sEditor: a Prototype for a Sign Language Interfacing System

Beifang Yi, Xusheng Wang, Frederick C. Harris, Jr, and Sergiu M. Dascalu

***Abstract*—Sign languages are as capable of expressing human thoughts and emotions as traditional (spoken) languages. The distinctive visual and spatial nature of sign languages makes it difficult to develop an interfacing system as a communication medium platform for sign language users. This paper targets this problem by presenting some explorations in the areas of computer graphics, interface design, and human-computer interaction with emphasis on software development and implementation. We propose a sign language interfacing system, as working platform, that can be used to create virtual human body parts, simulate virtual gestures, and construct, manage and edit sign language linguistic parts. It is expected that the system and the results presented in the paper would provide an example for the future sign language "editor."**

***Index Terms*—Graphical user interfaces, human gesture simulation, interactive systems, sign language.**

I. Introduction

SIGN languages have been proven linguistically to be natural languages [1], [2], just as capable of expressing human thoughts and feelings as traditional languages are. The visual and spatial nature of sign languages contributes to the lack of "editors" in such languages. The current writing systems, while making full use of various suggestive 2D icons or phonetic symbols, are indirect, unnatural transcriptions and transformations of the 3D expressions inherent in sign languages. This symbol representation for a sign language is, in fact, like a text encoding of spatial contents.

To address these problems, we draw from computer graphics and human-computer interaction (HCI), specifically human body modeling, user interface design, and software implementation, to develop a framework of a sign language interfacing system which we call sEditor. Based on an expanded version of [3], we present this system.

Considering biomechanics, a virtual human body is first constructed as a set of functional body components. Using the virtual body and focusing on the creation of natural hand configurations and the application of body joint motion constraints, virtual gestures are created by controlling the movements of the functional components. From the virtual gestures, sign language linguistic parts can be constructed by using the Movement-Hold Model. A graphical user interface supports the operations of gesture generation and editing, gesture database management, and creation, editing, storage, and retrieval of sign language linguistic parts.

Under the Fedora Core of Red Hat Linux operating system, sEditor is implemented in C++ and OpenGL and uses the Coin3D graphics library (an Open Inventor clone). The GUI interface is implemented using the Qt API and C++. It also can run in the VMware Virtual environment under Windows 7.

sEditor is designed to support constructing, managing, and editing virtual gestures. From these signing components, then sign language linguistic components, then phonemes and finally sentences can be created. To support other linguistic components, components may be stored and retrieved from sign language databases.

The organization of this paper is as follows: Section 2 presents related literature. Section 3 overviews the interfacing system. Section 4 discusses modeling and simulation of the human body. Section 5 presents the design, creation, and management of virtual gestures. Section 6 discusses the creation of sign language linguistic parts. Section 7 presents a discussion. See http://cs.salemstate.edu/~byi/sEditorDemos/ for additional related materials.

II. Related literature

An early interactive system analyzed and modeled the complex hand and arm movements of sign language [4]. Through the reconstruction and manipulation of actual sign movements, this system was designed to convey American Sign Language (ASL) essential grammatical information using line drawing.

The Dictionary of ASL on Linguistic Principles (DASL [5]), now the Multimedia Dictionary of American Sign Language (MM-DASL [6]), presents ASL signs in full-motion (video of ASL entries), enabling users to search for words by entering English words or ASL pronunciation criteria.

Live-action video clips with graphical user interfaces

Manuscript received August, 10, 2012. This work was supported in part by the National Science Foundation under Grant number IIA-1301726.

Beifang Yi is with the Department of Computer Science, Salem State University, Salem, MA 01970 USA (email: byi@salemstate.edu).

Xusheng Wang is with the Department of Mathematics, Computer Science and Cooperative Engineering, University of St. Thomas, Houston, TX 77006 (email: xwang@stthom.edu).

Frederick C. Harris, Jr is with the Department of Computer Science and Engineering, University of Nevada, Reno, Reno, Nevada 89557 USA (email: fredh@cse.unr.edu).

Sergiu M. Dascalu is with the Department of Computer Science and Engineering, University of Nevada, Reno, Reno, Nevada 89557 USA (email: dascalus@cse.unr.edu).

support sign language studies. For example, SignStream is a multimedia database tool designed to facilitate ASL linguistic and computer vision research on visual-gestural language [7], [8]. Data from native signers are collected with video collection equipment and users can enter annotation information into data distinct fields [9], [10]. The video clips and associated linguistic annotations are available in multiple formats for ASL studies and gesture analysis. One example concerns the design of an online web browser for the deaf community [11]. It provides hyperlinks within video in a sign language-based, text optional web environment.

Today, lifelike virtual human figures can be constructed [12], [13]. Human avatars can imitate human actions and even facial expressions. All the body joints and featured parts (such as eyebrows or mouth), represented as various parameters, are controlled in their motions, allowing the creation of virtual gestures.

The use of virtual human figures in sign language studies is a popular approach [14]-[18]. However [19] provides framework concepts and [20] provides sign language gesture issues (with respect to modeling transitions between signs, modeling inflectional processes, and related concerns) to inform virtual signing systems.

Human avatars (i.e., virtual human bodies) may provide advantages over videos of native signers ([9], [10], [14], [15], [16], [18]). However current systems are limited because the linguistic parts (the sign language phonemes, words, and sentences) are fixed and the lexicons are limited. Therefore, the users cannot create new sign language "words" or "phrases."

To target this issue, sEditor is an "open" platform for different sign languages with user interfaces for the creation and management of sign language linguistic parts (from phonemes to sentences). To produce more natural hand configurations, the handshapes (the most important sign language parameters) generated by sEditor incorporate hand biomechanical constraints.

sEditor serves as sign language "word" editor prototype with which sign language users can "write" in their languages like a regular text editor for spoken languages [21].

III. THE ARCHITECTURE OF THE PROTOTYPE

The sign language interfacing system provides an interactive virtual environment in which users can construct virtual gestures through a virtual human body, associate the gesture sessions with sign language linguistic parts (such as morphemes, words, and phrases), create virtual signs with the use of databases for managing the linguistic parts, and even "write" in a sign language. The architecture of the system is shown in Figure 1.

The system consists of five main functional components:

1) *A virtual human body*: This central and foundational part of the interfacing system is modeled based on the anatomical structure of the human body. The whole body is divided into various functional parts, each of which is represented by a set of parameters (discussed in detail later).
2) *A virtual environment*: Several virtual cameras are "installed" in this virtual box, and the background can be set to different colors. Because the hand plays the most important role in signing, two virtual settings are created for both hands of the virtual body.

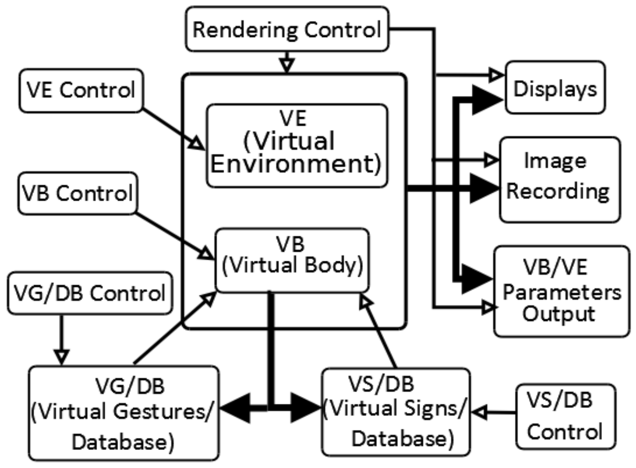


Figure 1: The sEditor architecture (VE: Virtual Environment; VB: Virtual Body; VG: Virtual Gestures; VS: Virtual Signs; DB: Database).

3) *Inputs and Control*: This provides the inputs either from the databases or from users to the virtual body for producing virtual gesture sessions, to set up a particular virtual environment (such as background and camera setups), and to control the output processes and formats. This part includes the following components:
   - VB Control: to provide rotations at the body joints and define facial expressions.
   - VE Control: to set up the virtual environment and fine-tune virtual cameras; to select and place a character as the virtual sign from a character pool (male and female virtual characters in diverse races).
   - Rendering Control: to render the virtual body in a chosen style, to display and record the outputs (images and parameter values for the virtual body and environment) on the screen or storage in a certain format.
   - VG/DB Control: to construct and edit virtual gestures and store/retrieve them to/from the virtual gesture database.
   - VS/DB Control: to create and edit virtual signs and associate them with a particular sign language notation; to store and retrieve from the virtual sign database the virtual signs and their corresponding related notations.
4) *Outputs*: The output of the system is represented in different formats: visual signs displayed on the screen,

their image session recorded and saved to storage, and parametric representations of the virtual gestures of the virtual body and those of the virtual environment.

