practical authoring of realistic motion definitions.

By far the most common use of MOCAP is defining the gestures of virtual humans. Typically, a real actor would perform the required body movements, possibly while wearing some sort of smart suit, and some detecting equipment would record the derived control parameters for the motion, usually the positions and orientations of the joints.

MOCAP techniques can be broadly categorized either as passive or active depending on the underlying technologies (Figure 1.5).

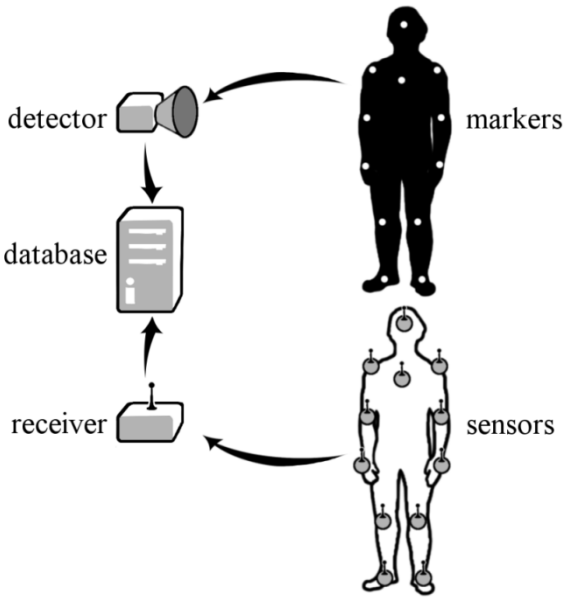

Figure 1.5. Passive and active motion capture: The system at the top is a passive MOCAP system that uses markers that are detected by an external device. At the bottom is an active system using sensors that measure the position and transmit the information to a receiver.

1.2.5.1. Passive Motion Capture. Passive motion capture can be characterized by the lack of data collecting sensors. Instead, passive methods rely on other equipment to derive a description for the actor's pose. The most common of these are the optical methods.

The optical approaches to the motion capture problem originate from the rotoscope devices of the early 1900's [56]. The first MOCAP implementations used image processing techniques coupled with prior knowledge about the human body to recognize human poses from images [57][58][59][60]. Such attempts are somewhat less accurate but have the benefit of being non-invasive, which makes them more useful for surveillance applications [61][62].

The lack of accuracy of image processing based methods is not acceptable for control or analysis applications. For such scenarios, the common solution is to have markers of some sort be worn on the actor's body. The markers can be either reflective or emissive. With markers, the human body is assumed to be a stick skeleton and the positions of the markers, as tracked by cameras, are used to triangulate the 3D positions of the appropriate limbs. [63] contains an excellent overview of the method as well as some historical perspective into the original moving light display concept described in [64] and [65].

[66] describes another passive motion capture approach which replaces the optical markers with sound emitting devices. The relative positions of the devices can be estimated from the differences in the delays the sound from each device takes to reach a receiver. The 3D limb positions can then be similarly triangulated.

1.2.5.2. Active Motion Capture. An alternative to the passive approaches is to use sensors. In this case, sensors actively measure their own positions and orientations, and transmit them to the recording equipment. This not only avoids the problems with the passive methods (such as occlusion of optical markers) but also decreases the dependence on prior skeleton specifications. The sensors can be magnetic [67], gyroscopic [68] or even electromechanical, collectively forming an exoskeleton [69][70]. In this work, we use a magnetic MOCAP system.

1.2.5.3. Magnetic Motion Capture. Magnetic trackers were initially developed for the helmet mounted displays of military aircraft during the 1960's [56]. These sensors detect the position and orientation by measuring the low frequency electromagnetic field generated by a transmitter source. Each sensor outputs 6 DOF's, making the system as

competent as an optical system with 50 per cent more markers than magnetic sensors. Compared to the optical systems, magnetic systems are also typically less expensive. Unlike electromechanical systems, magnetic systems do not require the actor to wear cumbersome equipment, but still achieve considerable accuracy. In terms of latency, magnetic systems generally fall in between the slow optical systems and fast electromechanical systems, but often have lower sampling rates due to the noise filtering required.

Among the shortcomings of magnetic systems are their recalibration requirements, limited range and nonlinear behavior near the limits, especially as the azimuth approaches zero. These problems arise from the nature of magnetic fields. In addition, magnetic fields are vulnerable to intrusions by other magnetic fields that may be in the environment. Besides the magnetic field of the Earth, most electrical equipment (motors, cabling etc) can cause magnetic interference. Even at the absence of other magnetic fields, the magnetic field of the transmitter itself can induce eddy currents in the surrounding metals (especially ferrous metals such as iron and steel) which can interfere with the field. All of these problems are reduced by using AC fields instead of DC fields, but not completely negated [56].

1.2.6. Sign Language Synthesis

It was argued earlier in this document why computer visualization of sign language communications is a useful research task. This is not a recent realization. The problem has found attention earlier, but it was not until the 1980’s that computer hardware grew enough in power for true multimedia solutions. Many of the earlier systems aim to provide full machine translation between text (and sometimes audio) and sign languages. Such a task would obviously require sign language recognition (e.g. from video) components for the return path and a linguistic structure for the languages in question. Such extensions are out of the scope of this work; we concentrate, instead, only on the synthesis of graphic visualizations of sign sequences, focusing primarily on visual quality.

1.2.6.1. A Generic Sign Language Synthesis System. There are two input requirements for sign synthesis (Figure 1.6). The first is a list of signs to synthesize. How this list is constructed is completely dependent on the nature of the application. A realtime translation system would likely create the list by first processing speech, while a book reader application would be parsing text as input. The format of the list itself is also relevant. Often, it is designed in a generic list or script form, but in translation contexts, it can also be an intermediate language representation of the original input.

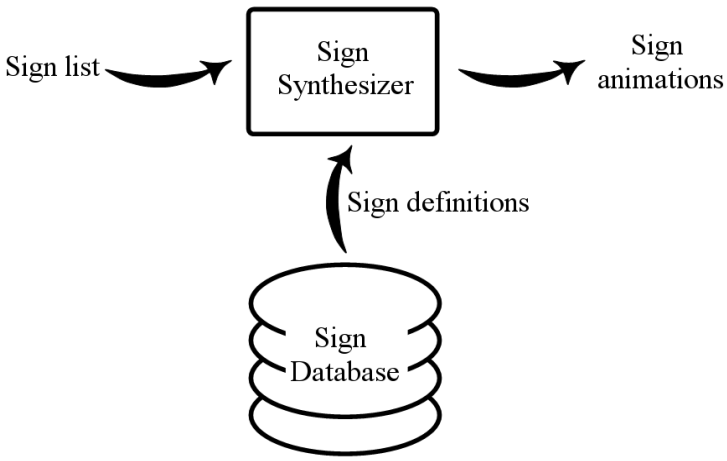

Figure 1.6. A generic sign language synthesis system overview: The synthesizer module receives the list of signs to generate from another module and looks up the definitions of the signs from the database.

The second input to sign synthesis is a database of sign definitions to generate the animations from. This database essentially forms the lexicon for the possible outputs of the system and typically contains video samples or MOCAP data. With the former approach, proper blending of video data at frame and pixel levels is necessary to produce smooth movies, resulting in heavy computational requirements. In the latter case, a post processing step is necessary in which the raw animation data are visualized on a virtual human actor. In addition, some applications use sign definition scripts instead of either video or MOCAP. These scripts describe the signs in terms of more trivial predefined movements, effectively demoting the database to a sign alphabet, but often only specify the movements on relevant body parts, not the entire model. Hence for a final animation to be produced, the missing parameters need to be discovered through a process like IK. Scripts are notably more practical for editing afterwards, but they are also known for their artificial look.

A sign synthesis system can output animation in one of several formats. Structured numeric formats, such as VRML, are common but also the least dependable in terms of realism. Blended video sequences and rendered visualizations of 3D models are known to be more successful from that point of view.

1.2.6.2. Previous Sign Language Synthesis Systems. An early attempt at computer assisted sign language processing is described in [71]. The system employs an optical MOCAP system, conceptualized by [64], to drive the synthesis process but does not output any real synthesized signs, but merely the observed trajectories.

One of the first true sign synthesis systems is described in [72]. This is a complete translation system which features a voice recognition frontend to drive the synthesis process. The database consists of motion definitions captured by an optical MOCAP system and data gloves. The system generates signs on a 3D model driven by a skeleton identical to the one used in MOCAP.

In [73], the TEAM machine translation system for English is described. This system is driven by English text and generates script-specified 3D signs.

The SigningAvatar system developed by VCOM3D (formerly Seamless Solutions Inc.) is one of the first commercially available solutions for authoring 3D skeletal animations [74]. It also supports sign languages through a dedicated proprietary authoring tool. Note that this system does not produce rendered output. Instead, virtual reality modelling language (VRML) output is produced that can be rendered by compatible software.

The ViSiCAST project and other related work are described in [75], [76], [77], [78] and [79]. These projects describe a machine translation system that generates signs from English text. The signs themselves were originally specified through an assembly involving a magnetic MOCAP system and data gloves. However, later work focuses on generating the signs synthetically by script definitions. The work was funded by Independent Television Commission, UK Post Office and the European Union.

The work described in [80] does script driven synthesis of signs. However, it is favored to use video samples to generate the sign sequences, with the argument that they produce more realistic results.

The sign synthesis problem is approached as an articulatory process, instead of a concatenative one, much like speech synthesis, in the system described by [81]. The system is driven by a scripted list of movements (not complete signs) and generates a VRML output defined on a simple 3D model.

In [82] and [83], the HANDY system is described, which features a database of movements (sublexical sign elements) that are obtained through optical MOCAP. The synthesis is driven in a similar way to the method described in [81] except here, sublexical elements are used in an XML based script, producing rendered output based on a modified form of the model described in [40].

In [84], a translation system for the Czech Sign Language is described. The system receives text input and translates it, producing 3D rendered sign language animations. The signs themselves are defined with the same scripting system used for the ViSiCAST project [75].

The system described in [85] is the first to incorporate Turkish Sign Language support. It is a learning tool that recognizes signs from video input for demonstration. For feedback, it also synthesizes sign sequences from a database of manual specifications on a simple 3D humanoid. The system employs the help of the Xface project for the animation of the face [86], [87] and [88].

A brief summary of the sign language synthesis systems discussed above is given in Table 1.1. The table highlights the unique features of the systems and classifies them according to the formats of the original input to the systems, the sign definitions in their databases and their products.

Table 1.1. Summary of previous sign language synthesis systems

System References	Name	Sign Language	Original Sign List Input Format	Sign Definitions Format	Sign Animations Format
[72]		Japanese	Speech	MOCAP	3D Render
[73]	TEAM	American	Text	Scripts	3D Render
[74]	Signing Avatar	American	Generic	Scripts	VRML
[75] [76][77][78] [79]	ViSiCAST (Simon the Signer, TESSA, e-Sign, HamNoSys)	British & Others	Text	MOCAP & Scripts	3D Render
[80]		Slovenian	Generic	Video	Video
[81]	SignSynth	American	Text	Scripts	VRML
[82][83]	HANDY	Hungarian	Text	MOCAP	3D Render
[84]	MUSSLAP	Czech	Text	Scripts	3D Render
[85]	SignTutor	Turkish	Generic	Scripts	3D Render

1.3. Contributions

The importance of demonstrations in language learning and the need for sign language tutoring systems were stressed above. As the background research suggests, a complete sign language tutoring tool with ample demonstration capabilities is a complex challenge. Instead, this work aims to provide a practical solution to the more isolated problem of producing realistic sign language animations. While trading off a broader scope, dedicated attention can be paid to the realism of the output. However, it is the intention of the author that the software delivered as a result of this work can eventually be used as a frontend to a sign language tutoring tool.

The implemented system incorporates a sign definition database which lacks linguistic structure except for identifying tags. As the literature on the subject matter indicates, linguistic analysis of a sign language is often considered together with the data

originally driving the synthesis process, usually some form of natural language input, while an intermediate language is designed. Hence, it is assumed to be a role reserved for full translation systems and is omitted from this implementation. This also has the added benefit of making the system transparent to any sign language. Yet, the current database contains signs from the Turkish Sign Language finger spelling alphabet. Given the scarce interest toward this particular language, it is understood that the database itself is an asset as well.

