A new method of hand gesture configuration and animation

Nicoletta Adamo-Villani,
Department of Computer Graphics Technology, Purdue University, West Lafayette, IN, USA.

Gerardo Beni,
Department of Electrical Engineering, University of California at Riverside, Riverside, CA, USA.

Abstract

We have developed a new keyboard-based method for configuring and animating hand gestures. Hand configuration and animation is important to many areas such as: (1) sign language education; (2) education and training in professions that require high levels of dexterity (e.g., surgery, dentistry etc.); (3) technical illustration; (4) art education; (5) character animation; (6) visual communication gestures in general. Our method is based on the realization that the hand has 26 degrees of freedom which can be controlled by the 26 letters of the English alphabet. Via keyboard input the hand can be positioned in space and manipulated to attain any configuration. The keystrokes corresponding to a particular hand pose are recorded and reduced to an alphanumerical compact code. The codes can be used as keyframes to produce animation sequences, stored in text files for later retrieval and applied to different hand models. The method presents several advantages over standard computer animation techniques such as: ease of use (any user with no animation skills can quickly produce a large number of hand configurations and animations); accuracy (even if the method produces a discretized representation of joint rotations and translations, the resolution of the discretization can be tailored to suit different levels of hand configuration precision and complexity); flexibility (the method can be applied to a variety of hand models created with different software packages); simplicity of storage and transmission (configurations and animations can be stored in text files of minimum size and transmitted via internet).

Keywords: 3D Animation, HCI (Human-Computer Interaction), Educational Technology, 3D Visualization.
1. Introduction

There is a need for modeling, configuring and animating hands. Hand representation plays a special role in artistic compositions [1], [2], [3] and in character animation. Research shows that the audience is drawn to the eyes of the animated character first, and second to its hands to better understand its personality and emotional state [4]. Because hands respond unconsciously to mental states, they often reveal what the face tries to conceal [1].

Accurate representation of hand configuration and motion is crucial to teaching signed communication, e.g., American Sign Language (ASL) [5]; to communicative gestures in general, e.g., Human Computer Interface (HCI) visual recognition gestures [6]; to teaching dynamics manipulative tasks as, e.g., musical instrument playing, sport devices handling, tools handling; and to teaching fine manipulative skills as, e.g., dentistry, surgery, and mechanics.

With its 27 bones and 26 degrees of freedom, the skeleton of the hand is the most complex part in a human model. To accurately reproduce the infinite number of hand configurations and motions, the animator needs to have a solid understanding of the mechanics of the hand as well as a deep knowledge of the 3D animation software.

Currently, the majority of 3D character animation software packages offer Graphical User Interfaces (GUIs). Generally, once the skeleton has been created, the animator selects the individual joints and/or the Inverse Kinematics (IK) handles in the 3D scene and applies a series of transformations (rotations and translations) to attain a particular hand configuration.

Many 3D packages (such as Maya 5.0) allow the creation of customized Graphical User Interfaces for modelers and animators to facilitate and speed up the selection and transformation of the character’s components. Typically, for character animation, the user points and click at joints and control handles at the exact body location on a static reference image in an ad hoc window [7]. The motion of the joints is controlled by sliders included in another GUI window.

In Poser 5 the user can select a hand configuration from the “hands library” and accept the pose completely or use it as a basis for making further modifications [8]. In order to modify a particular library pose or to reach a hand configuration non-existent in the hands library, the user poses (rotates) each joint individually.

Even with a customized and user-friendly Graphic User Interface or with access to a large library of pre-made hand configurations, the process of configuring and animating the hand is tedious and time consuming because of the large number of joints and degrees of freedom (dof) involved.

The method that we present in this article is not a GUI but a Keyboard User Interface (which we shall refer to as KUI for simplicity). Although keyboard based, this interface
allows the user to reach the desired hand configuration and animation without prior knowledge of any 3D software and without selecting and applying transformations (i.e., translations and rotations) to the individual joints.

This interface differs from traditional input-display methods. Traditionally, keyboard input results in alphanumeric display. Hot keys are used for specific actions but keys are not used systematically to produce graphic output. For example, even in the simplest drawing program, such as the one embedded in Microsoft Word, the user cannot draw with the keyboard. The interface for drawing is based on mouse input as are most graphic interfaces.