5) *Virtual gesture/sign database*: All the created gestures, virtual signs, and their sign notations are stored in their databases. Gestures and signs are sets of parametric values defined in particularly designed data structures. Sign notions are English translations of the signs. Two databases are included in the system:

- *VG/DB Database*: a database for virtual gestures. A gesture is a list of virtual postures with a timing factor for each of the postures. Thus a gesture, after being loaded from the database, is displayed as an animation session on the screen. This gesture database includes sub-databases for body postures and hand configurations.
- *VS/DB Database*: a database for virtual signs. A sign consists of one or more virtual gestures and is a sign linguistic part in a particular sign language. A sign is stored in the database together with its sign notation and the links to its related signs.

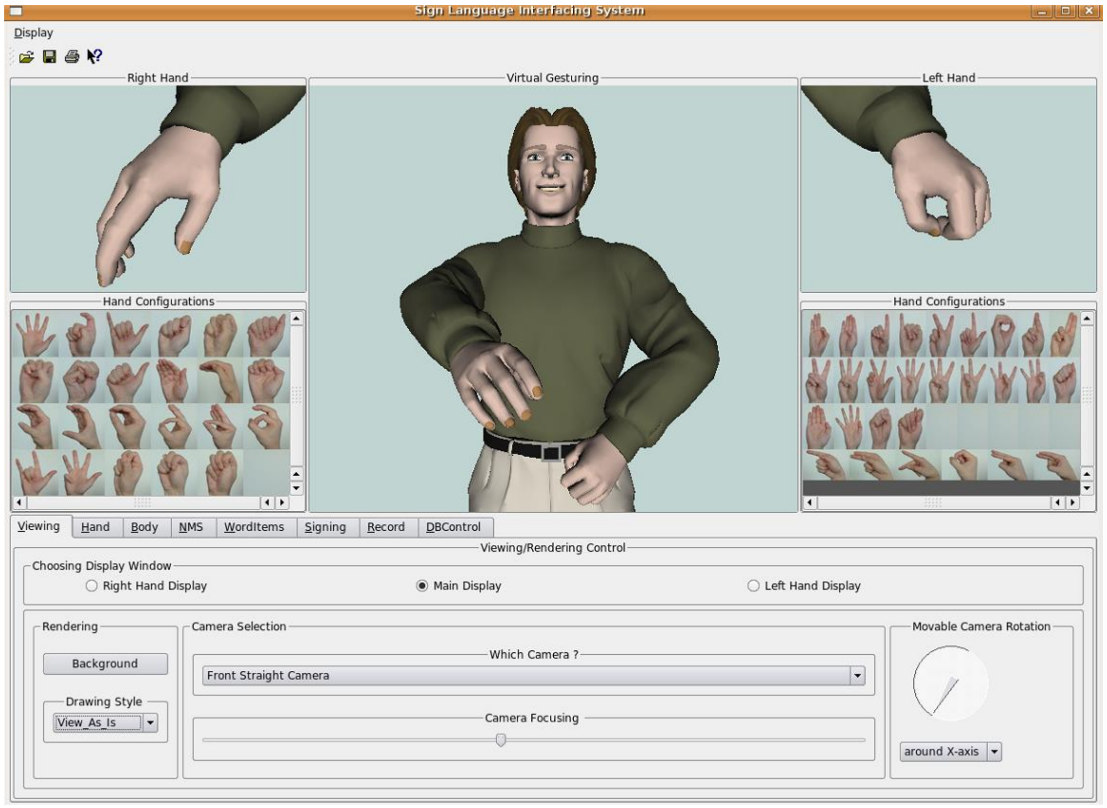
One screen shot of the sign language interfacing system is shown in Figure 2. The upper part of the system layout is for display with a main display window in the middle for displaying body gestures/signs and two accessory (smaller) ones on either side for demonstrating hand gestures. Below the hand display windows are hand icons of the most frequently used hand configurations in sign languages. Clicking on a hand icon will load from the hand database the corresponding hand configuration to the current hand (right or left). The lower part is arranged as a set of graphical tabs for controlling virtual body and environment, creating and editing gestures and signs, managing their databases, and rendering and recording outputs.

IV. Constructing a Virtual Human Body

A virtual human body is a module that can be sub-divided into sub-modules (i.e., body parts) according to the hierarchical structure of the human body as it moves and how body part motion is coordinated. In this section, we introduce the structure of the body modules and then describe the simulation of the motions of the human body parts.

A. Modeling the Human Body

The first step in human body modeling is the classification of body parts according to their contributions to gestures and signing. The lower parts of the body (legs, feet, and hips) are rarely used during signing, and thus they are abstracted using a single body part. The Torso represents the body trunk, which is connected to the shoulders and head. The body part *Head* contains the eyes, hair, neck, and the frontal part, which is used to model the facial expressions or NMS (nonmanual signals) in sign languages. The part *Shoulder* consists of the upper arm and the clavicle. The part *Hand*, composed of the palm, fingers and thumb, is critical in the simulation of signing and is modeled differently than the other body parts. The part *Forearm* connects the shoulder (upper arm) and the hand.



Choosing Display Window: ○ Right Hand Display [✓ Main Display] ○ Left Hand Display

Drawing Style: View_As_Is

Which Camera ?: Front Straight Camera

Movable Camera Rotation: around X-axis

Figure 2: One screen shot of sEditor: an example of creation and edition of hand configurations on left and rights.

We use a tree structure (see Figure 3) to design algorithms for the movements of the body parts that are connected and thus the movements of a parent node in the tree will propagate to all of its child nodes (children); the ultimate movements of a child are the cumulative combination of the movements of all its parents in the tree. Each part's motions are modeled in its own (local) coordinate system; the movements upon the body parts to which that part is connected will be recorded in a *motion engine* (see the following subsection), which in turn will *drive* the connected body parts for corresponding reactions automatically in the system.

For example, *rForearm*'s movement at its local coordinate system is noted as *rForearm-local*; the relationship between *rForearm* and *rShoulder* is noted as *rForearm-rShoulder*; the relationship between *rShoulder* and *Torso is rShoulder-Torso*; and so on. Thus, *rForearm*'s movement in the *World Reference Frame*, is expressed as *MTorso-World*🡐*rShoulder-Torso*🡐 *rForearm-rShoulder*🡐*rForearm-local*.

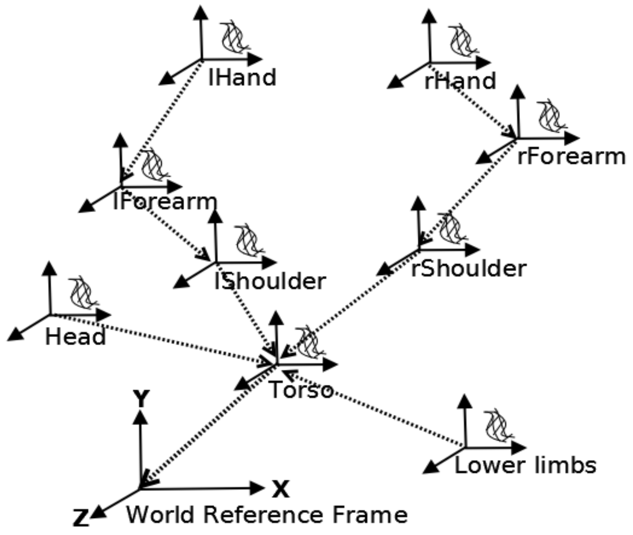


Figure 3: Coordination of the motions of the human body parts.

The body parts that make up the human body share some common features that take the form of user-defined data types, functions, and rendering algorithms for the modeling implementation. We use *xBody* for the abstract representative body part that accomplishes all operations (different motions such as rotation and bending) for these common features. For a particular body part, its deformation algorithms and movement patterns (represented as a set of parameters) are embedded into the body object upon its substantiation.