2. APPROACH AND METHOD

2.1. Arguments

2.1.1. Realism

Among the qualities of a sign synthesizer is the realism of its output. Realism is a definite requirement if the objective is teaching of or communication with sign languages.

Intuitively, using videos of real signers in the sign database and composing the output directly from these videos should produce the most realistic results. With the latest advances in computer graphics, using 3D virtual actors can also be considered a good alternative. The animation of the 3D actors can be driven with MOCAP data or scripts. Using scripted sign definitions has its benefits, but usually produces animations that look less natural while MOCAP data is collected from actual signers [72][80].

2.1.2. Performance

As with all software, performance is another issue with sign synthesizers, especially for the availability of the system. Realtime performance can enable a whole new set of possible applications (e.g. simultaneous translation). In this context, realtime means achieving not only immediate response, but also motion picture frame rates in synthesizing and playing sign animations. Note that performance may especially be an issue with older hardware or for future porting considerations to low power devices, such as mobile phones.

Today, with proper hardware acceleration, video handling is usually not a problem but blending between the signs is not trivial [80]. On the other hand, 3D hardware acceleration is also widely available, even in mobile devices. With proper handling of the hardware, animating virtual actors can be a feasible option. Scripted sign definitions should also benefit from the hardware acceleration capabilities, but they require additional IK processing to solve for the skeleton parameters that are not explicitly specified [81].

2.1.3. Communication

If the software is to be useful in networked environments (e.g. the World Wide Web), communication requirements have to be addressed. Typically, communications of the sign synthesizer with the database and with the receiving end are affected (Figure 1.6).

If the sign database is formed from sign videos, bandwidth requirements for communication with the synthesizer would be great. On the other hand, scripted sign definitions would likely require the least bandwidth. The case is similar for the outputs. If the system produces video (video sourced or 3D generated), the bandwidth requirements would definitely be greater than that for producing VRML compatible animation definitions that can be rendered locally.

2.1.4. Data Reusability

Since a sign synthesis system has to incorporate a database, how the data in this database can be exploited is also an item to ponder on. A sign video is only externally marked, but does not contain any structural information inside; it is just a video that happens to contain signing content. A MOCAP database has somewhat better structure, but the data is still difficult to edit, reorganize or apply to another system (e.g. virtual reality applications, games etc.). Scripted sign definitions have the most internal structure and are best suited for data reusability.

2.2. Approach Decisions

In the light of the above arguments, we have opted to keep a database of MOCAP sign definitions obtained using a magnetic system. Magnetic MOCAP systems are generally more accurate than optical systems, which is important for capturing detailed finger motions. They are also more comfortable than electromechanical systems in that they require lighter, more flexible setups, which is necessary for our actor to produce natural sign animations.

Our system uses a synthesizer to drive a 3D virtual actress to produce rendered output. Rendering our own models is a safer approach, since client renders, especially those by the browsers, may not produce realistic outputs. These choices, in our opinion, give the best compromise of the qualities we see as significant.

2.3. System Overview

Figure 1.6 summarizes the structure of a typical sign language synthesizer system. The way this prototype system is implemented in our work is depicted in Figure 2.1. The operation of our system can be explained in two phases.

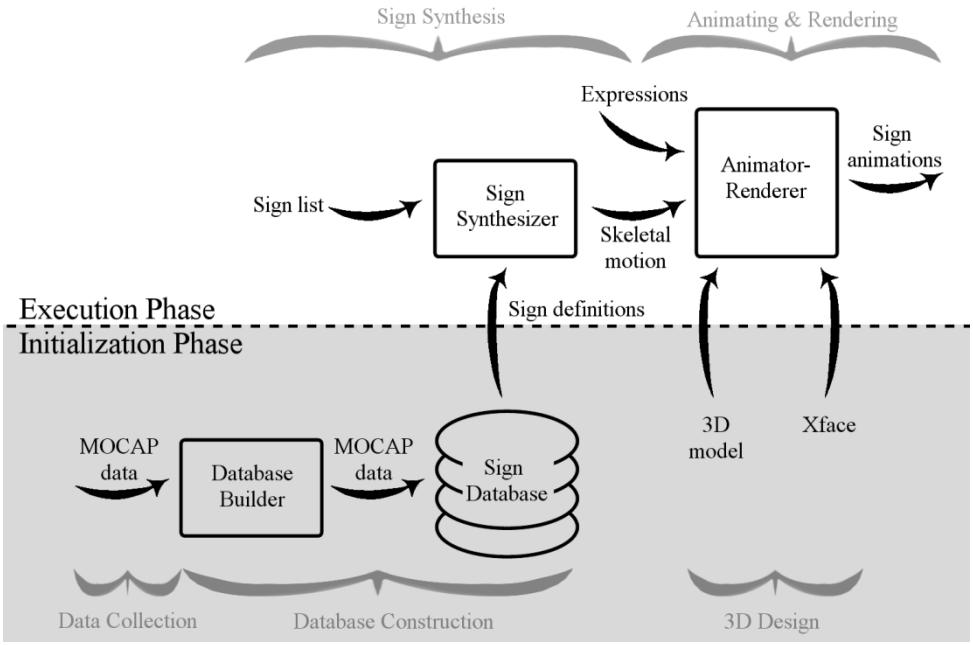

Figure 2.1. System overview: The initialization phase consists of data collection, database construction and 3D design tasks while the execution phase includes sign synthesis and animating & rendering tasks.

2.3.1. Initialization Phase

During the initialization phase, system operators define the application data files for use during the execution phase. This phase must be completed before the system can be

made available to the public. Among the work belonging to this phase, three separate tasks can be identified. First, the data collection task is done to obtain separate MOCAP data files for each sign. These data files are fed to the database builder module for the database construction task. After some preprocessing and cleanup, this software collects the MOCAP data into a sign database. Meanwhile, 3D artists carry out the production cycle of the two main digital assets for the application: the body model for skeletal animation and the face model for expressions. When all the information is properly stored, this phase can be called complete. Note that the initialization phase for a preset configuration and feature set is already completed. The database in this configuration contains signs from the Turkish Sign Language finger spelling alphabet. The initialization phase is detailed in section 3.

2.3.2. Execution phase

This phase follows the initialization phase once all the necessary data are available and is characterized by end user access to the system. It is a cyclic phase in which the user issues sign queries, to which the system responds by carrying out the two tasks required to generate the sign animations. For the sign synthesis task, the synthesizer module fetches the definitions of the desired signs from the sign database and concatenates them as necessary, producing skeletal animation specifications. Next, for the animation & rendering task, these specifications are fed to the animator-renderer module which, in turn, drives the virtual actress as necessary and renders the resulting mesh to obtain a realistic animation. The execution phase is detailed in section 4.

3. INITIALIZATION

The objective of this phase is preparation of the system for actual, in-field usage. The most significant part of this preparation task is construction of the sign database. For this purpose, a motion capture task was defined to initially collect sign data in separate data files. The database can be called ready once this data is loaded and bound to the skeleton of the virtual actress by our dedicated database builder application. This application is basically a standard database editor with a simple 2D interface.

3.1. Motion Capture of Turkish Signs

3.1.1. Hardware Setup

We used a Shapewrap III system by Measurand Inc. for our MOCAP tasks [89]. This system is not only magnetic based, but is also capable of tracking all the limbs we require for sign languages. In contrast, for example, [72] and [76] use MOCAP systems that do not track the hands and use data gloves instead, which adds synchronization problems to their implementations. Our system supports wireless communication and does not require a spandex body suit to be worn and hence is most practical for expecting realistic sign data from our actor.

The skeleton of our MOCAP system contains 40 joints. The root has six DOF's (Euler orientation and Cartesian position) and the rest of the joints have three DOF's each (orientation only), adding up to 123 DOF's in total. The missing DOF's (three for each joint except the root) are fixed into the skeleton design, since real bones cannot be resized when moving. The detail in the articulation of the hands is worth mentioning; all the fingers have a full set of three bones each. In addition, note that although our MOCAP system also features leg sensors, they were not used because leg movements are not relevant to signing. The skeleton is visualized in Figure 3.1.

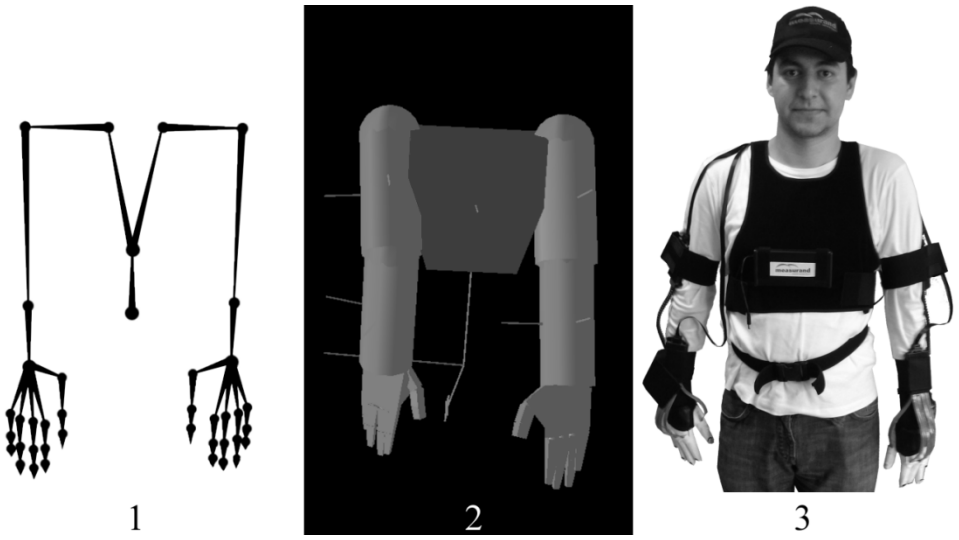

Figure 3.1. Motion capture model of the system: Note the detailed articulation of the hands. 1 is the underlying MOCAP skeleton, 2 is the realtime preview of the MOCAP model in the software and 3 is our actual actor wearing the Shapewrap III system.

The MOCAP hardware was connected wirelessly to a PC with dual core CPU running at 2.4 GHz and 3 GB of RAM. OEM software, ShapeRecorder, was used to record the motions in standard Biovision Hierarchy (BVH) format. Running Microsoft Windows, this setup managed to record joint parameters at 77 fps (frames per second).

3.1.2. Sign Scope

As an initial data set, we aimed to capture the signs for the Turkish sign language finger spelling alphabet (Figure 3.2). This not only limits the work required before the second phase, but also enables a much simpler interface and format in specifying the sign list. However, the system is not limited to this set of signs and can easily be extended with more data.

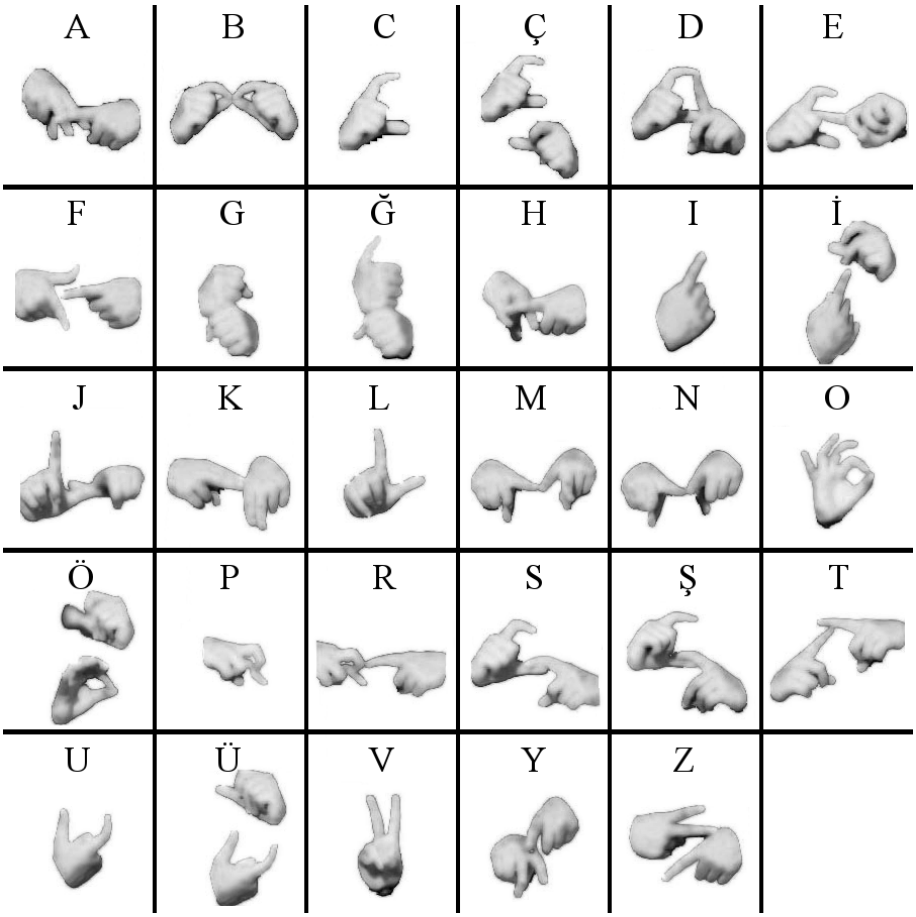

Figure 3.2. Turkish sign language finger spelling alphabet.