In particular, for the configuration of 3D characters in modeling and animation, custom interfaces are often built to speed up the process of varying configuration. Such interfaces, see eg. [7], are also built on the basis of mouse input. A variant are motion capture input modes, in which case a motion capture suit with sensors is used to input character configuration data (see e.g. http://www.metamotion.com/hardware/motion-capture-hardware-gloves-Cybergloves.htm).

The reason why in such applications the keyboard input is not used is primarily because the keyboard input is a discrete type of input while the graphic output to be controlled is generally continuous. For example, in drawing a straight line the possible angles span a continuum of values from 0 to 360 degrees. If the possible values of the angles were restricted to multiple of, say, 18 degrees, it would be possible to use 20 hotkeys to specify the angle. Moreover, simplifications could be designed. At the opposite extreme, one single hot key could be enough if the user were willing to hit the hotkey up to 20 times to reach the desired angle. It is clear that some intermediate number of hotkeys, e.g. four, would require the user a maximum of 5 key strokes to reach the desired angle. This simple illustration contains the basic idea of the possibility of designing keyboard based interfaces for graphic output whenever discrete (quantized) values of the geometric parameters are acceptable.

This is not an artificial or uncommon situation. In fact, discretization is widely used. Practically all drawing programs, for example, have the ‘snap to grid’ option while producing a drawing. The grid forces a discretization of the plane in which the figure is drawn so that the resulting geometric parameters are discretized. Such situations are indeed useful not only to improve the speed but also the accuracy of the drawing. Similar advantages are offered by our method of discretizing the joint parameter values for the hand configuration so as to allow keyboard entry. Higher speed and accuracy of configuration can be achieved, as we discuss below.

2. Potential applications of our method

Our method can be applied to a variety of fields such as 2D illustration, technical/medical animation, signed communication, and character animation.
As mentioned earlier, in character animation it is very important to capture and clearly convey the expressiveness of the hands. Typically, to facilitate the animation process, the animator uses reactive animation or expressions to create a series of user-defined attributes which drive the rotations of the hand joints. Examples of these standard attributes are finger curl, finger spread, pinky cup, fist, etc. [9], [10]. While these attributes alleviate the animator of the tedious task of individually selecting and manipulating the hand joints, their creation is time consuming and requires software expertise. Usually a limited number (8-10) of custom hand configurations is produced for each character. In the majority of the cases, the user-defined attributes are used to bring the character’s hand into a configuration that is close to the desired one. The animator is still required to manually select and rotate the joints to tweak the hand pose.

Our method allows the user to reach any hand configuration with just a few keystrokes, no user-defined attributes are required. Because of the simplicity of the method, the user can easily create a large library of code-based hand configurations for each character. The codes, stored in a text file, can be easily loaded into the 3D scene and applied to a variety of characters when needed.

A 2D artist can quickly produce a large number of hand poses, apply them to different hand models (as explained below, our method can be used with any hand model rigged with a standard skeletal setup) and produce 2D images to be used in many applications such as technical illustration, multimedia and web content production, 2D animation, signed communication.

Figure 1 shows a 2D image of a code-based hand-tool configuration [11] captured from 2 different points of view along with the raw code (in black) and the compacted code (in red). Figure 2 shows 2D images of 8 basic hand configurations for ASL [12]. Table 1 contains the relative compacted codes.

Our method not only allows the user to quickly configure the hand, but also to animate it with high level of realism. Because of its ease, speed and accuracy, our method can be used to quickly produce complex technical animations such as the medical animation illustrated in Figure 6 [13] and the ASL sequence illustrated in figure 3 [12] representing the animation of the sign “travel”.

Figure 1. Example of code-based hand-tool configuration.
Figure 2. Example of code-based hand configurations for use is ASL (American Sign Language).