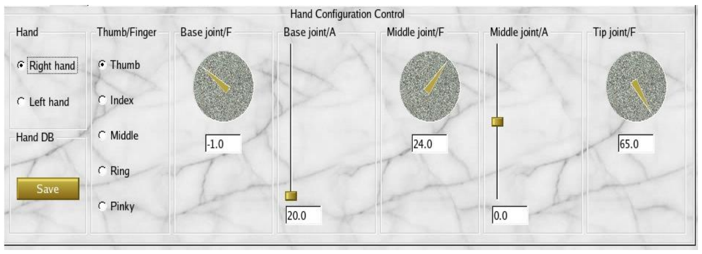
B. *Simulation of Body Motion*

A body part has its own movements and will influence and be influenced by others connected to it, depending on the relationship between them. More importantly, there is a deformation at the joint that connects the body parts. We have designed and implemented a *motion engine* which is embedded in each body part to "propagate" a body part's motions to its connecting body parts and to simulate the deformation around the connection joint.

When a body part (i.e., a cluster of 3D vertices) moves or other parts connected to the part move, different types of signals (different types of rotations) will be sent from this part to the motion engine connected to the part. These signals will trigger a set of calculations of new vertex positions, normals, and properties, depending on the types of the signals. The updated vertex data will be used for rendering the body part. The resulting outputs (including motion information of the local coordinate system for the body part will spread to the body part's immediate child (in the tree), which generates a chain of operations and renderings on the connected body parts (through the engines).

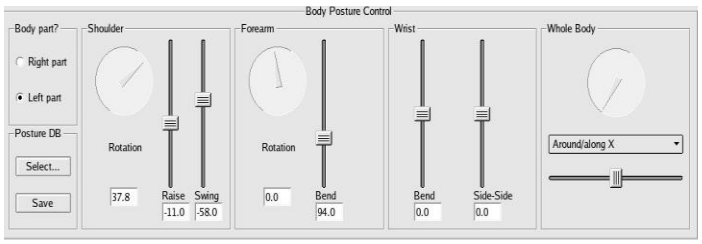
Simulation of the Hand Configurations and Motions

There are sixteen hand parts in a hand: a palm, three finger parts on each of the four fingers (the index, middle, ring, and pinky), and three thumb parts. All these sixteen parts are modeled with an ordered tree structure ([22]). The hand, when in motion under constraints, generates natural hand gestures. One such example is that bending of the middle finger will result in the follow-up movements on the index and ring fingers. The hand constraints have been implemented and the results from [22] have been incorporated in sEditor. Figure 4a is a screenshot of the control panel for generating hand configurations.



Hand: [✓ Right hand]
Thumb/Finger: [✓ Thumb]
Base joint/F: -1.0
Base joint/A: 20.0
Middle joint/F: 24.0
Middle joint/A: 0.0
Tip joint/F: 65.0

Figure 4a: Hand configuration control panel.



Body part?: [✓ Left part]
Shoulder: Rotation 37.8, Raise -11.0, Swing -58.0
Forearm: Rotation 0.0, Bend 94.0
Wrist: Bend 0.0, Side-Side 0.0
Whole Body: Around/along X

Figure 4b: Upper body posture control panel.

Modeling the Upper and Lower Parts of the Body

The upper body consists of the abdomen, chest, clavicle, shoulders, and part of the hips and neck. All of these parts are covered by clothing, and their movement patterns can be described with a tree structure. All nodes in the tree should be independent body part modules with an individual motion paradigm, but in our interfacing system we combine the hips, abdomen, chest, neck and clavicle into one large body part, as a rigid body part with deformation only on the borders with the shoulders. The right and left shoulders, together with their corresponding upper arms, are modeled separately. Thus, we have five upper body parts to be modeled individually: the main part of the torso (the hips, abdomen, chest and neck), right and left shoulders (including the upper arms), and right and left forearms (attached to the right and left shoulders in the tree). Particular attention was focused on the simulations of the shoulders and forearms and specific algorithms were implemented to deal with the deformations in the shoulder and forearm movements. Figure 4b depicts a screenshot of the control panel for the simulation of the upper body movements.

There are three widget groups on the panel for controlling the upper body movements: (1) *Shoulder:* the shoulder and upper arm's twist (rotation) are adjusted with an iconic dial, and the raising and swing movements are adjusted with two vertical slide-bars; (2) *Forearm:* the forearm's rotation (twist) is adjusted with a dial widget, and its bending movement is controlled with a vertical slide-bar; and (3) *Wrist:* the wrist's bending and side-to-side movements are adjusted with two vertical slide-bars. All these widget groups are under the control of another widget group on the same panel entitled *Body* part, which indicates whether the left body part or the right one will get inputs from the three widget groups. There is another widget group that controls the whole body's movements: *displacements* (translations) along and rotations around three perpendicular axes of the coordinate system for the whole body.

V. Creating Virtual Gestures

In this section, we discuss how to transcribe the movements generated from the virtual body and then how to create and coordinate the movements of individual body parts for the simulation of the human gestures.

A. *Parametric Representation of Human Gestures*

A virtual human body is made of many different body parts, each of which can be implemented with an abstract representative body part, *xBody*, with an extension based on its motion patterns and deformation methods. To control and simulate the movement of that body part, parametric values of the motions (translation and/or transformation) must be fed into that body part through an interface. This process is shown in Figure 5.

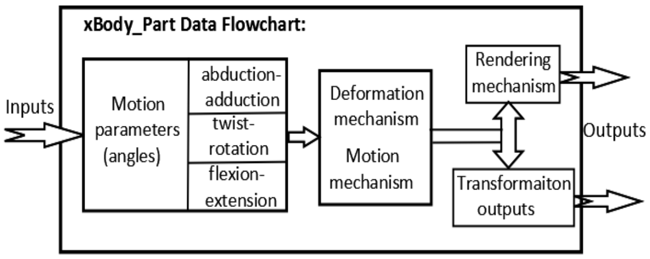


Figure 5: Process of how a body part module processes its input data and model the movements.

When a body part receives inputs, it will "interpret" the inputs, based on the nature of the body part, as values for some or all motion parameters for three types of rotations: abduction-adduction, twist-rotation, and flexion-extension. Then the deformation and motion mechanisms of the body part will calculate the new locations, normals, and transformations for the body part according to the motion patterns and parametric values. Finally, the calculations will be fed back for rendering this part and for updating its neighboring body parts.

Thus the movements of the virtual body can be described by and controlled with the inputs of the component body parts of the avatar, and a virtual gesture is a set of movements of the virtual body in a certain order. These inputs of the body components are a set of motion parametric variables with certain values, and therefore, a virtual gesture can be described with a cluster of sets of parametric variables. In the following we will illustrate the parametric representation of the gestures. But we will first give a definition of body posture and its data structure.

The Representation of Body Posture

Body posture means the position, pose, and bearing of the body, for example, sitting posture and erect posture. In sEditor, we extend this definition such that body posture defines the positions and bearing characteristics of all body parts including facial expression features and hand configurations. This makes it convenient to design and implement data structures that are used to represent and process the linguistic parts of a sign language. A data structure for body posture is defined as a tree structure (Figure 6).

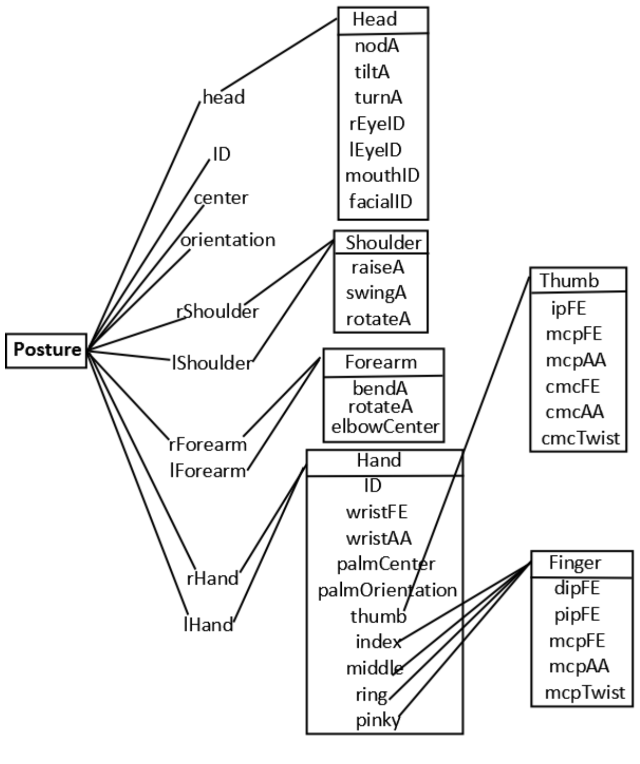


Figure 6: Data structure for body posture.