3.1.3. The Capture Process

3.1.3.1. Preparation. Prior to our MOCAP sessions, we first prepared the equipment by adapting the dimensions of the predefined MOCAP skeleton template to our actual actor. This ensures the accuracy of the captured data. For this task, it is necessary to take specific measurements from various limbs of our actor and declare them to the MOCAP software. Fortunately, the software features a wizard that guides this task in a step-by-step manner. Note that this process has to be done only once, since the measurements can be stored and reused for each session.

3.1.3.2. Testing. Before capturing the signs to be stored in the database, the hardware setup, preparation and our virtual actress' dimensions were tested by capturing predefined poses on our actor and visually checking if they are reproduced on the 3D preview of the

software and by our virtual actress. Problems evident in the preview were fixed by repeating preparation or calibration as necessary, and virtual actress problems were fixed by manually rearranging the model or noting and compensating for the inaccuracies while recording. The test poses are designed to check the relative dimensions of parts of the body and the accuracy of the motion by having the actor bring various limbs in contact at certain angles, such as those in Figure 3.3.

Figure 3.3. The basic test poses: 1-3 test the calibration of the hands and the arms while 4-5 test the compatibility of the Xface and the MOCAP virtual actress models.

3.1.3.3. Calibration. One typical weakness of magnetic MOCAP systems is that they are prone to lose their calibration quickly. Calibration involves specification of the internal joint correction offsets of the software to match the MOCAP data, which, unfortunately, does not remain consistent for a long time. For this reason, before capturing each sign, the calibration of the equipment was checked by visually matching the detected pose of the actor with his actual pose using the realtime 3D preview produced by the software (Figure 3.1). If there a mismatch is detected, calibration is restored by having the actor assume a predetermined home pose and signaling the software to reset its internal definition of this pose accordingly.

3.1.3.4. Standard Positioning. In our work, we capture the signs individually, and later concatenate them to obtain a sign sequence. However, while motion capture is known to produce realistic movement descriptions, concatenation is not. Possible problems at the sign transition points include returning to the rest position between every sign and sudden jumps of the body or the limbs at the transition points.

The jump problems can be solved if the signs take the same pose at the transition points. Since the signs can be ordered in any fashion, this requires defining the signs at the same pose for their start and end points. Hence, before recording a sign, the actor takes a standard rest pose, to the possible accuracy.

It is generally desired that the virtual actor does not return to the rest position between the signs. Such an animation would look rather robotic. An easy solution to prevent this effect would be to blend the signs in the output animation. Note that blending would also solve the jump problems mentioned above, but the movement rate during blending is more homogeneous when the rest positions are alike.

3.1.3.5. Attack and Decay. Since blending will be performed on the captured signs, the blended parts of the motions will lose some of their details. To prevent loss of the expressive parts of the signs, attack and decay periods were defined as the first and last 1 second parts of the captured motions respectively. The expressive part of a motion not belonging to either period is referred to as the body of the motion (Figure 3.4).

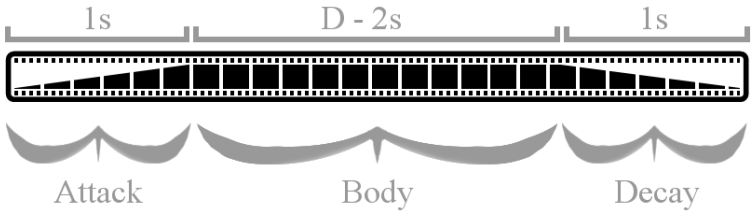

Figure 3.4. Structure of a sign: The first 1 second is the attack, the last 1 second is the decay and the rest of the sign in between the attack and the decay is the body. D represents the duration of the sign.

During the attack period, the signer starts moving his limbs from the rest pose. Likewise during the decay period, the signer is only allowed to return to the rest pose. Blending will be discussed later in more detail.

3.2. Construction of the Sign Database

A special database builder was developed to construct a database of the signs that can be later queried by the sign synthesizer. Using the database builder involves the following steps.

3.2.1. Model Acquisition

When first launched, the software builder requires that the user specifies a compatible model file to use as the virtual actor. This step is not directly related to the database construction (except a future implementation of retargeting), but is necessary because the motions defined on the virtual actor have to be specified on its skeleton. For the same reason, the storage for motion data is integrated into the skeleton file. When a model is first loaded, any signs already in its database are used to initialize the environment.

3.2.2. Sign Collection and Tagging

Once the software environment is initialized with a model, the user must specify the captured sign animation files to insert them to the active collection, or delete existing ones as necessary.

When querying the database for sign animations, the sign synthesizer requires a way to identify each sign. The current implementation of the database supports string tags to be specified for each sign and index the database accordingly. For our limited database of the Turkish sign language finger spelling alphabet, the tags for the signs were selected simply as the letters represented by the signs.

3.2.3. Sign Repositioning

Once the active collection contains the necessary signs, the user can choose to build the database file. However, before building the database, the software can do some preprocessing on the sign animations. In particular, the model can be repositioned.

Regardless of the actor's effort on standardizing the start and end poses, fixing the model at a particular position in all the frames may be desirable. First, this would make it possible to place the model at the appropriate location in relation to the camera during rendering. Second, it would remove the floating effect that would otherwise be produced by blending different model positions at the transition points.

To reposition the model, it is necessary to reset the root joint parameters appropriately. Unlike the other joints that have three DOF's, the root joint has six, including position as well as orientation. In our application, we simply reset them all to zero, positioning the model at the origin and aligning it with the principal axes, facing the +Z direction. This also prevents the model from moving or turning away from the camera while signing.

Note that the parameters of the other joints are not affected by this process. They will continue to drive the limbs. At any rate, they work better with blending because limb movements are expected in sign animations, but floating is not. The only downside of the repositioning process is that it removes from the capture data any navigation the virtual actress would make in the available space, but such movements are not necessary or desired in sign animations.

A sign can be defined as a sequence of poses changing over time. Since given a skeleton, a pose is just a vector of joint parameters, it is possible to represent a sign as a function P=S(t) where P is the parameter vector for the sign at time t. Then, the ith parameter of the vector can be defined as $p_i=S^i(t)$. Then, the repositioning process can be described by redefining all the signs as follows.

$$S^i{}'(t) = \begin{cases} 0, & \text{if } 0 < i < 7 \\ S^i(t), & \textit{otherwise} \end{cases}$$

where the first six parameters are known to represent the position and orientation of the root bone. Once the signs are redefined, the database file is written out.

3.3. 3D Model and Face Design

3.3.1. The Model

It was stressed above why 3D visualization is preferable in sign demonstrations. Therefore, this work includes a 3D virtual actress to display the sign animations on (Figure 3.5). It was also argued before that skinning based surface models, rather than layered representations, are preferred for representing virtual humans in interactive applications. Hence, our actress was modeled as a mesh surface model whose deformations are driven by an underlying skeleton, designed to be identical to the MOCAP skeleton.

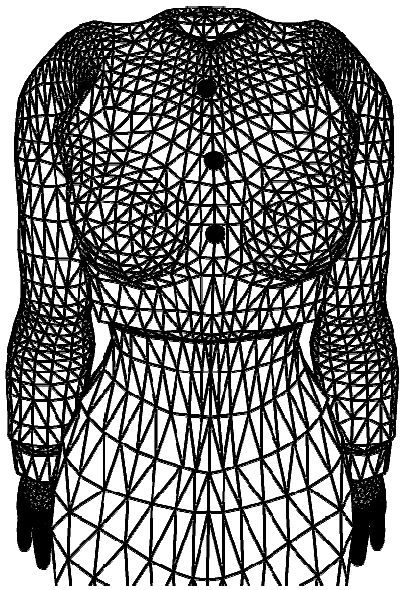

Figure 3.5. The virtual actress: The model has a mesh surface model that is driven by an internal skeleton, identical to the MOCAP skeleton.

The mesh of our virtual actress has 6,875 vertices and 7,128 faces and was prepared using Alias Maya modeling software. The details of the 3D modeling methods are outside the scope of this document and will not be discussed further.

3.3.2. The Face

Since sign languages make extensive use of facial expressions and mimics, a complete sign animation system needs to reproduce facial expressions as well as body movements. While this work incorporates direct support only for body movements, it also includes an integrated Mpeg4 compliant dynamic face engine, Xface [86][87], to enable future extensions for facial expressions. Xface uses face meshes generated by FaceGen software and implements Mpeg4 facial animation (FA) standard driving the facial definition points (FDP) of the model. It supports multiple blended facial expressions through morphing and keyframing.

This work uses the standard face model distributed with the Xface package, Alice (Figure 3.6), containing a total of 6,887 vertices and 13,412 faces.

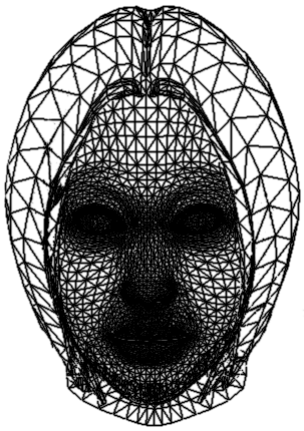

Figure 3.6. The Xface face model, Alice.

Xface is dynamically integrated into the system during rendering and attached to the chest joint of the model skeleton. This does not allow neck movements, but instead, makes

the face follow the chest. We believe this is an acceptable compromise, since there is currently no input from the MOCAP hardware to drive the neck separately at the point.

Note that the face mesh is not actually connected to the body mesh. It is merely linked so that it is displayed with the body at the appropriate location.

3.3.3. Model and Face Merging Issues

Our data set, the Turkish sign language finger spelling alphabet, does not require the use of head gestures or facial expressions. However, sign languages often employ them in connection with the body movements, sometimes with interactions as well. For instance, a sign might involve the left thumb touching the chin. With our system, the more basic task of supporting expressions is as simple as providing a data stream to the Xface engine. However, to handle interactions between the body and the face correctly, care must be taken when merging the body model and the face together. In particular, the position of the head and its dimensions should be compatible with the body. For this task, we simply measured the head features during the initial preparation and testing tasks, and transformed the Xface model accordingly. This does not ensure an exact match, but provided visually acceptable results during our tests (Figure 3.3).

A virtual signer with a head should also support head rotations. The most straightforward way to support these motions would be defining a neck joint that would only drive the head. Then, the neck joint can be provided with a data stream appropriately. However, the lack of actual connectivity between the polygons of the Xface mesh and the body would still cause these movements to look unnatural, because none of the polygons in the body would deform in response to the head, not replicating the elastic behavior of the skin at the junction. This problem can be solved by manually creating new polygons between the two meshes at runtime and defining them partial weights to follow the neck joint and the parent of the neck joint accordingly.

4. EXECUTION

The two software components used in this phase, the sign synthesizer and the animator-renderer are integrated into a single Microsoft Windows application based on OpenGL graphics. The user interface is mouse driven, used to access the graphical user interface (GUI) controls. In particular, the user is provided with an edit box to type a list of signs to play, a read-only text box indicating the sign that is currently played by the actress and two buttons; a play button to initiate playback and another to quit the application. This interface can also be driven by the keyboard through shortcut keys (Figure 4.1).