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bent</td>
<td>a2bcD4g5Hk5o5ps5TU2</td>
</tr>
<tr>
<td>Bent L</td>
<td>ABefi4j9k5m6n8o6pq7r8s6T2U2</td>
</tr>
<tr>
<td>Bent 5</td>
<td>ad2e6f2g3hi5j3k2lm4n3o3pq2r5s4u</td>
</tr>
<tr>
<td>Bent V</td>
<td>a4b4c3de4f4gi4j5km4n9o6q4r9s6T2u2</td>
</tr>
<tr>
<td>Curved</td>
<td>b2cD3g4Hk4o4ps4u2</td>
</tr>
<tr>
<td>Curved 3</td>
<td>a4de5f3gi5j4km5n9o5q5r9s5Tu2vw</td>
</tr>
<tr>
<td>Flattened O</td>
<td>a4bc2d2e4f5g3Hi2j6k3m2n4o4pqr4s4u2</td>
</tr>
<tr>
<td>Open</td>
<td>abcD4Hpt2</td>
</tr>
</tbody>
</table>

Table 1. “Compact” codes corresponding to 8 basic ASL configurations.

Figure 3. Example of keyframes in code-based ASL animation sequence illustrating the word “travel”.

3. How the method works

As mentioned, our method is based on the realization that the hand has 26 degrees of freedom which can be controlled by the 26 letters of the English alphabet. Via keyboard
input the hand can be positioned in space and manipulated to attain any configuration: by touching a letter key the user rotates the corresponding hand joint a pre-specified number of degrees around one of the three cardinal axes [14].

### 3.1 Description of the interface

The HCI of this method, being based on keyboard entry, is graphically very simple. It consists of only two windows: (1) the “Hand Configuration” window and (2) the “Bookmarks” window.

![Figure 4. The “Hand Configuration” window with code for the configuration at left.](image)

The “Hand Configuration” window (Figure 4) has three collapsible frame layouts: (1) the “Character Parts” frame; (2) the “Action Fields” frame; and (3) the “Action Buttons” frame.

The upper frame is used to select the character part. At this stage only hands are selectable and the right hand only is operational (checked box in Fig. 4).

The middle frame consists of two fields: the upper field echoes the hotkeys used to configure the hand (it can also be used to type in code in any form, raw or compacted, sorted or not); the lower field contains compacted code (that is code that is in the standard form as described below. See also figure 1, red fonts).

The third frame contains six buttons: (1) the upper-left button compacts the code (in raw or unsorted form) in the upper field and writes it in the lower field. (2) the upper-middle button executes the compacted code in the lower field (the hand will reconfigure itself accordingly and the reconfiguration is relative to the neutral position (see figure 4 on the left side). (3) The upper-right button executes ‘ASL code’ written in the lower field. ‘ASL code’ refers to fingerspelling hand configurations of ASL. To detect the ASL configurations we write the letters between zeros. Thus, for example, 0B0 corresponds to the configuration of the ASL letter B. If the lower field contains code of the form
0character0 it will interpret as ASL and the button execute ASL will configure the hand in the corresponding fingerspelling shape. (4) The lower-left and (5) lower-middle buttons simply clear the upper and lower fields respectively. (6) The lower right button opens the “Bookmarks” window.

The “Bookmarks” window (represented in Figure 5) is opened by the “Bookmarks” button in the “Hand Configuration” window. The window “Bookmarks” consists of a “File” menu, three buttons and an arbitrary number of text fields followed by a checkbox.

Each text field is used to write one hand configuration code, typically cutting and pasting from other files or the “Hand Configuration” window but also by directly reading a text file in which hand configuration codes have been saved. The role of the “File” menu items is related to this.

The first menu item (“Save bookmarks”, not visible in Figure 5 in which the File menu is not expanded) of the File menu saves all the hand codes written in all the fields of the window “Bookmarks” to a text file chosen by browsing. The second menu item (“Load bookmarks”, also not visible in Figure 5) reads hand codes written in a text file sequentially separated by blank spaces and loads them, one per text field, in the window “Bookmarks”.

The left button is used to create additional text fields with the corresponding checkbox. The middle button executes whatever hand code has the corresponding box checked. If more than one box is checked the hand configurations are ‘added’ from top to bottom field. ‘Adding’ two configurations means that the second code is executed starting from the configuration of the first code (instead of starting from the neutral configuration). This is particularly useful in correcting and/or refining hand configurations. The right button sets a keyframe for the hand in the configuration specified by the hand code with a box checked.