A posture contains the following elements:

1) *head:* for modeling the head movement (through parameters *nodA*, *tiltA*, and *turnA*) and for simulation of facial expressions, or NMS (through parameters *rEyeID*, *lEyeID*, *mouthID*, and *facialID*).
2) *ID:* a unique integer for the posture.
3) *center*: a point in 3D space (an array of three floating point numbers) for the center of the body.
4) *orientation:* a vector for identifying the body's orientation in 3D space.
5) *rShoulder* and *lShoulder:* for the description and control of the movements of both shoulders (through parameters of *raiseA*, *swingA*, and *rotateA*).
6) *rForearm* and *lForearm:* for the description and control of the movements of both elbows (through parameters of *bendA*, *rotateA*, and *elbowCenter*).
7) *rHand* and *lHand:* for the description and control of the movements of both hands (through parameters of *wristFE* for wrist's flexion-extension or rotation/bending) and *wristAA* for wrist's abduction-adduction or side-side movement), *palmCenter* and *palmOrientation* for the palm, *thumb*, *index*, *middle*, *ring*, and *pinky* for the thumb and fingers). Since the hand has the most complicated configurations, we assign each hand pose a unique *ID*.
8) *thumb:* for the description and control of the movements of the three thumb parts (through parameters *ipFE* for the flexion-extension of the thumb tip, *mcpFE* and *mcpAA* for the flexion-extension and abduction-adduction of the middle thumb part, and *cmcFE*, *cmcA*A, and *cmcTwist* for the movements of the thumb's base part).
9) *index*, *middle*, *ring*, and *pinky:* for the description and control of the movements of the three finger parts (through paramenters *dipFE* for the flexion-extension of the finger tip, *pipFE* for flexion-extension of the finger's middle part, and *mcpFE*, *mcpAA* and *mcpTwist* for the movements of the finger's base part).

The *elbowCenter* and *palmCenter* data structures record the positions of the hand and elbow and play an important role in classifying the virtual gestures and searching a sign language for linguistic parts. As a new posture is constructed, the positions for the elbow and the hand are automatically calculated by the system and become a part of the parametric representation of the posture.

The Representation of Body Gestures

People make gestures by starting with a posture and ending with another posture, assuming a series of varying postures in between. In a similar way, a virtual gesture can be described as an *ordered* set of postures of a virtual human body. A timing factor is thus introduced to describe the order of the postures: posture p_i occurs at time t_i where p_i is a parametric representation of the posture at time t_i and has a data structure defined in the previous section. Thus we use a list to define the gesture, $vg = [(p_0, t_0), (p_1, t_1), \dots (p_n, t_n)]$ in which p0 is the starting posture at time t0 and p_n is the ending one at time t_n.

There is a big difference between a human's gesture in real life and the gesture defined above: the former is a continuous process, which means an infinite number of postures in a gesturing session, while the latter is only a limited number of postures in a posture list. We address this problem in two steps. First, for a virtual gesture, a group of distinctive postures is selected (like key frames in video) that reflect characteristics of the gesture; then temporary postures between two adjacent postures (i.e., key frames) are interpolated during the output process of the virtual gesture based on the rendering speed (frames per second) and time difference between the two adjacent postures (i.e., key frames).

B. Construction and Management of Body Postures

As illustrated above, body posture in the sign language interfacing system is interpreted as a set of parametric variables that describes the distinctive features of a body's bearing. According to the characteristics of the body parts and their functionalities in a signing process, we classify the parametric representations of the body parts into three separate groups: hand configuration, upper limb positioning, and NMS (nonmanual signal).

We now discuss the construction and management of virtual postures in the interfacing system. We focus on the introduction of the functions (wrapped in a graphical user interface) of these operations. The functions are "wrapped" in an efficient graphical user interface, through which body postures are created and edited by providing and adjusting parametric values for the posture's representative parametric variables. The management of body postures such as storing, editing, and retrieving of the postures is handled by a posture database.

The Hand Configurations

A *Hand Configuration Control* panel has been built and embedded in sEditor (see Figure 4a) for the creation and editing of hand shapes. Graphical widgets are used to provide and adjust values (degrees of rotation angles) of the parametric variables for a certain hand configuration.

A hand configuration database is used to assist in the creation, editing, and management of the hand configurations. We constructed several dozen hand shapes (like in Figure 2), stored them into the hand configuration database, and embedded them into the system.

To create a new hand shape, we first search for a basic hand configuration in the database that has a similar pattern. If we cannot find one, we use a default hand shape with a neutral position. Then we use the hand configuration control panel to fine-tune the angles of the hand's joints (including the wrist's joints). Finally, the newly created hand configuration is saved in the database. The hand shapes can be applied to both hands of the virtual body.

The Upper Body Postures

The upper body parts of the virtual figure in the interfacing system include the shoulder (together with the upper arm), forearm, and the wrist joint, responsible for the hand (palm) orientation. A body posture control panel is incorporated into the sign language interfacing system for providing inputs for upper body part joints as (Figure 4b), using the data structure previously described.

A body posture database is used together with the NMS and hand configuration databases for storing, editing, and retrieving body postures. Some basic body postures are included in the posture database.

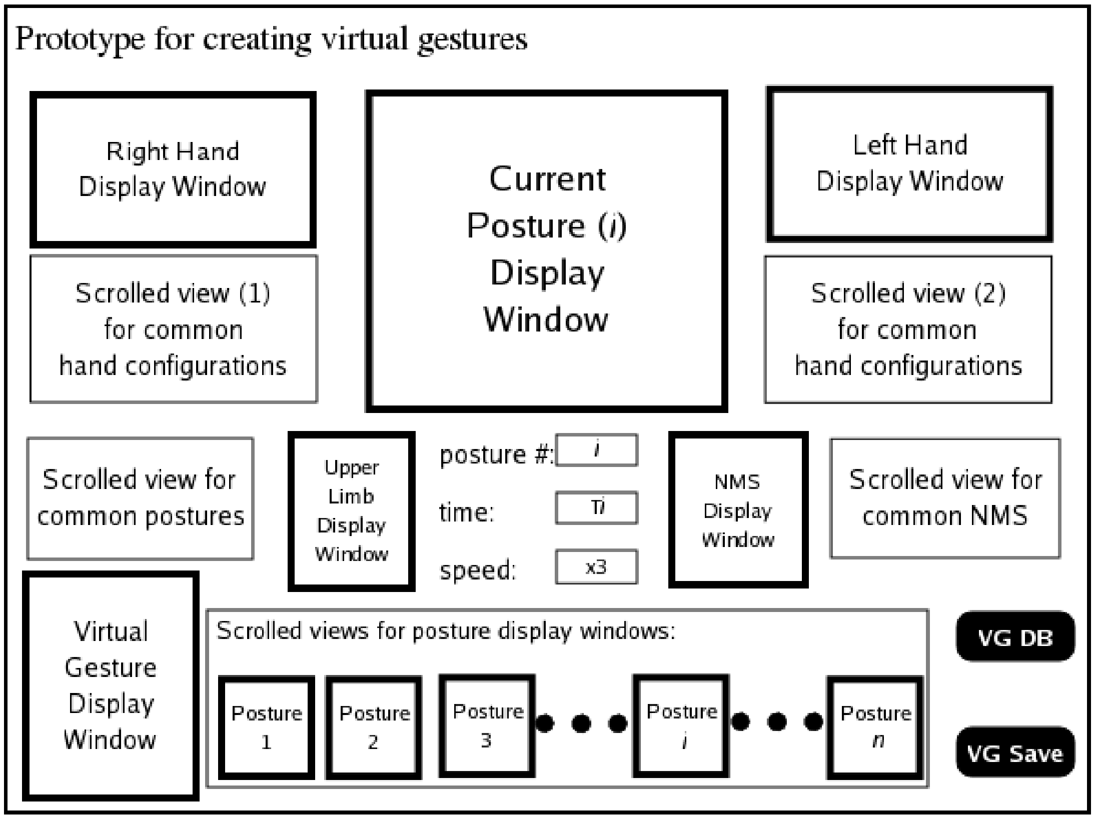
When a new body posture is created, the parametric values corresponding to the posture's data structure (as defined above) can be stored in the body posture database. Of the parameters, two groups are critical for categorizing and designing virtual gestures: the locations of the center of each hand palm and of the center of each elbow in the coordinate system for the whole virtual body. These locations are automatically calculated based on the inputs to the body posture control panel.