Figure 4.1. The user interface of the synthesizer-animator-renderer module: In the background is a 3D display of our virtual actress and in the foreground are the GUI controls; the edit and the read-only boxes above, and the play and quit buttons below.

Besides the 2D GUI, it is also possible to switch to 3D navigation mode via the control key or the right mouse button. While in 3D navigation mode, the user can manipulate the camera using the mouse or the keyboard.

When launched, the application presents the user with a virtual actress shown at an idle stage. When idle, the actress performs an idle loop from the motion database.

When the play button is pressed, the application first builds a list of signs by parsing the user's input in the edit box and fetches the sign definitions from the database. Next, the two primary tasks of this phase are executed in order: to synthesize a single continuous animation that will drive the model skeleton, the sign definitions are concatenated together with appropriate blending and to drive the deformation of the virtual actress mesh, a rendering loop is used, after posing the skeleton for each frame according to the animation. When playing the sign list is complete, the application returns to the idle mode.

4.1. Sign Synthesis

The sign sequence animation is formed by concatenating sign definitions fetched from the database in order. When concatenating, the frame data (parameter vector) of the animation is copied from those of the constituent signs, offset in time as necessary.

When MOCAP data are used to synthesize sign sequences, there is little concern over the realism of the produced animation. In fact, the only places where realism may be lost are the transition points between the signs.

4.1.1. Jumps

The most important problem that can occur is visible jumps in the actor's pose at the transition points, caused by direct concatenation of the signs with different start and end points. Continuity of the motion (more precisely, C^0 continuity) can be restored by simply having the signs start and end at the same poses.

4.1.2. Restarting

The problem with matching all start and end poses is that it would cause the virtual actor take this particular pose between every sign. However, if, at the end of a sign, the

next sign is known to him, a real actor would just make a smooth transition to the first movements of the next sign instead of going back to the rest pose and starting over. The virtual motion can also be made to look smooth by blending the motions of the signs at the transition points. Note that this would also solve the jump problem, but the signs were still captured with similar start and end poses as this produces more natural transition speeds when blended.

4.1.3. Blending

To be able to blend, we make use of the attack and decay period specifications. Earlier, we allocated these brief durations at the beginning and at the end of the signs and required them not to contain any movement or pose significant to the meaning of the sign. Then, blending can be applied between the decay of the preceding sign and the attack of the succeeding sign. Since the signs can be concatenated in any order, the attack and decay durations must be the same for every sign, decided as a convenient 1 second (Figure 3.4).

The blending procedure is linear in that it simply takes a weighted average of the poses of the two signs while arranging the weights to vary linearly between 0 and 1. The sum of the weights is 1 at all times during blending. If the signs are rewritten as $P=S_n(t)$, a sign sequence with m signs $S_1 \ldots S_m$ can be defined piecewise as:

$$T(t) = \begin{cases} S_1(t), & \text{if } 0 < t < E_1 \\ S_2(t - E_1), & \text{if } E_1 < t < E_2 \\ \ldots & \\ S_m(t - E_{m-1}), & \text{if } E_{m-1} < t < E_m \end{cases}$$

where E_n is the time the nth sign has completed playing, obtained by adding durations of the previous signs to that of the nth sign. When the blending process and the overlap duration are incorporated, E_n is redefined as:

$$E_n' = \sum_{i=1}^{n} D_i - a$$

where a is the length of the attack and decay periods and D_i is the duration of the ith sign. Then, blending the signs S_n and S_{n+1} can be formulated with the following reassignment.

$$T'(t)=\begin{cases} S_1(t), & \text{if } 0<t<E_1' \\ \dots & \\ S_n(t-E_{n-1}'), & \text{if } E_{n-1}'+a<t<E_n' \\ \dfrac{(E_n'+a-t)}{a}S_n(t-E_{n-1}')+\dfrac{(t-E_n')}{a}S_{n+1}(t-E_n'), & \text{if } E_n'<t<E_n'+a \\ S_{n+1}(t-E_n'), & \text{if } E_n'+a<t<E_{n+1}' \\ \dots & \\ S_{m-1}(t-E_{m-2}'), & \text{if } E_{m-2}'+a<t<E_{m-1}' \\ \dfrac{(E_{m-1}'+a-t)}{a}S_{m-1}(t-E_{m-2}')+\dfrac{(t-E_{m-1}')}{a}S_m(t-E_{m-1}'), & \text{if } E_{m-1}'<t<E_{m-1}'+a \\ S_m(t-E_{m-1}'), & \text{if } E_{m-1}'+a<t<E_m'+a \end{cases}$$

4.2. Articulated Body Animation and Rendering

4.2.1. Forward Kinematics

As with all skeleton driven, skinning based surface models, our model has its mesh-skeleton relationship built in. There are no submeshes to assign to joints, but rather each vertex in the mesh has a vector of normalized weights, indicating how much it must be affected by each joint in the skeleton. However, the MOCAP data contain only individual angles for each joint, so a preprocessing step is necessary to convert these angles into global transformation matrices for each joint before they can be applied to the vertices. This is done through forward kinematics. The process, as done in each frame of the animation, can be described as follows.

First, let the skeleton be defined as a vector $J=j_1, j_2 \dots j_n$ of joints and the function p(j) as a function that returns the parent of a given joint or is undefined. To be able to drive this skeleton, one also needs to parse the parameter vector.

At an instant t, the parameter vector returned by the sign function P=S(t) includes the parameters to be applied to each joint. Since the order the joints are defined in the parameter vector are known, it is possible to extract the parameters belonging to a particular joint from the vector. In particular, if $P=p_1,p_2\ldots p_m$, then the parameters of a joint j_i are p_{3i+1}, p_{3i+2} and p_{3i+3}. These parameters are the Euler angles the joint is to be rotated with. Note that the parameters p_1, p_2 and p_3 are not accounted for by these expressions. These parameters belong to the root joint, which has six DOF's, and specify its Cartesian coordinates.

From the parameters for each joint j_i, it is possible to calculate a local rotation matrix L_i as:

$$L_{i,z}=\begin{bmatrix}\cos\theta_z & -\sin\theta_z & 0\\ \sin\theta_z & \cos\theta_z & 0\\ 0 & 0 & 1\end{bmatrix}\quad L_{i,x}=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\theta_x & -\sin\theta_x\\ 0 & \sin\theta_x & \cos\theta_x\end{bmatrix}\quad L_{i,y}=\begin{bmatrix}\cos\theta_y & 0 & \sin\theta_y\\ 0 & 1 & 0\\ -\sin\theta_y & 0 & \cos\theta_y\end{bmatrix}$$

$$L_i = L_{i,z}L_{i,x}L_{i,y}$$

The MOCAP hardware uses the ZXY convention with the Euler angles (in degrees), both in specification in the parameter vector and in application of the simple rotations. Hence, of the above angles, θ_z is p_{3i+1}, θ_x is p_{3i+2}, and θ_y is p_{3i+3}.

Once the local rotation matrices L_i are known for each joint j_i, the global rotation matrices G_i can be recursively defined as follows.

$$G_i=\begin{cases}L_i G_{p(i)}, & \text{if p(i) is defined}\\ L_i, & \textit{otherwise}\end{cases}$$

Note that, no motion validity checking or DOF limiting is applied during the calculations. Such checks are not necessary, since the motions are captured from a real person.

Once the G_i matrices are calculated, the forward kinematics calculations are completed and deformations can be applied.

4.2.2. Deformation

As well as positions, the model also keeps a weight vector for each vertex. These weights indicate how much a particular vertex is affected by the rotations of each of the joints and add up to 1 for each vertex. The deformation procedure we employ is derived from the SSD algorithm in that to obtain a final vertex position; we calculate the weighted average of the positions directed by the contributions of each of the joints in order to produce the final vertex position.

Mathematically, if our skeleton has the joints $J=j_1, j_2 \ldots j_n$ and our model has a set of m vertices $V=\{v_1, v_2 \ldots v_m\}$, then the model must also have a vector $W_i=w_{i,1}, w_{i,2} \ldots w_{i,n}$ of weights for each vertex v_i. Note that the sum of the weights in each W_i is 1. If v_i is the vector of Cartesian coordinates for the ith vertex, then for each frame, a new position v_i' is calculated as:

$$v_i' = \sum_{j=1}^{n} w_{i,j} G_j v_i$$

where G_j is the global rotation matrix for the jth joint. For clarity, we have omitted the transformation of the vertex coordinates from the model space to skeletal space.

Once the new vertex positions v'_i are calculated for each vertex, the new coordinates are fed to the rendering pipeline to produce one frame of the animation. The operations of the rendering pipeline are beyond the scope of this document.

5. RESULTS

The approach and methodology of the system are explained above; yet, the quality and performance of the final product merits further discussion. It is from these results that conclusions for the work can be derived.

5.1. Realism

In terms of realism, the system can be said to perform moderately successful. While the produced animations look adequately smooth and continuous, and result in high quality renders, the sign animations themselves are not always fit for teaching purposes. To evaluate the realism of our results from this perspective, we check the animations the system produces against reference videos. Our conclusion is that, realism problems of the signs stem from the difficulties in capturing consistent data during MOCAP.

5.1.1. Motion Capture Issues

Collecting a representation of the actor's motions with MOCAP is quite a challenge in practice. Ultimately, as it turns out, most of the problems with the captured MOCAP data can be attributed to inaccurate calibration. Note that while preparation can be used to adapt the virtual model to the actor in terms of dimensions, manual calibration is still required to make sure the orientations are matched as well.

Calibration is a tricky process, especially for the arms. In our MOCAP system, the arms have to be manually calibrated with correction angles, which seem to be quite unstable, in each direction in a trial and error manner. Unfortunately, it is very difficult to obtain an accurately calibrated model with this approach, which is akin to manually using forward kinematics. A common result of this problem is hands that individually move and pose correctly, but fail to cooperate accurately. Naturally, this result is most noticeable in signs involving the two hands coming together or snapping the fingers. Figure 5.1, Figure 5.2, Figure 5.3 and Figure 5.4 exemplify these cases.

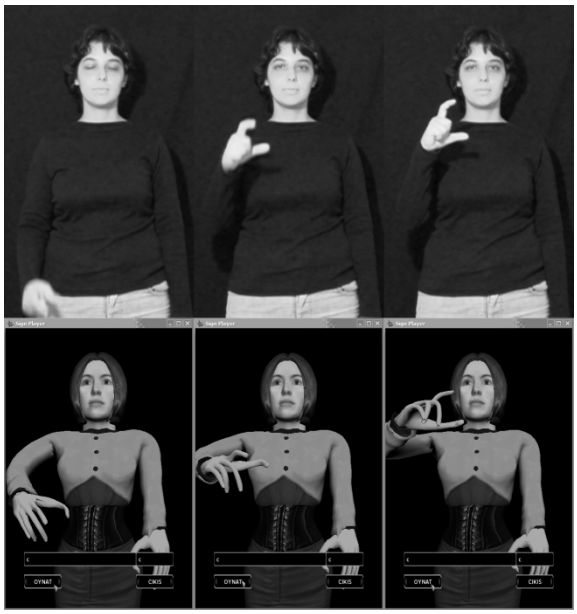

Figure 5.1. Sample reproduction of the letter “C”: Left to right above are sample frames from the reference video for the letter, below are from the reproduction. A simple letter, “C” is clearly reproduced accurately.

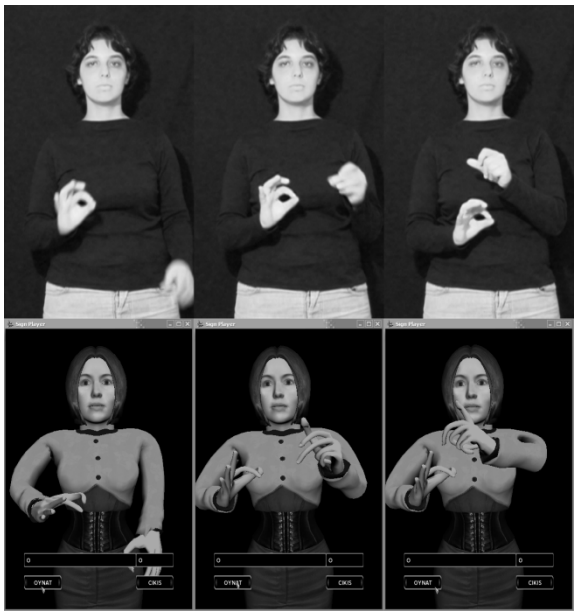

Figure 5.2. Sample reproduction of the letter “Ö”: Left to right above are sample frames from the reference video for the letter, below are from the reproduction. “Ö” requires finger snaps, which are correctly reproduced but the relative position and orientation of the hands is skewed.

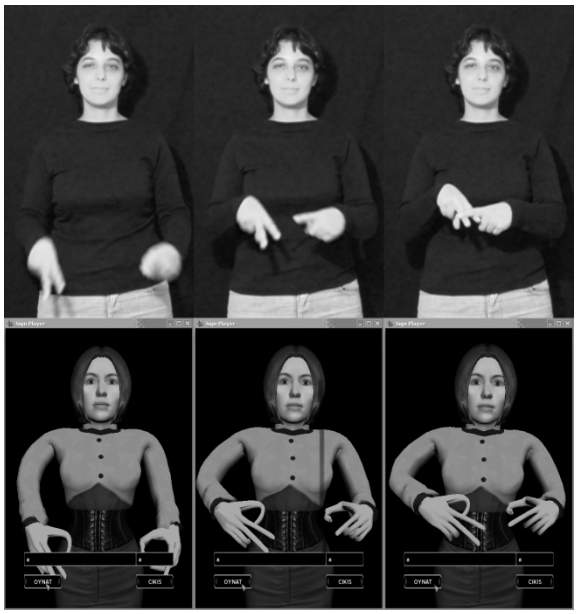

Figure 5.3. Sample reproduction of the letter “A”: Left to right above are sample frames from the reference video for the letter, below are from the reproduction. “A” requires precise association of the hands and the fingers, but while the hand shapes are correctly reproduced, the association is missing.

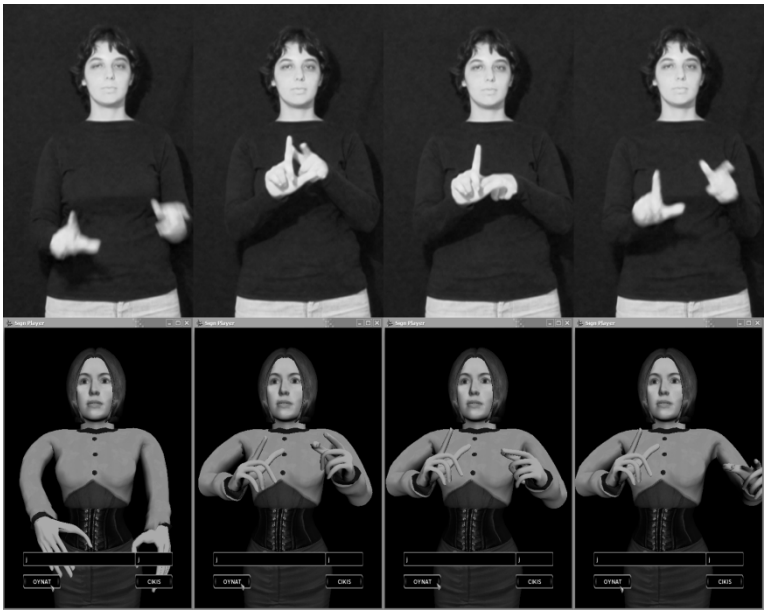

Figure 5.4. Sample reproduction of the letter “J”: Left to right above are sample frames from the reference video for the letter, below are from the reproduction. “J” requires association between the hands as well as movement of the left hand, but while the hand shapes and the movement are correctly reproduced, the association is missing.

5.1.2. Standard Repositioning

While a real actor is signing, it is required to produce a realtime preview of the observed motion for immediate feedback and correction as necessary. The MOCAP software produces such a 3D preview, but it is not the most practical kind of feedback for the actor. A real actor would be used to looking at a mirror for a feedback of his motions, but to the inconvenience of the actor, the 3D preview is not inverted like a mirror.

Moreover, the actor looking at the preview actually degrades the data quality because it affects the detected positions. In particular, the head and part of the body of the actor instinctively turns to look at the PC monitor where the preview is displayed. While using a helmet mounted display would help prevent this problem, it is also possible to extend the standard repositioning procedure to fix the rotations of these other bones to zero, instead.

5.1.3. Blending

Linear blending, implemented in this system, is the most straightforward way to smoothly concatenate two animations. However, it only preserves C^0 continuity and hence looks somewhat artificial in the output. In particular, the motions are transitioned smoothly but the transition rates change abruptly, causing the virtual actress to look like she has suddenly changed her mind. Note that inaccurate adherence to the attack and decay specifications also cause this problem. We assume that a better blending method that would preserve another order of continuity would definitely produce more natural transitions. A sample of the obtained blending transitions is given in Figure 5.5.

Figure 5.5. Sample blending sequence: Transition between the simple sign for the letter “L” and the two hand sign for the letter “J” demonstrates blending. Notice that the right hand stays in position instead of returning to the rest position.

5.1.4. SSD Artifacts

The realism of the outputs also suffers mildly from the lack of special handling of the SSD artifacts of the joints, which can cause the limbs to look unnatural at certain configurations. Fortunately, methods are available to remedy these problems [52] [54]. Figure 5.6 demonstrates one of the worst SSD artifacts of the system.

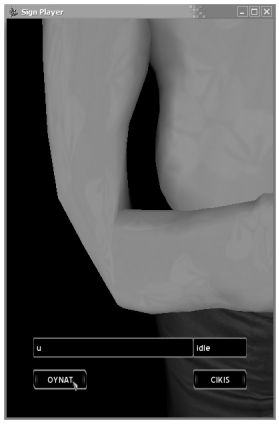

Figure 5.6. Sample SSD artifact: The right elbow loses part of its thickness while signing for the letter “U”.

5.2. Performance

From a performance point of view, the implemented system can be considered very successful. On the same workstation PC previously mentioned in MOCAP, with hardware

acceleration for 3D graphics using an NVIDIA 8400M GS graphics adapter, the playback application achieved frame rates between 550 fps to 700 fps in our tests (at 400x600 image resolution and 24 bit color depth, using OpenGL without frame rate limiting and no Vsync).

5.3. Communication

While the playback application is running, the sign animations are loaded on demand and kept in memory as long as there is memory available. Assuming the database is housed on another server, the bandwidth requirements for connecting the server can be calculated with the expression bandwidth = (size of a frame) x (frame rate). Size of a frame is the size of the parameter vector for a frame, which is 123 x 8 = 984 bytes, assuming the parameters are stored as 8 byte doubles. Hence, for the attained MOCAP frame rate, 77 fps, the bandwidth required is 984 x 77 = 75,768 bytes per second (Bps) or 606,144 bits per second (bps). This requirement reduces to 984 x 20 = 19,680 Bps or 157,440 bps for the minimum frame rate required for the moving picture illusion. Both numbers are practical with a modern digital subscriber line (DSL) connection.

Since actual images are transferred between the renderer and the final client, the bandwidth requirements are much higher. Uncompressed video at 400x600 image resolution and 24 bit color depth at 20 fps requires a 400 x 600 x 24 x 20 = 115,200,000 bps connection. While such rates are only possible in local area networks (LAN), it is possible to reduce these requirements by reducing the resolution and color depth or compressing the produced video.

5.4. Additional Discussion

Once system tests are done in the execution phase, a number of other results emerge. For instance, the need for facial expressions is stressed once again. As it is currently implemented, the virtual actress lacks any expression whatsoever. At this state, she looks little more than a humanoid robot. Some form of expression must be fed to the face of the actress if she is to look like a human at all.

The lack of facial expressions is also exaggerated by the availability of 3D navigation in the playback application. At the default pose, the virtual actress is conveniently looking forward, at the camera. If the camera is moved, however, the actress needs to react, just as a real human would, by turning to look at the camera, even if with her eyes only. The current implementation, however, fixes the eyes to look forward, regardless of the position of the camera, preventing the user from feeling communicated to.

The lack of head movement support in the system also limits the realism of the virtual actress. Our system has temporarily attached the head to the chest bone of the model. This provides a suitable placement of the head, but does not allow it any independent movement. On the contrary, a real person moves his or her head even when idle.

6. CONCLUSIONS

6.1. Highlights

Generally speaking, the implemented system can be considered a successful frontend to any sign producing application. As long as the application implements the concatenative sign synthesis model, this frontend can be exploited. Furthermore, since no assumptions are made about the particular sign language represented in the database, the system can virtually support any language.

The system provides not only synthesis and playback of signs, but also high quality realtime rendering of a virtual actress. The realism of the renders provided by the system is notable, as the quality is certainly above common implementations.

An even more important achievement of the system is its realtime performance. With realtime rendering, the playback module can be used in any interactive 3D application. Moreover, there is no limitation on using a different virtual actor to suit the needs of different future requirements.

Finally, the fact that the system was designed to support the broadly recognized BVH format is also an asset. While the current database contains a realistic set of signs, it can easily be extended with more, provided they are in BVH format. There is no restriction on the source of the sign data either. Besides MOCAP, the system would work equally well with authored animations, as long as they can be provided in BVH format. Note that, supporting skeletal animation data for input is also a strategically sound choice; as such data can be easily manipulated to be used in or adapted from other projects.

6.2. Future Directions

Aiming to be the frontend of a sign tutoring tool, this work is limited in its goal and feature set. Yet, there are many areas that can benefit from improvements.

6.2.1. Facial Expressions and Eye Movements

As discussed before, facial expressions are a definite requirement in a system such as ours. They are important for the perception of reality in the virtual actor and an integral part of many sign languages, so they must be supported if the system is to remain generic. Eye control is another important feature that is missing. Real social interactions involve the two parties actively look at each other, but our actress only looks ahead.

Our implementation does not reproduce facial expressions, because there is no data source for generating them. In particular, a video processor is required to capture the facial expressions along with the MOCAP data so that the data can be fed to a facial expression synthesizer. However, our work does incorporate the Xface engine so that it can properly support such an extension as soon as a data source is available. Xface also supports full control of eye movements. A future extension of our work can easily use the Xface engine to actively direct the actress' eyes to look at the camera. Moreover, since our model is directed by a well defined skeleton structure, she can also be programmed to turn partially to the camera with her body.

6.2.2. Head Movements

It was already stated that head movements are necessary if the actress is idle. In addition, many sign languages actively use head movements, just like facial expressions. Hence, our system must be extended to support them in the future.

Currently, the head is attached properly to the body at the chest joint but never moves. However, a recent improvement in our MOCAP hardware added proper head MOCAP support. The motions of the head are recorded on dedicated joints in the skeleton and can be reproduced on the virtual actress. To support the head motions with our implementation, the only necessary change is to attach the head to the appropriate neck bone instead.

When the head movements are independently supported and there are two separate engines to support the face and the body, as in our system, there is often a unification problem. Specifically, the meshes for the head and the body are separate and do not reproduce the elastic skin between them when moved in different directions. This problem does not arise in our system, because we do have head motion data to drive the head independently. If the system is extended to support such data, the easiest solution would be to manually create polygons between the border vertices of the head and the body meshes. As the vertices are deformed, the polygons will stretch and create an elastic skin effect.

6.2.3. Retargeting

An important extension would be supporting retargeting, manually or automatically, of the motion data to virtual models with different skeleton topologies. Our implementation currently requires that the virtual model have the same skeleton as the MOCAP model. We also implement basic support for retargeting on the software side while building the database, but do not do retargeting itself. The benefits of retargeting are twofold. First it would allow other, significantly different, models be used in the visualization, without modifying the database of MOCAP data. Second, it would allow the database to be extended with MOCAP data taken from other actors or other MOCAP systems, which may produce different skeleton topologies.

6.2.4. Additional Improvements

The MOCAP data of our system are manually entered into the database and the sign sequence is acquired from the user interface controls. In return, the output visualization is produced in the application’s own window. It is possible to move all of these data streams into generic inputs and output. For instance, the database can be directly constructed from the command line with the names of separate BVH files, the sign sequence also expected from the command line and the output can be directed to a generic operating system window. If all of these interfaces are made generic, the system can be embedded in any other system without modification. The ouput can be even more generalized by producing

VRML output (since 3D models are already available) instead of a rendering for better network performance, should the application be distributed.