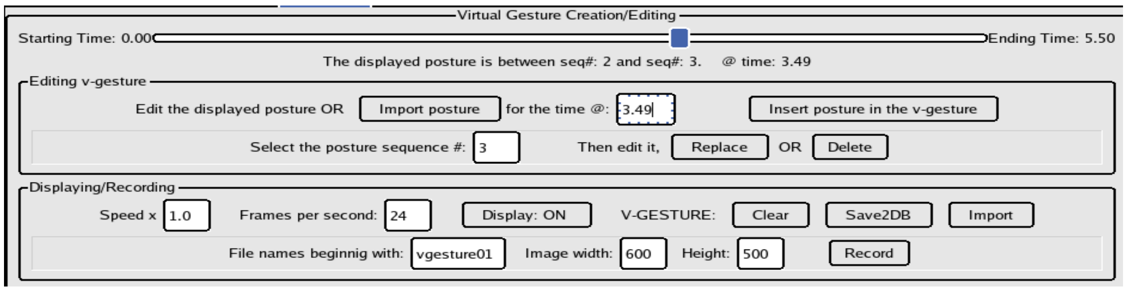
C. *Creation and Management of Virtual Gestures*

We have defined a virtual gesture *vg* as $[(p_0, t_0), (p_1, t_1), ..., (p_n, t_n)]$ with p_i being the *i*th posture in *vg* at the time t_i. The postures $p_0, p_1, …, p_n$ constitute a *complete* set of postures for a *given gesture* and are the most representative and characteristic postures for that gesture, which describe the gesture process. Once a posture list *vg* is extracted for a certain gesture, intermediate and temporary postures can be interpolated between any two adjacent postures in the list for display and output. The problem of creating a virtual gesture is how to construct such a posture list given the gesture. We next describe the architecture and implementation of the construction of virtual gestures.

The Architecture for Creating and Editing Virtual Gestures

Figure 7a illustrates the architecture for creating and editing virtual gestures and Figure 7b shows the implemented control panel). In the figure, the boxes with bold edges represent display windows for displaying the temporary and overall


Prototype for creating virtual gestures
Right Hand Display Window
Current Posture (i) Display Window
Left Hand Display Window
Scrolled view (1) for common hand configurations
Scrolled view (2) for common hand configurations
Scrolled view for common postures
Upper Limb Display Window
posture #: i
time: Ti
speed: x3
NMS Display Window
Scrolled view for common NMS
Virtual Gesture Display Window
Scrolled views for posture display windows:
Posture 1
Posture 2
Posture 3
Posture i
Posture n
VG DB
VG Save


Virtual Gesture Creation/Editing

Starting Time: 0.00 Ending Time: 5.50

The displayed posture is between seq#: 2 and seq#: 3. @ time: 3.49

Editing v-gesture

Edit the displayed posture OR [Import posture] for the time @: [3.49] [Insert posture in the v-gesture]

Select the posture sequence #: [3] Then edit it, [Replace] OR [Delete]

Displaying/Recording

Speed x [1.0] Frames per second: [24] [Display: ON] V-GESTURE: [Clear] [Save2DB] [Import]

File names beginnig with: [vgesture01] Image width: [600] Height: [500] [Record]

Figure 7a (above): A prototype for creating virtual gestures; Figure 7b (below): Virtual gesture creation/editing panel.

results in the gesture construction process. In the upper part (corresponding to that of Figure 2) there are right and left hand display windows on either side of a main display window for displaying the current posture of the whole body. In the lower part, there are NMS and upper body posture display windows, the virtual gesture display window (for displaying the gesture animation process), and posture display windows (for displaying all of the postures of the gesture).

The other boxes are used for the interactions with the databases and operations on gesture controls (such as setting time and speed). When users are constructing a virtual gesture, they first select from repertoires of upper body postures, hand configurations, and NMS—the representative components for the gesture. If some components are not in the databases, users can use the corresponding control panels to create and save them to the databases. When there are some components that are close to the desired ones, the users can select them and use the control panels to fine-tune them. In the following discussion we assume that such components exist in the databases.

For example, when a user is about to insert an *i*th posture into the virtual gesture, she/he first clicks on one posture icon in the box and a pop-up window appears containing postures close to (or related to) the posture. The user then chooses one posture from the window, and the selected posture replaces the old posture of the virtual body and is displayed in both the *Upper Limb Display Windows*.

When this process is completed, the posture *i* has been inserted into the virtual gesture. *Virtual Gesture Display Window* then automatically shows the gesture animation session based on the speed setting. On the right of the virtual gesture display window are display windows for all postures of the current gesture. A user can click any of them to edit the gesture with the posture control panels and change the time setting. The newly created virtual gesture can be saved to the gesture database with push button *VG Save*.

The Virtual Gesture Creation/Editing Panel

With the gesture creation prototype (Figure 7a) as a guide, we have implemented a virtual gesture creation and editing interface (Figure 7b) and incorporated it into the sign language interfacing system. Users use this interface, together with the posture control panels, to create, edit, store, and retrieve any virtual gestures.

At the top of the gesture interface, there is a posture sequence sliding bar and a gesture information line below it. Users use the sliding button on the bar to display (on the upper part of the interfacing system (Figure 2)) the postures of the current gesture, either those representative postures (p_0, p_1, …, in the *vg*) or any temporary interpolated ones between any two adjacent postures in the *vg*. The middle part of the interface is used for posture editing functions: importing (from the posture database), editing, replacing, deleting, and inserting (into the current gesture). The bottom of the interface is reserved for displaying, saving, and recording the current gesture.

At the beginning, the gesture is made up of any two postures distributed at time 0 and 2 (seconds). Users can replace these postures with postures from the posture database with graphical widgets such as the push buttons *Import posture*, *Replace*, *Insert posture in the v-gesture* and the input space *Select the posture sequence #* to type posture numbers of postures to be edited or replaced. The timing factors for new postures are input in space entitled *for the time @*. The current posture can also be deleted with the Delete button.

The gesture session can be displayed dynamically depending on the status of the *Display* switch button, which appears in only one mode: *Display: ON* or *Display: OFF*. The gesture display sliding bar and the sliding button on it are used for accurate control of the gesture postures.

An Example of Creating and Editing a Gesture

Suppose we are about to create a gesture that has the four characteristic postures shown in Figure 8a. These postures (from left to right in this figure) will appear in the gesture at time (in seconds) 0, 1.5, 2.8, and 4.2. These postures are loaded from the posture database (or created instantly with the use of the control panels introduced above) and are inserted in the gesture (the posture at time 2:0 is deleted). Now we select the *Display* switch button, and the display windows will display the gesture animation process, which lasts 4.2 seconds with the default speed of 24 frames per second. In this case, sEditor will have automatically interpolated about 97 (*i.e.*, 24 x 4.2 - 4) intermediate postures for this gesture.

Testing the gesture process we discovered that some unnatural postures were generated and interpolated at 3.12 seconds (see Figure 8b).

We used sEditor control panels to edit these (interpolated) postures by adjusting the left and right shoulders' (and forearms') positions with the widgets on the *Body Posture Control* panel. The modified postures were automatically recorded and combined as the characteristic postures for the gesture. These changes resulted in a new and natural posture shown in Figure 8c. The gesture can be saved into the gesture database with the button *Save2DB* and recorded in image files with the button *Record*.

VI. Conveying Linguistic Meaning

A signer of a particular sign language makes gestures according to the grammar of that sign language; an avatar can also imitate this process by following commands on the movements of the virtual body parts if these movements are designed to abide by grammatical rules of that sign language. Thus a virtual gesture session, virtual signing, acquires a meaning, and the virtual body makes virtual signs.

In the following we describe how to use this system to build basic linguistic parts (such as “phonemes” and “morphemes”) of sign languages, create sign language vocabularies, and even “write” in a sign language (a control panel for this task is shown in Figure 9). America Sign Language (ASL) is used as an example.

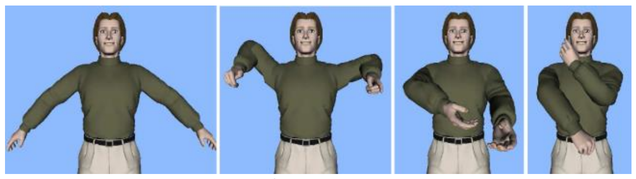


Figure 8a: A virtual gesture: its four posture components.