Our system is tuned for our specific models, skeletons and meshes. However, with other digital data, substantial rearrangement may need to be done. While our code is designed to be easily adaptable to other models, it does not provide an external interface to do these settings without rebuilding the binaries. The addition of a few GUI screens of command line parameters could make the system a lot more useful.

Finally, an obvious improvement of any system involving a database is the extension of the database with more data. Our system is no exception. Since there is no restriction on the signs the system can store, synthesize and play, it is only natural to expect that the database will be extended with more signs in the future.

APPENDIX A: SYSTEM OPERATION GUIDE

This document is a guide to the operation of our Sign Language Animation Synthesis system. The system is comprised of software and data files that are used to generate realtime, 3D sign animations of a virtual actress. The animations are obtained by combining sign definitions that are fetched from a sign database in order specified by a sign list. The sign list is built from end user's input, whereas the database contains MOCAP data for each sign.

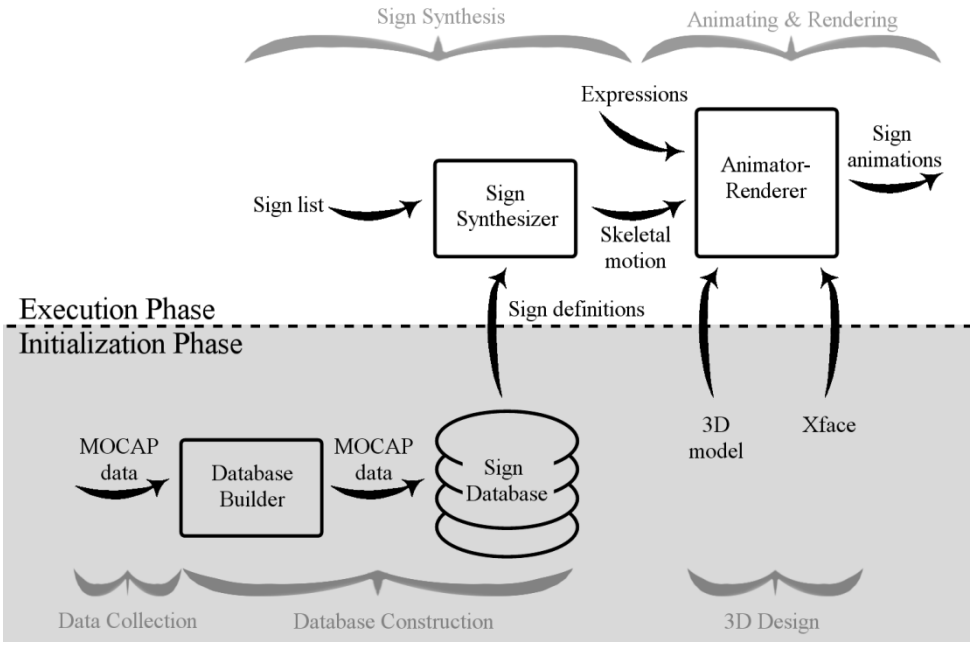

Figure A.1. System overview.

The operation of the system (Figure) can be explained in two phases, each of which logically applies to one type of user. The initialization phase is normally carried out by system operators to prepare the application data files that are used during the execution phase. The data files include the virtual actress model and the sign database. The execution phase spans the query-response loop of the system, which creates and plays animations for the end user. The rest of this document describes the operation of the system from the point of view of these users.

A.1. Initialization: Operator's Manual

A.1.1. Motion Capture

To be able to construct sign animation sequences, first, a database of sign definitions is required. The definitions are obtained through MOCAP, the first task in the initialization phase. While any MOCAP system can be used to capture the sign definitions, the system currently uses a Shapewrap III system by Measurand Inc.

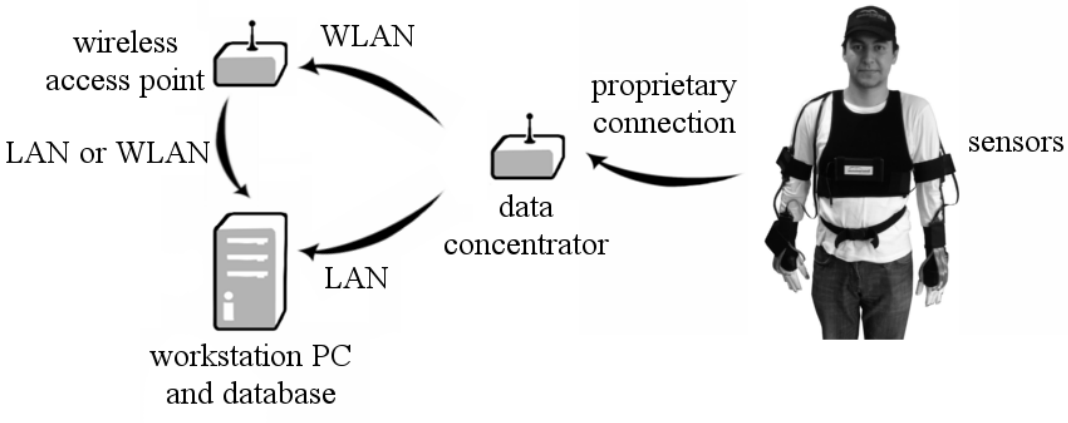

Figure A.2. ShapeWrap III MOCAP system.

Shapewrap III is a magnetic MOCAP system that can track most of the limbs of an actor, including the legs and the fingers. The system includes a software application, called ShapeRecorder for Microsoft Windows to record the captured motion data. ShapeRecorder does not connect to the sensors directly. Instead, it connects to a data concentrator box that is mounted on the actor along with the sensors. The communication between ShapeRecorder and the data concentrator can be wired (Ethernet based local area network, LAN) or wireless (IEEE 802.11 wireless local area network, WLAN with SSID "ShapeRecorder"). In contrast, the data concentrator is connected to the sensors through sensor hubs with cables, over proprietary protocols (Figure).

A.1.1.1. Mounting the Sensors. To capture motion data with Shapewrap III, the first step is to mount the sensors and the data concentrator to the actor using the provided Velcro

straps, belt, hat and vest, according to the Shapewrap III manual. There are three orientation sensors, connected to a single sensor hub. One is attached to the back of the head, one to the back of the waist and the other, which is embedded inside the hub, on the back of the actor. Arm sensors are thick black strips and are attached to the outside of either forearm in parallel, minding orientation (the right way up). Hubs of the arm sensors are attached to the upper arms. The hand sensor hubs are mounted on the outside of the wrists and the sensors themselves, thin blue strips, are inserted into the slits in the gloves.

Cabling should also be completed before the system is turned on. In particular, there should be a data cable from each sensor hub to the data concentrator. The cables and the available data concentrator slots are equivalent and can be used interchangeably. However, note that sensor to hub connections are somewhat flimsy and can get disconnected. It is recommended to check them regularly, although the data concentrator also signals disconnections with beeps.

A.1.1.2. Connecting. ShapeRecorder can connect to the data concentrator over LAN or WLAN. If wireless connection is preferred, the wireless access point must be turned on prior to powering the data concentrator. For wired connections, the wireless access point must be turned off.

When powered on (through a battery or AC adapter), the data concentrator first powers the sensor hubs, causing their red lights to turn on and then looks for the wireless access point to connect to. If the access point is active, wireless connection is made to the access point, after which the concentrator will assume the IP number 10.0.0.251 and emit four beeps to signal connection complete. At this point, the workstation PC should also be connected to the access point, through wireless or wired methods. If the data concentrator cannot find an access point, it attempts wired connection over the Ethernet port and takes the IP 10.0.0.250 instead, emitting two series of four beeps in the process. If the data concentrator can be pinged from the workstation PC, the connection is successful.

A.1.1.3. Starting ShapeRecorder. ShapeRecorder also supports serial connections, hence if the PC has serial ports (e.g. virtual ports for Bluetooth), they must be disabled before

launching the software. Once launched, ShapeRecorder looks for available data ports and should eventually discover the data concentrator on the network, if connection is successfully made.

Next, the detected sensors are shown and the ones to use in the MOCAP session are asked. Normally, all the sensors are used. Finally, ShapeRecorder requests a subject file from the user.

A.1.1.4. Preparing a Subject File and Calibration. A subject file is used to match the virtual skeleton to the MOCAP actor. To prepare a subject file, the operator must follow the instructions in the model link wizard of ShapeRecorder.

Before capturing motion data, the model has to be well calibrated. It is recommended that the calibration procedure is carried out before recording each movement. To calibrate ShapeRecorder, the software is instructed to reset its internal offsets in homing pose by clicking the homing button in the tape control window, while the actor puts out his hands in front, assuming the home pose. Once this process is done, the 3D preview in the software should reflect the actual pose of the actor. If it does not, or if there is drift in the preview, homing should be repeated.

A.1.1.5. Recording Data. If preparation and calibration is correct, recording can be initiated by the record raw data file option in ShapeRecorder. This brings up the recording control window, already recording. When done, the stop button should be clicked and data should be exported into a BVH file.

When recording signs for use in this system, care should be taken to have the actor in the same pose in the beginning and at the end of every sign. This is required to improve blending performance during playback. In addition, the first and the last 1 second part of the movement, called the attack and the decay, should only involve moving out of or into this pose. The attack and decay may be overwritten during blending and hence should not contain expressive poses.

A.1.2. Database Construction

Once the BVH files for the signs are available, the database can be constructed. The system keeps a sign database integrated into the skeleton file for the virtual actress for performance and integrity reasons. Hence, to construct the database, the virtual actress model must be built.

A.1.2.1. System Requirements. SignDBEditor is a simple application that is used to construct and edit databases for use in this system. SignDBEditor runs on Microsoft Windows XP or newer operating system, and requires a 2 Ghz CPU, 1 GB of RAM and 60 MB free disk space (shared with SignPlayer).

A.1.2.2. Using SignDBEditor. When first launched, SignDBEditor asks the user for a compatible model or skeleton file to initialize the user interface with. If there are signs specified in the skeleton file, they are loaded into memory and displayed to the user in an animation list window for editing or removal.

An animation can be removed from the memory by clicking on the dedicated remove button. A dedicated add button is also present, that, when clicked, causes the user to be prompted for a BVH file to load. Then, the animation extracted from the BVH file is loaded into memory and added to the animation list. The animations in the animation list are named automatically with their filenames when first loaded. This default name can be overridden with the F2 key.

When the list of animations is adequate for exporting into a database, the database file can be written by clicking on the OK button. At this step, SignDBEditor also allows the data to be written to another database file, still based on the skeleton in memory, if the user so desires.

A.1.3. 3D Design

The two digital assets of the system are the virtual actress model and the Xface head model. The virtual actress can be modeled in any standard modeling software and exported, as long as it uses the exact same skeleton as the MOCAP data. One of the BVH files can be imported by the software to obtain the skeleton, to which the mesh can be bound. The bone naming convention in original BVH files must be maintained in the model (except for the end affecters, which can be named with their parent, suffixed by a "_End"). In particular, a bone named "Chest" must exist so that it can be used to attach the Xface model to.

To model an Xface head, FaceGen software must be used. For more detail on this process, Xface documentation should be consulted.

A.2. Execution: End User's Manual

A.2.1. System Requirements

The sign animations are generated and played with a dedicated application, called SignPlayer. SignPlayer runs on Microsoft Windows XP or newer operating system, and requires a 2 Ghz CPU, 1 GB of RAM and 60 MB free disk space (shared with SignDBEditor). While it will detect and, if available, use it through OpenGL, SignPlayer does not require hardware acceleration for graphics.

A.2.2. User Interface and Tasks

SignPlayer features a hybrid user interface that works in both 2D and 3D style. The 2D interface is comprised of a number of controls, namely an edit box to type a list of signs to play, a read-only text box indicating the sign that is currently played by the actress and two buttons; a play button to initiate playback and another to quit the application. The user can move the mouse to control a pointer around the screen and use the left mouse

button to activate the controls that are interactive. For instance, the user can end the application by clicking on the quit button.