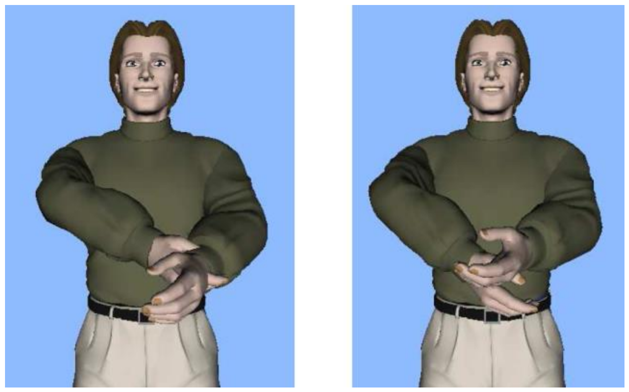


Figure 8b: How to create a natural gesture: (left) unnatural temporary postures interpolated during virtual gesture rendering; (right) choose a representative posture of these unnatural postures, edit it, and insert it to the gesture as a component posture for that gesture.

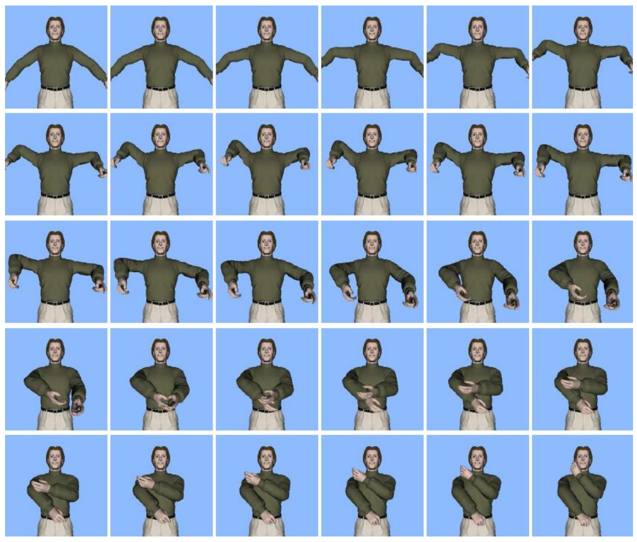


Figure 8c: An example of a virtual gesture sequence.

A. *Constructing Basic Linguistic Parts*

The concept of "articulatory bundle" [23], which describes hand posture with hand configuration, point of contact, facing, and orientation, provides good guidelines for designing virtual signing units. However, it is more effective to use graphical designs and implementations when dealing with the five basic linguistic parameters of a sign language: location, handshape, orientation, movement, and NMS [2]. We have also considered "local movement," a special case of the movement parameter. The Movement-Hold model [23] is embedded in the graphical implementation.

The five basic linguistic parts (parameters) can be simulated with a list of virtual gestures (defined above) combined with timing factors: $lp = [(vg_0, t_0), (vg_1, t_1), ..., (vg_n, t_n)]$ (lp represents any of the basic linguistic parts), which was a long sequence of postures. The question becomes how to quickly construct the vg_i's and combine these vg_i's with their t_i's. Our solution is to use an efficient GUI wrapper for the operations needed for the creation and editing of the basic linguistic parts.

Hand Shapes and Orientations

The hand shape is the most important phonological part of ASL and other sign languages; thus we have constructed some of the most frequently used hand shapes (as shown in Figure 2) and embedded them in the sign language interfacing system. The hand shapes can be applied to both the left and right hands of the avatar in the system with only mouse-clicks on the interface. New hand shapes can be built and embedded into the system.

For some hand shapes with many variations and/or other hand shapes related to them, there will not be enough space in the scrolled view areas to display them. One solution would be to activate a pop-up window box with related hand shapes and variations when the user clicks on the hand icons.

The hand (palm) orientation is dependent on the movements of the other body parts (such as forearms) and is relatively independent of hand shapes. The movements at the wrist joint also affect the hand orientation. Thus we use a neutral orientation as a default for all hand shapes before their application to the avatar. When being applied to the virtual body, a hand (shape) immediately takes on the orientation defined by the other body parts.

Gesture Space and Locations

The gesture space is the space domain of the hand motions when people make gestures and this space is divided into different sectors [24]. Liddell and Johnson's description of "point of contact" (POC) and their classification of about 20 major body locations provides direct guidance [23] for the implementation of the hand locations in virtual signing.

We have applied heuristic methods (together with the POC concept and the implementation of hand constraints) to classify and record the hand and elbow's locations. When a posture is created for the avatar, the locations of its hands and elbows are automatically calculated. These locations are part of the parametric representation of the posture and are stored in the posture database. When searching for a particular posture, we can use these locations to narrow down the search space.

A hand's location (palm center) is classified with three types of location: *hand height*, *hand depth*, and *hand across*.

- *Hand height* describes how high the palm center is from the ground. Its range is divided into *High*, *Mid*, and *Low*.
- *Hand depth* measures how far away the palm center is from the chest. Its range is divided into *Far*, *Mid*, and *Close*.

- *Hand Across* identifies the palm center with a horizontal right-left cross line. For example, if the right hand rests on the right side, it is marked as *Close*; when it goes across the chest to the left side, it will be on the Far side. This parameter is divided into three ranges: *Close*, *Middle*, and *Far*.

With this definition and classification, the hand's location can be represented with a set of three variables, [*across*, *depth*, *height*], each of which has one of three different values in its range domain as defined above. There are 3x3x3 = 27 different combinations to describe a hand's location. In other words, a hand's location will be in one of the 27 cubes in front of the signer, defined by three perpendicular axes in *the body coordinate system* marked with *across*, *depth*, and *height* in the virtual body's coordinate system with its origin at the body center.

The elbow's location is described with only one variable, *height*, which has one of three values *High*, *Mid*, and *Low*. Now with the consideration of locations of the avatar's two hands and two elbows, we have 3 x 3 x 3 x 2 x3 x 3 x 3 x 2 = 6561 different combinations of the hand and elbow's location, which means that we can divide the signing locations into 6561 different groups.

sEditor provides GUI interfaces to searching for postures or close ones based on the user’s selections of location parameter values of the hand and elbow. The resulting postures and the ones during the searching are rendered and displayed in real-time as a sequence of postures and the user may choose one or more of them (two such control panels are shown in Figure 7b and Figure 9).

Movements

According to [25] adding linguistically motivated pauses in sign durations will make the sign animations more understandable by the native ASL signers. Signs can be described with Movement-Hold mode and are composed of sequentially produced movement segments and hold segments [2]. In our implementation, we adapted this model but “disregard” the linguistic implications of a virtual sign. By movements, we mean any kind of movements used in signing and define a movement segment, *mSeg*, of a sign: $mSeg = [(p_0, t_0), (p_1, t_1), \ldots, (p_n, t_n)]$ in which any adjacent pair of $[p_0, p_1, \ldots, p_n]$ will be different from each other because during movement the articulation of linguistic parts is always in a state of transition.

A hold segment is a pair of the same posture that occurs sequentially at *different* times. For example, if the *i*th segment of a sign is a hold segment, $hSeg_i$, then, $hSeg_i = [(p_i, t_i), (p_{i+1}, t_{i+1})]$ where $p_i = p_{i+1}$ and $t_i \neq t_{i+1}$. For a virtual sign, vSign, which is composed of movement and hold segments, we have: $vSign = [mSeg_0, mSeg_1, hSeg_2, \ldots, mSeg_i, \ldots, hSeg_j, \ldots, mSeg_n]$ where the order of the movement and hold segments depends on the contents of the sign.

B. *Composing Other Linguistic Parts*

The basic linguistic parts share a common representation form: $lp = [(p_0, t_0), (p_1, t_1), \ldots, (p_n, t_n)]$ and are used to compose other linguistic parts. As in spoken language, given some basic linguistic parts (phonemes and some morphemes), we can build larger linguistic parts (*LP*) such as morphemes, words, phrases, and even sentences, which can be represented as a combination of the very basic linguistic parts, which turn out to be an ordered list of postures: $LP = [lp_0, lp_1, \ldots, lp_m] = [(p_0, t_0), (p_1, t_1), \ldots, (p_N, t_N)]$, where lp_i indicates a basic linguistic part.