Behind the controls, SignPlayer draws the virtual actress in its current mode in 3D (Figure A.). The 3D display can be manipulated after being activated via the right mouse button. When active the mouse is used to control the position and orientation of the 3D camera. Namely, horizontal movements rotate the camera around the Y axis and vertical movements around the X axis. Rotation around the Z axis is achieved by keeping the middle button down while moving the mouse horizontally. Finally, the roller can be used to move the camera forward or backward in its line of sight.

Figure A.3. SignPlayer user interface.

A.2.3. Playing Sign Animations

SignPlayer has two modes of execution. Initially, it is in idle mode and displays the virtual actress looping an idle animation. The application stays in the idle mode indefinitely, until the user instructs it to switch to playback mode.

To request sign playback, first, the list of signs must be entered into the edit box. The list of signs is a simple string, where each character, including spaces, represents a different sign. In ordering the signs, the list assumes the left to right order the characters are used in the string. The characters should be lowercase and in English whenever possible. For specific Turkish characters, the related capital letter is used instead. For instance, to request the sign for “ğ”, one needs to type “G”. In case a sign is not recognized, it is simply ignored.

Once the sign list is specified, playback can be initiated by clicking on the play button. In response, SignPlayer first completes the current idle loop of the virtual actress and then directs her to play the sign animations specified in the sign list, in order. The signs are continuously played one after the other and cannot be interrupted or cancelled. As the actress plays the signs, the name of the sign currently being played is displayed in the read-only text box. When playing the sign list is complete, the application returns to the idle mode, expecting additional playback requests while playing the idle loop.

APPENDIX B: BVH FILE FORMAT SPECIFICATION

The Biovision Hierarchy (BVH) file format was originally developed by Biovision to store and carry motion capture data. It can contain both specification of skeleton hierarchies and motion data to animate them.

B.1. Description of File Contents

A BVH file is actually an ASCII text file, commonly structured with whitespace. The keywords and punctuation can be separated by any combination of spaces, tabs and newlines. Throughout the file, numeric values are specified in decimal form, angles in degrees and time values are in seconds. The contents of a BVH file are organized into two sections, as described below.

B.1.1. Hierarchies

The first section describes the skeleton hierarchies driven by the motion data. The section begins with the keyword “HIERARCHY”. Next, the skeletons are described sequentially. A skeleton description is actually a hierarchical definition of joints, starting with the root joint. The description of the root joint starts with the keyword “ROOT”, followed by a name for the root bone. Internal joints are specified by the keyword “JOINT”, followed by a name, instead. The end affecters are donated with the “End Site” keyword and do not have names.

Following the name, the offsets of a joint are specified after a single curly brace “{“ with the keyword “OFFSET”. Following this keyword are the X, Y and Z offsets of the joint, relative to its parent, also describing its base pose. The next part is used to indicate which part of the motion data is used to direct this particular joint. First is the keyword “CHANNELS”, followed by an integer, the number of parameters in the parameter vector that correspond to this particular joint. Typically, the root joint has six parameters and the rest of the joints have only three. End affecters do not specify joints and hence, do not have

channels. Following the channel count are the channel labels, the number of which should match the number of parameters as specified. The labels can be one of the preset keywords “Xposition”, “Yposition”, “Zposition”, “Xrotation”, “Yrotation” or “Zrotation” and indicate how to use the particular parameter obtained from the parameter vector. Note that the order the channels are specified is also used when the transformation matrices of the associated transformations are multiplied. In other words, if “Zrotation” comes before “Xrotation”, the rotation matrices are multiplied as $L_z L_x$.

Unless the joint is an end affecter, next, the child joints described in order. Finally, the root, joint or end affecter specification is completed with a single curly brace “}”.

B.1.2. Motion

The second section starts with the “MOTION” keyword, followed by the “Frames: “ keyword. Next, the number of frames specified in the motion section is located, which is an integer. Following this values, the playback rate is specified with the “Frame Time:” keyword followed by the frame time. The rest of the file is a sequence of parameter vectors, the count of which is equal to the number of frames specification. Each parameter vector is a simple sequence of real numbers in the order the joints and their channels are specified.

B.2. Common Conventions

While whitespace can be used liberally for formatting a BVH file, there are a few conventions that have found widespread adoption. Namely, the “HIERARCHY” and “MOTION” keywords and curly braces are usually placed on a line by themselves and joint specifications, including the braces themselves, are usually indented with tabs to indicate their level in the hierarchy. These conventions are honored in the sample file that follows.

```
HIERARCHY
ROOT Hips
{
	OFFSET 0.00 0.00 0.00
	CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
	JOINT Chest
	{
		OFFSET 0.00 8.00 0.00
		CHANNELS 3 Zrotation Xrotation Yrotation
		JOINT LeftArm
		{
			OFFSET 5.00 2.00 1.00
			CHANNELS 3 Zrotation Xrotation Yrotation
			JOINT LeftHand
			{
				OFFSET 0.00 -8.00 0.00
				CHANNELS 3 Zrotation Xrotation Yrotation
				End Site
				{
					OFFSET 0.00 -6.00 0.00
				}
			}
		}
		JOINT RightArm
		{
			OFFSET -5.00 2.00 1.00
			CHANNELS 3 Zrotation Xrotation Yrotation
			JOINT RightHand
			{
				OFFSET 0.00 -8.00 0.00
				CHANNELS 3 Zrotation Xrotation Yrotation
				End Site
				{
					OFFSET 0.00 -6.00 0.00
				}
			}
		}
	}
}
MOTION
Frames: 2
Frame Time: 0.05
 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 0.00 0.00 0.00 0.00 5.00 0.00 -5.00 0.00 5.00 45.00 0.00 0.00 -90.00 0.00 0.00 0.00 -15.00 0.00 0.00 -135.00 0.00
```

Figure B.1. A sample BVH file: The file contains just a simplified upper torso model and a 100 millisecond, two frame animation.

B.3. Grammar

The BVH file format can be more properly specified using a grammar for the file structure (excluding integrity requirements) as follows.

```
bvhFile: hierarchySection motionSection
hierarchySection: hierarchyHeader hierarchyData
hierarchyHeader: “HIERARCHY”
hierarchyData: skeleton+
skeleton: rootJoint
rootJoint: “ROOT” name { offsetSection channelSection joint* }
offsetSection: “OFFSET” real real real
channelSection: “CHANNELS” integer real+
joint: internalJoint | endAffecter
internalJoint: “JOINT” name { offsetSection channelSection joint* }
endAffecter: “End Site” { offsetSection }
motionSection: motionHeader frameCountSpec frameTimeSpec motionData
motionHeader: “MOTION”
frameCountSpec: “Frames:” integer
frameTimeSpec: “Frame Time:” real
motionData: parameterVector*
parameterVector: parameter+
parameter: real
name: string
string: alpha+ alphanumeric*
alpha: {“A”-“Z”} | {“a”-“z”}
alphanumeric: alpha | digit | “_”
integer: sign digit+
real: integer | sign digit+ “.” digit+
digit: {“0”-“9”}
sign: “” | “+” | “-”
```

Figure B.2. BVH file format grammar: The top object is bvhFile. Note that unquoted spaces represent whitespace.

REFERENCES

1. Kochanek, Doris H. U. and Richard H. Bartels, "Interpolating Splines with Local Tension, Continuity, and Bias Control," *ACM SIGGRAPH Computer Graphics*, pp. 33-41, 1984.

2. Steketekee, Scott N. and Norman I. Badler, "Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control," *ACM SIGGRAPH Computer Graphics*, vol. 19, no. 3, pp. 255-262, 1985.

3. Baecker, Ronald M., "Picture Driven Animation," in *AFIPS Joint Computer Conferences*, Boston, Massachusetts, pp. 273-288, 1969.

4. Burtnyk, N. and Wein M., "Interactive Skeleton Techniques for Enhancing Motion Dynamics in Key Frame Animation," *Communications of the ACM*, vol. 19, no. 10, pp. 569-569, 1976.

5. Zeltzer, D., "Motor Control Techniques for Figure Animation," *IEEE Computer Graphics and Applications*, vol. 2, no. 9, pp. 53-59, 1982.

6. Watt, A. and M. Watt, *Advanced Animation and Rendering Techniques*, Peter Wegner, Ed., ACM Press, New York, New York, 1992.

7. Barr, Alan H., "Global and Local Deformations of Solid Primitives," in *International Conference on Computer Graphics and Interactive Techniques*, pp. 21-30, 1984.

8. Sederberg, Thomas W. and Scott R. Parry, "Free Form Deformation of Solid Geometric Models," in *International Conference on Computer Graphics and*

Interactive Techniques, pp. 151-160, 1986.

9. Korein, J. U. and Norman I. Badler, "Techniques for Generating the Goal-Directed Motion of Articulated Structures," *IEEE Computer Graphics and Applications*, vol. 2, no. 9, pp. 71-81, 1982.

10. Girard, Michael and A. A. Maciejewski, "Computational Modeling for the Computer animation of Legged Figures," in *International Conference on Computer Graphics and Interactive Techniques*, pp. 263-270, 1985.

11. Girard, Michael, "Interactive Design of 3-D Computer-Animated Legged Animal Motion," in *Symposium on Interactive 3D Graphics*, pp. 131-150, 1987.

12. Badler, Norman I., K. H. Manoochehri, and G. Walters, "Articulated Figure Positioning by Multiple Constraints," *IEEE Computer Graphics and Applications*, vol. 7, no. 6, pp. 28-38, 1987.

13. Zhao, J. and Norman I. Badler, "Real Time Inverse Kinematics with Joint Limits and Spatial Constraints," University of Pennsylvania, Technical Report MS-CIS-89-09, 1989.

14. Phillips, Cary B., J. Zhao, and Norman I. Badler, "Interactive Real Time Articulated Figure Manipulation Using Multiple Kinematic Constraints," in *Symposium on Interactive 3D Graphics*, pp. 245-250, 1990.

15. Zhao, J. and Norman I. Badler, "Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures," *ACM Transactions on Graphics*, vol. 13, no. 4, pp. 313-336, 1994.

16. Lee, J. and S. Y. Shin, "A Hierarchical Approach to Interactive Motion Editing for

Human-Like Figures," in *International Conference on Computer Graphics and Interactive Techniques*, pp. 39-48, 1999.

17. Tolani, Deepak, Ambarish Goswami, and Norman I. Badler, "Real Time Inverse Kinematics Techniques for Anthropomorphic Limbs," *Graphical Models and Image Processing*, vol. 62, no. 5, pp. 353-388, 2000.

18. Phillips, Cary B. and Norman I. Badler, "Interactive Behaviors for Bipedal Articulated Figures," *ACM SIGGRAPH Computer Graphics*, vol. 25, no. 4, pp. 359-362, 1991.

19. Hahn, J. K., "Realistic Animation of Rigid Bodies," *ACM SIGGRAPH Computer Graphics*, vol. 22, no. 4, pp. 299-308, 1988.

20. Barzel, Ronen and Alan H. Barr, "A Modeling System Based on Dynamic Constraints," in *International Conference on Computer Graphics and Interactive Techniques*, pp. 179-188, 1988.

21. Terzopoulos, Demetri, John Platt, Alan Barr, and Kurt Fleischer, "Elastically Deformable Models," in *International Conference on Computer Graphics and Interactive Techniques*, pp. 205-214, 1987.

22. Barr, Alan H. and John Platt, "Constraints Methods for Flexible Models," in *International Conference on Computer Graphics and Interactive Techniques*, pp. 279-288, 1988.

23. Armstrong, William W. and Mark W. Green, "The Dynamics of Articulated Rigid Bodies for Purposes of Animation," *The Visual Computer*, vol. 1, no. 4, pp. 231-240, 1985.

24. Wilhelms, J. and B. A. Barsky, "Using Dynamic Analysis to Animate Articulated

Bodies such as Humans and Robots," *Canadian Information Processing Society Graphics Interface*, pp. 97-104, 1985.

25. Wilhelms, J., "Using Dynamic Analysis for Realistic Animation of Articulated Bodies," *IEEE Computer Graphics and Applications*, vol. 7, no. 6, pp. 12-27, 1987.

26. Armstrong, William W., M. Green, and R. Lake, "Near-Real-Time Control of Human Figure Models," *IEEE Computer Graphics and Applications*, vol. 7, no. 6, pp. 52-61, 1987.

27. Cohen, Michael F., "Interactive Spacetime Control for Animation," *ACM SIGGRAPH Computer Graphics*, vol. 26, no. 2, pp. 293-302, 1992.

28. Isaacs, Paul M. and Michael F. Cohen, "Controlling Dynamic Simulation with Kinematic Constraints," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 215-224, 1987.

29. Witkin, Andrew and Michael Kass, "Spacetime Constraints," in *Computer Graphics and Interactive Techniques*, pp. 159-168, 1988.

30. Willmert, K. D., "Graphic Display of Human Motion," in *ACM Annual Conference/Annual Meeting*, pp. 715-719, 1978.

31. Potter, T. E. and K. D. Willmert, "Three Dimensional Human Display Model," in *International Conference on Computer Graphics and Interactive Techniques*, pp. 102-110, 1975.

32. Badler, Norman I., J. O'Rourke, and H. Toltzis, "A Spherical Representation of a Human Body for Visualizing Movement," in *IEEE*, pp. 1397-1403, 1979.

33. Catmull, Edwin, "A System for Computer Generated Movies," in *ACM Annual Conference/Annual Meeting*, pp. 422-431, 1972.

34. Badler, Norman I. and Stephen W. Smoliar, "Digital Representations of Human Movement," *ACM Computing Surveys*, vol. 11, no. 1, pp. 19-38, 1979.

35. Chadwick, J. E., D. R. Haumann, and R. E. Parent, "Layered Construction for Deformable Animated Characters," in *International Conference on Computer Graphics and Interactive Techniques*, pp. 243-252, 1989.

36. Chen, David T. and D. Zeltzer, "Pump It Up: Computer Animation of a Biomechanically Based Model of Muscle Using the Finite Element Method," in *International Conference on Computer Graphics and Interactive Techniques*, pp. 89-98, 1992.

37. Thalmann, Daniel, J. Shen, and E. Chauvineau, "Fast Human Body Deformations for Animation and VR Applications," in *Computer Graphics International*, p. 166, 1996.

38. Wilhelms, J. and A. Van Gelder, "Anatomically Based Modeling," in *International Conference on Computer Graphics and Interactive Techniques*, pp. 173-180, 1997.

39. Scheepers, F., R. E. Parent, W. E. Carlsson, and S. F. May, "Anatomy-Based Modeling of the Human Musculature," in *International Conference on Computer Graphics and Interactive Techniques*, pp. 163-172, 1997.

40. Nedel, Luciana Porcher and Daniel Thalmann, "Modeling and Deformation of the Human Body Using an Anatomically-Based Approach," in *Computer Animation*, pp. 34-40, 1998.

41. Cani-Gascuel, M.P. and M. Desbrun, "Animation of Deformable Models Using

Implicit Surfaces," *IEEE Transactions on Visualization and Computer Graphics*, vol. 3, no. 1, pp. 39-50, 1997.

42. Aubel, A. and D. Thalmann, "Realistic Deformation of Human Body Shapes," in *Computer Animation and Simulation*, pp. 125-135, 2000.

43. Magnenat-Thalmann, N., R. Laperriere, and D. Thalmann, "Joint-Dependent Local Deformations for Hand Animation and Object Grasping," in *Graphics Interface*, pp. 26-33, 1988.

44. Hodgins, J. K., W. L. Wooten, D. C. Brogan, and James F. O'Brien, "Animating Human Athletics," in *International Conference on Computer Graphics and Interactive Techniques*, pp. 71-78, 1995.

45. Singh, K. and E. Kokkevis, "Skinning Characters Using Surface Oriented Free Form Deformations," in *Graphics Interface*, pp. 35-42, 2000.

46. Lewis, J. P., M. Cordner, and N. Fong, "Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton Driven Deformation," in *International Conference on Computer Graphics and Interactive Techniques*, pp. 165-172, 2000.

47. Sloan, P. P., C. F. Rose, and Michael F. Cohen, "Shape by Example," in *Symposium on Interactive 3D Graphics*, pp. 135-143, 2001.

48. Kry, P. G., D. L. James, and D. K. Pai, "EigenSkin: Real Time Large Deformation Character Skinning in Hardware," in *Symposium on Computer Animation*, pp. 153-159, 2002.

49. Bloomenthal, Jules, "Medial Based Vertex Deformation," in *Symposium on Computer*

Animation, pp. 147-151, 2002.

50. Wang, X. C. and C. Phillips, "Multi Weight Enveloping: Least Squares Approximation Techniques for Skin Animation," in *Symposium on Computer Animation*, pp. 129-138, 2002.

51. Mohr, Alex and Michael Gleicher, "Building Efficient, Accurate Character Skins from Examples," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 562-568, 2003.

52. Kavan, L. and J. Zara, "Real Time Skin Deformation with Bones Blending," in *WSCG Short Papers*, pp. 69-74, 2003.

53. Shoemake, Ken, "Animating Rotation with Quaternion Curves," in *Computer Graphics and Interactive Techniques*, pp. 245-254, 1985.

54. Kavan, L. and J. Zara, "Spherical Blend Skinning: A Real Time Deformation of Articulated Models," in *Symposium on Interactive 3D Graphics*, pp. 9-16, 2005.

55. Kavan, L., S. Collins, J. Zara, and C. O'Sullivan, "Skinning with Dual Quaternions," in *Symposium on Interactive 3D Graphics*, pp. 39-46, 2007.

56. Menache, Alberto, *Understanding Motion Capture for Computer Animation and Video Games*, Morgan Kaufmann Publishers, San Francisco, CA, 2000.

57. Akita, Koichiro, "Image Sequence Analysis of Real World Human Motion," *Pattern Recognition*, vol. 17, no. 1, pp. 73-83, 1984.

58. Rohr, K., "Towards Model Based Recognition of Human Movements in Image Sequences," *CVGIP: Image Understanding*, vol. 59, no. 1, pp. 94-115, 1994.

59. D. M. Gavrila, L. S. Davis, "3D Model Based Tracking of Humans in Action: A Multi View Approach," in *Conference on Computer Vision and Pattern Recognition*, p. 73, 1996.

60. Wren, C. R., A. Azarbayejani, T. Darrell, and A. P. Pentland, "Pfinder: Real-Time Tracking of the Human Body," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 780-785, 1997.

61. Moeslund, Thomas B. and Eric Granum, "A Survey of Computer Vision-Based Human Motion Capture," *Computer Vision and Image Understanding*, vol. 81, no. 3, pp. 231-268, 2001.

62. Moeslund, Thomas B., Adrian Hilton, and Volker Krüger, "A Survey of Advances in Vision-Based Human Motion Capture and Analysis," *Computer Vision and Image Understanding*, vol. 104, no. 2, pp. 90-126, 2006.

63. Aggarwal, J. K. and Q. Cai, "Human Motion Analysis: A Review," *Computer Vision and Image Understanding*, vol. 73, no. 3, pp. 428-440, 1999.

64. Johansson, Gunnar, "Visual Perception of Biological Motion and a Model for its Analysis," *Gunn*, vol. 14, no. 2, pp. 201-211, 1973.

65. Webb, J. A. and J. K. Aggarwal, "Structure from Motion of Rigid and Jointed Objects," *Artificial Intelligence*, vol. 19, no. 1, pp. 107-130, 1982.

66. Ward, Andy, Alan Jones, and Andy Hopper, "A New Location Technique for the Active Office," *IEEE Personal Communications*, vol. 4, no. 5, pp. 42-47, 1997.

67. Raab, F. H., E. B. Blood, T. O. Steiner, and H. R. Jones, "Magnetic Position and Orientation Tracking System," *IEEE Transactions on Aerospace and Electronics*

Systems, vol. 15, no. 5, pp. 709-718, 1979.

68. Miller, N., O. C. Jenkins, M. Kallmann, and M. J. Mataric, "Motion Capture from Inertial Sensing for Untethered Humanoid Teleoperation," in *IEEE-RAS International Conference on Humanoid Robotics (Humanoids)*, Santa Monica, CA, 2004.

69. Vlasic, D. et al., "Practical Motion Capture in Everyday Surroundings," *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3, 2007.

70. Sturman, David J., "Character Motion Systems," in *ACM SIGGRAPH*, 1994.

71. Loomis, Jeffrey, Howard Poizner, Ursula Bellugi, Alynn Blakemore, and John Hollerbach, "Computer Graphic Modeling of American Sign Language," in *International Conference on Computer Graphics and Interactive Techniques*, Detroit, Michigan, pp. 105-114, 1983.

72. Lu, Shan, Seiji Igi, Hideaki Matsuo, and Yuji Nagashima, "Towards a Dialogue System Based on Recognition and Synthesis of Japanese Sign Language," in *Gesture and Sign Language in Human-Computer Interaction.*, Springer Berlin / Heidelberg, 1998, pp. 259-271.

73. Zhao, Liwei et al., "A Machine Translation System from English to American Sign Language," in *In Association for Machine Translation in the Americas.*, Springer-Verlag, 2000, pp. 54-67.

74. Sims, Ed, "Virtual Communicator Characters," *ACM SIGGRAPH Computer Graphics*, vol. 34, no. 2, p. 44, 2000.

75. Prillwitz, S., R. Leven, H. Zienert, T. Hanke, and J. Henning, *HamNoSys. Version 2.0 - Hamburg Notation System for Sign Languages. An Introductory Guide*, Signum Press,

Hamburg, 1989.

76. Bangham, J. A. et al., "Signing for the Deaf using Virtual Humans," *IEE Digest*, 2000.

77. Bangham, J. A. et al., "Virtual Signing: Capture, Animation, Storage and Transmission - An Overview of the ViSiCAST Project," *IEE Digest*, 2000.

78. Kennaway, Richard, "Synthetic Animation of Deaf Signing Gestures," *Lecture Notes In Computer Science, Vol. 2298*, pp. 146-157, 2001.

79. Zwitserlood, Inge, Margriet Verlinden, Johan Ros, and Sanny van der Schoot, "Synthetic Signing for the Deaf: eSign," in *Conference and Workshop on Assistive Technologies for Vision and Hearing Impairment*, Granada, Spain, 2004.

80. Solina, Franc, Slavko Krapez, and Alas Jaklic, "Multimedia Dictionary and Synthesis of Sign Language," in *Design and Management of Multimedia Information Systems*, Mahbubur Rahman Syed, Ed., Idea Group Publishing, 2001, pp. 268-281.

81. Grieve-Smith, Angus B., "Sign Language Synthesis Application Using Web3D and Perl," in *Lecture Notes in Computer Science Vol.2298*, Ipke Wachsmuth and Timo Sowa, Eds., Springer, 2001, pp. 134-145.

82. Havasi, Laszlo and Helga M. Szabo, "HANDY: Sign Language Synthesis from Sublexical Elements Based on an XML Data Representation," in *Lecture Notes in Computer Science Vol. 3206.*, Springer Berlin / Heidelberg, 2004, pp. 73-80.

83. Havasi, Laszlo and Helga M. Szabo, "A Motion Capture System for Sign Language Synthesis: Overview and Related Issues," in *Lecture Notes in Computer Science Vol. 3804.*, Springer Berlin / Heidelberg, 2005, pp. 636-641.

84. Krnoul, Z., J. Kanis, M. Zelezny, and L. Müller, "Czech Text-to-Sign Speech Synthesizer," in *Lecture Notes in Computer Science, Vol. 4892.*, Springer Berlin / Heidelberg, 2008, pp. 180-191.

85. Aran, O. et al., "SignTutor: An Interactive System for Sign Language Tutoring," *IEEE MultiMedia*, vol. 16, no. 1, pp. 81-93, 2009.

86. Balci, Koray, "Xface: MPEG-4 Based Open Source Toolkit for 3D Facial Animation," in *Working Conference on Advanced Visual Interfaces*, 2004.

87. Balci, Koray, "Xface: Open Source Toolkit for Creating 3D Faces of an Embodied Conversational Agent," *Lecture Notes in Computer Science Vol. 3638*, pp. 263-266, 2005.

88. *Xface Web Site*, http://xface.itc.it/

89. *Shapewrap Web Site*, http://www.motion-capture-system.com/shapewrap.html

90. Parent, Rick, *Computer Animation Algorithms & Techniques*, 2nd ed., Morgan Kaufmann Publishers, Burlington, MA, USA, 2008.