Theoretically, we can use the formula, $LP = [(p_0; t_0), (p_1; t_1), \ldots, (p_N; t_N)]$, to construct a sign language's words, phrases, and even sentences—that is, to create or retrieve from databases every posture, $p_0, p_1, \ldots, p_N$. This means that the size of *LP* will become too large. So we have to use another formula: $LP = [lp_0, lp_1, \ldots, lp_m]$. But there is a problem in sign languages with the transition between two adjacent postures, for example, movement epenthesis, hold deletion, and assimilation in ASL. Our solution is to use both of them: $LP = [lp_0, lp_1, \ldots, lp_m] = [(p_0, t_0), (p_1, t_1), \ldots, (p_N, t_N)]$. First, we give some definitions:

1) We define *wordItem* to be any of the linguistic parts, either basic or larger ones.
2) Every *wordItem* has several (zero to any number in theory) keywords or related words associated with it. In the case of the creation of words and phrases for a sign

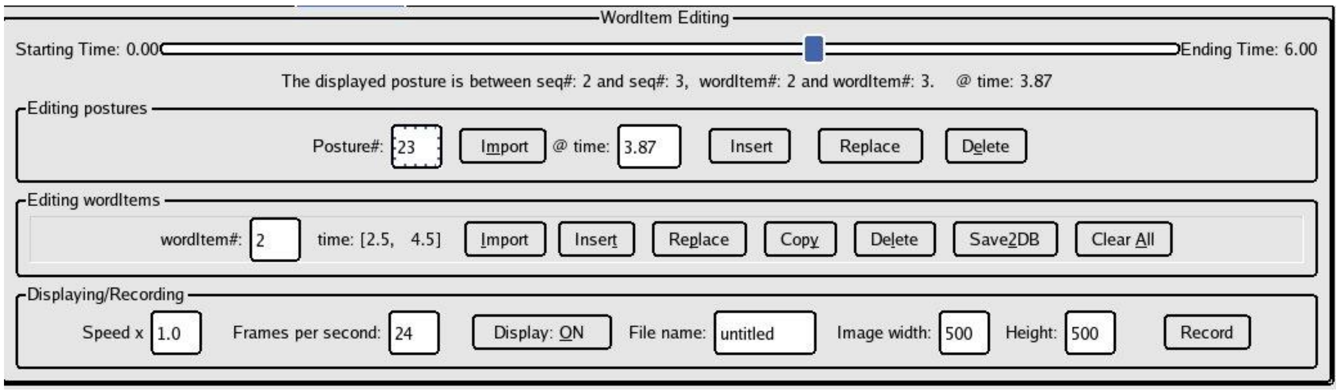

WordItem Editing
Starting Time: 0.00 Ending Time: 6.00
The displayed posture is between seq#: 2 and seq#: 3, wordItem#: 2 and wordItem#: 3. @ time: 3.87
Editing postures
Posture#: 23 @ time: 3.87
Editing wordItems
wordItem#: 2 time: [2.5, 4.5]
Displaying/Recording
Speed x 1.0 Frames per second: 24 File name: untitled Image width: 500 Height: 500

Figure 9: A general control panel for creating and editing linguistic parts.

language, we use *related words* for the association; in other cases, use *keywords*. But for the current version of our sign language interfacing system, we use both of them interchangeably. Thus we have following representations:

- *wordItem* = {lp_i}: combination of any number of basic linguistic parts.
- *wordItem* = {LP_j}: combination of any number of larger linguistic parts.
- *wordItem* = {lp_i, LP_j}: combination of any number of basic and larger linguistic parts.
- *wordItem* = {$mSeg_i$, $hSeg_j$}: combination of any number of movement segments and hold segments in the Movement-Hold model representation.
- *wordItem* = {lp_i, LP_j, KW_m} or *wordItem* = {lp_i, LP_j, RW_m}: combination of any number of basic and larger linguistic parts and associated *keywords* or *related words*.
- *wordItem* = {lp_i, [(p_k, t_k)], LP_j, KW_m} or *wordItem* = {lp_i, [(p_k, t_k)], LP_j, RW_m}: same as above, but [(p_k, t_k)] indicates the inserted postures modified postures in {lp_i} or {LP_j} .

To construct a graphical user interface for creating and editing linguistic parts, we have considered the following requirements for such an interface:

1) It should be able to create and edit postures, and store to and retrieve these newly built postures.
2) Based on the posture database, the interface should be able to create basic linguistic parts with time controls, edit them dynamically, store them to and retrieve them from a linguistic part database.
3) It should be able to create large linguistic parts from the posture database and linguistic part database and input *keywords* or *related words* for them.
4) It should provide, if possible, an editing mechanism for editing both the posture constituents *and* the linguistic part constituents for a linguistic part.

These requirements for the linguistic part creation and management interface were implemented in a control panel (Figure 9) and its associated databases; the user can insert, delete, and edit the postures and the linguistic parts. The results are displayed in pop-up windows and can be saved in the database. Figures 10a and 10b give two examples of virtual signing for an ASL word and one sentence.

VII. Discussion

sEditor is a prototype sign language interfacing system for creating and managing sign language linguistic parts. The system provides a GUI interactive mechanism for the creation of correct basic signs (Figures 4c, 4d, 7b, 8b, and 9).

With the use of virtual gesture and sign databases, the users of sEditor can construct, save, retrieve, and edit basic linguistic parts and then build larger linguistic parts such as words, phrases or sentences based on the basic ones (as shown in Figures 10a, 10b, and 10c) at “lower level” (i.e., without consideration of the grammatical conjugations), which may not be correct unless under intensive examination and with

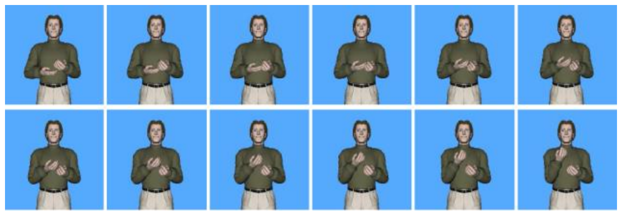


Figure 10a: Virtual signing output of an ASL word: EASY (the session should repeat once more time).

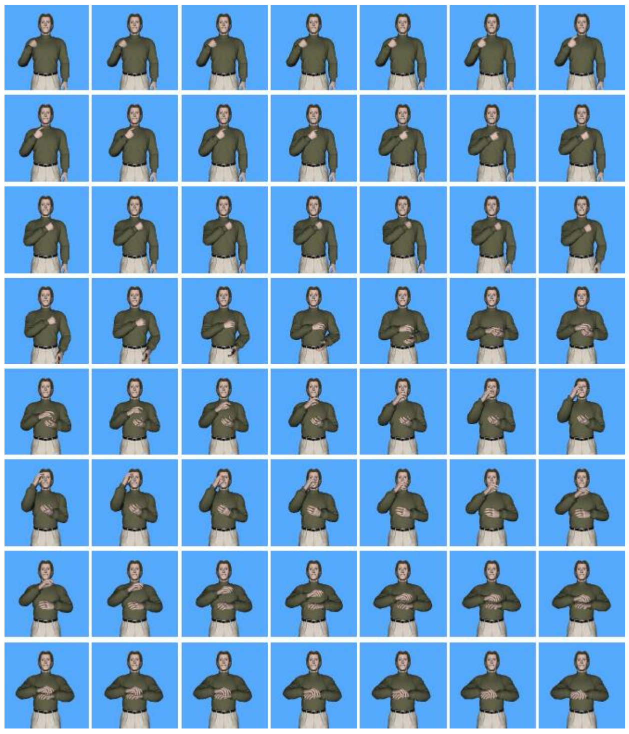


Figure 10b: Virtual signing output (signed English) of an ASL sentence: WE LEARN ENGLISH.

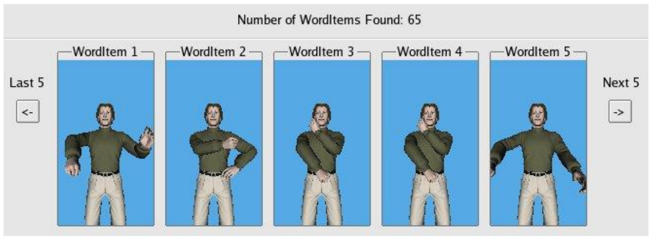


Figure 10c: An example of how to quickly retrieve a sign language \word": these display windows display virtual signs (animation sessions) for different but related (associated) \words"; the word “WordItem" above the display window is supposed to be notation symbols of a sign language for the sign below it. The user will have three input methods to choose from: (1) clicking on the virtual sign in a display window, (2) typing in the notation symbols, and (3) typing the number (1, 2, …) above the virtual sign.

necessary corrections (as shown in Figure 8b).

It would be an ultimate goal for our sign language interfacing system to become (or at least, give a direction for creating) a sign language "editor" like a text editor (such as Microsoft Word) for the spoken languages, in which users can "write" with the system. There are two major problems to consider for designing a sign language "editor": (1) how to retrieve (input) sign language "words" and (2) how to deal with the transition between two adjacent "words" following the sign language syntax.

As for the first problem, we can borrow methods such as *autocomplete* used in several Asian language text input techniques. When one clicks on a sign or types in the transcription code for a sign, the signs related to that sign (e.g., with higher associated weights or sharing the first transcription coding symbols) will be displayed on the screen, each of which is an animation sequence accompanied by a number or notation symbols, rendered in an easy-to-understand style in a small screen area. One can click on the desired sign or type in its representative number or notation symbols. Figure 10c gives an explanatory example without consideration of the association weights of the individual signs.

The solution to the second problem is much more challenging. In a text editor for spoken languages, letters, words, and phrases are sequentially juxtaposed, but in a sign language there is a transitional process between two signing parts in which the two parts exert influence over each other, following the *syntax rules* of a sign language. This means that postures (including their corresponding time factors) on the border of two adjacent signs have to be changed. For example, there are four typical variations in a phonological process in ASL: movement epenthesis, hold deletion, metathesis, and assimilation [24]. In their computer graphics implementations, this presents a movement-control-over-time design issue. sEditor transits from one sign sequence to another one with the use of interpolation (inserting "mid-signs" based on the two adjacent signs) without consideration of the syntactical rules.

REFERENCES

[1] S. K. Liddell, "Grammar, Gesture, and Meaning in American Sign Language," Cambridge University Press, 2003.

[2] C. Valli and C. Lucas, "Linguistics of American sign language: an introduction," Gallaudet University Press, 3rd edition, 2000.

[3] B. Yi. "A Framework for a sign language interfacing system," PhD dissertation, Department of Computer Science and Engineering, University of Nevada, Reno, 2006.

[4] J. Loomis, H. Poizner, U. Bellugi, A. Blakenore, and J. Hollerbach, "Computer graphic modeling of American sign language," *ACM SIGGRAPH Computer Graphics*, vol. 17, pp. 105-114, July 1983.

[5] W. C. Stokoe, D. C. Casterline, and C. G. Croneberg, *A Dictionary of American Sign Language on Linguistic Principles*, Linstsok Press, 1976.

[6] S. Wilcox, "The multimedia dictionary of American sign language: learning lessons about language, technology, and business," *Sign Language Studies,* 3(4), pp. 379-392, Summer 2003.

[7] C. Meidle, S. Sclaro, and V. Athitsos, "SignStream: a tool for linguistic and computer vision research on visual-gestural language data," *Behavior Research Methods, Instruments, & Computers*, vol. 33(3), pp. 311-320, 2001.

[8] C. Neidle, "A database tool for research on visual-gesture language," report no. 10, ASL Linguistic Research Project, August 2000 [Online]. Available: http://www.bu.edu/asllrp/rpt10/ASLLRPr10.pdf (URL) [accessed, 1/4/2014].

[9] P. Lu, "Modeling animations of American sign language verbs through motion-capture of native ASL signers," *ACM SIGACCESS Accessibility and Computing*, 96, pp. 41-45, Jan. 2010.

[10] H. Kaneko, N. Hamaguchi, M. Doke, and S. Inoue, "Sign language animation using TVML," in *2010 Proc. 9th ACM SIGGRAPH Conf. on Virtual-Reality Continuum and its Applications in Industry*, New York, 2010, pp. 289-292.

[11] D.I. Fels, J. Richards, J. Hardman, S. Soudian, and C. Silverman, "American sign language of the web," in *Proc. CHI EA '04 Human Factors in Computing Systems*, pp. 1111-1114, CHI 2004.

[12] UPENN HMS Center, http://hms.upenn.edu/, [accessed, 1/4/2014].

[13] Virtual reality lab, http://vrlab.epfl.ch/, [accessed, 1/4/2014].

[14] DePaul ASL Synthesizer, http://asl.cs.depaul.edu, [accessed, 1/4/2014].

[15] eSign: Vitural Human Signing at UEA, http://www.visicast.cmp.uea.ac.uk/, [accessed, 1/4/2014].

[16] Vcom3d, http://vcom3d.com/, [accessed, 1/4/2014].

[17] M. Huenerfauth, "A survey and critique of American sign language natural language generation and machine translation systems," technical report, Computer and Information Sciences, University of Pennsylvania, September 2003. [Online]. Available: http://www.cis.upenn.edu/grad/documents/huenerfauth.pdf (URL) [accessed, 1/4/2014].

[18] M. Huenerfauth, L. Zhao, E. Gu, and J. Allbeck, "Design and evaluation of an American sign language generator," in *2007 Proc. Workshop on Embodied Language*, Prague, Czech Republic, 2007, pp. 51-58.

[19] N. Frishberg, S. Corazza, L. Day, S. Wilcox, and R. Schulmeister. "Sign language interfaces," in *Proc. CHI/INTERACT '93 conf. Huamn Factors in Computing Systems*, pp. 194-197, 1993.

[20] S.C.W. Ong and S. Ranganath, "Automatic sign language analysis: a survey and the future beyond lexical meaning," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 27, pp. 873-891, Jun. 2005.

[21] B. Yi, F. C. Harris, Jr., and S. M. Dascalu, "From creating virtual gestures to 'writing' in sign languages," in *proc. CHI EA'05 Conference on Human Factors in Computing Systems* (CHI 2005), Apr. 2005, pp. 1885-1888.

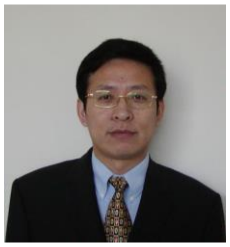
[22] B. Yi, F. C. Harris, Jr., and S. M. Dascalu, "Real time natural hand gestures," IEEE *Computing in Science and Engineering*, vol. 7, pp. 92-97, May 2005.

[23] S. K. Liddell and R. E. Johnson, "American sign language: the phonological base," *Sign Language Studies*, 64, pp. 195-227, Fall 1989.

[24] D. McNeil, "Hand and mind: what gestures reveal about thought," *The University of Chicago Press*, Chicago, 1992.

[25] M. Huenerfauth, "A linguistically motivated model for speed and pausing in animations of American Sign Language," *ACM Trans. Accessible Computing*, vol. 2, article No. 9, Jun. 2009.

Beifang Yi is currently an assistant professor in the Department of Computer Science at Salem State University, Salem, Massachusetts, USA. He received his MS in in Computer Science from Southwest Jiaotong University, Chengdu, China in 1988 and Ph.D. in Computer Science and Engineering from the University of Nevada, Reno, USA in 2006. His main research interests are in the areas of human-computer interaction, information visualization, computer graphics, and education in the computer science.



Xusheng Wang is an Associate Professor in the Department of Mathematics, Computer Science and Cooperative Engineering at the University of St. Thomas, Houston, Texas, USA. He received his MS in Computer Science from Southwest Jiaotong University, Chengdu, China in 1986 and Ph.D. in Information Technology with concentration in Computer Graphics from George Mason

University, Fairfax, VA, USA in 2003. His main research interests are in the areas of computer graphics, virtual reality, human-computer interaction, and information visualization. He has published over 20 peer-reviewed papers.

Frederick C. Harris, Jr. is currently a Professor in the Department of Computer Science and Engineering and the Director of the High Performance Computation and Visualization Lab and the Brain Computation Lab at the University of Nevada, Reno, USA. He received his BS and MS in Mathematics and Educational Administration from Bob Jones University in 1986 and 1988 respectively, his MS and Ph.D. in Computer Science from Clemson University in 1991 and 1994 respectively. He is a member of ACM (Senior Member), IEEE and ISCA (Senior Member). His research interests are in parallel computation, computational neuroscience, computer graphics and virtual reality.

Sergiu M. Dascalu is an Associate Professor in the Department of Computer Science and Engineering at the University of Nevada, Reno, USA, which he joined in 2002. In 1982 he received a Master's degree in Automatic Control and Computers from the Polytechnic University of Bucharest, Romania and in 2001 a PhD in Computer Science from Dalhousie University, Halifax, NS, Canada. His main research interests are in the areas of software engineering and human-computer interaction. He has published over 140 peer-reviewed papers and has been involved in numerous projects funded by industrial companies as well as federal agencies such as NSF, NASA, and ONR. In 2011 he received the UNR Donald Tibbitts Distinguished Teacher of the Year award.