Typically, after refining the hand configurations created by hotkeys and recorded by the “Hand Configuration” window, the codes are written sequentially in the “Bookmarks”
window and keyframed individually at chosen times to produce the desired animation. (see Figure 6).

![Figure 6. Illustration of the process of setting keyframes using the “Bookmarks” window.](image)

Refining the times of the keyframes is thus very quick and simple. Also, inserting additional keyframes requires only to keyframe an additional hand code. Erasing a keyframe is accomplished by keyframing a blank field (which creates the neutral hand configuration) or repeating the previous frame. This is not exactly removing the keyframe but in many practical cases it accomplishes the required results.

### 3.2 Description of the coding

The skeleton of the hand consists of 16 movable joints: 15 finger movable joints (three per finger) and the wrist joint. The 16 joints have a total of 26 dof. The wrist joint has 3 translation dof and 3 rotation dof; each of the 15 finger joints has one rotational dof (pitch); the lower joint of each finger has an additional rotational dof (yaw).

These 26 dof can be controlled independently. Generally rotational motions do not commute so interchanging the order of two rotations results in different configurations. If this were the case, the keyboard input method would not be practically useful since the process of finding the correct configurations is done by successive approximations regardless of the order of the rotations. Fortunately it is possible to design a method for keeping the rotations independent of each other. This is based on using incremental Euler angles-- a feature that is available in current versions of Maya.

The rotations of the finger joints and the translation and rotation of the wrist joint are quantized at the desired resolution. In our case the finger joints pitch motion is quantized in steps of 10 degrees; the finger joint yaw motion is quantized in steps of 5 degrees; the wrist rotations are quantized in steps of 20 degrees and the wrist translations in steps of 1 cm. These values are used as a practical example but it is clear that the values of the quantization steps can be tailored to the needs of the range of tasks required. It is useful
to keep in mind that there is a proportionality between size of the quantization steps and speed of reconfiguring the hand.

In the process of reconfiguring the hand, we use alphabetical hot keys to move the hand into the desired configuration. The 26 letters of the alphabet correspond to the 26 dof of the hand. Capital letters are used for reversing the motions. As a hotkey is pressed, the corresponding dof is incremented or decremented (lower or upper case hotkey) in value by one quantization step. The hotkey letter is recorded by the program and appears in the upper text field of the “Hand Configuration” window for checking and monitoring one’s keyboard actions.

Different human operators generally will reach the same configuration with different keystrokes. For example, starting from a reference configuration one operator may reach the ‘victory’ configuration (see figure 4) with the keystrokes:

SssssrrssrrrrrRrrqqQqqqqToNNnnnooooonnnnnnppnnmccmmcdDDdbbaabaahL

another operator may reach the same configuration with any permutation of the same keystrokes and/or with additional self-canceling sequences such as, e.g., CCcc.

As we can see from this example, the code, as recorded from the typed keystrokes, is not practical for storing, transmitting and/or combining with codes of other configurations. The ‘compact code’ button changes the code to a compact form in which letters are sorted in alphabetical order, each letter being followed by a number indicating the number of repetitions of that letter keystroke. For the example above the compact code is:

a4b3c3DhLm4n9o6pq5r9s6T

This form of the code is much more legible and an operator can easily produce the corresponding hand configuration by typing in the alphanumeric keystrokes in the lower field of the same window and pressing the ‘execute code’ button. Again we remark that this is possible because the order of the rotations of the joint angles does not affect the final result.

The internal representation of the configuration code is a 26 component vector. For the modeler/animator it is convenient to have an alphanumeric representation of the hand configuration as described above. For computational purposes, however, it is convenient to represent each value of the 26 dof as a signed integer. Positive values correspond to the lower case letters and negative values to the upper case letters. Geometrically the integers are related to the joint rotations (and translation of the wrist) by the magnitude of the quantization steps chosen. For example, as we have mentioned, in our case all the finger joint pitch rotations have quantized rotations with 10 degree steps. The finger yaw rotations have steps of 5 degrees and the wrist rotation dof have steps of 20 degrees. The translations have steps of one unit of length. In the example above the alphanumeric code:
is internally represented as the vector:

\[ [4,3,-1,0,0,0,1,0,0,0,-1,4,9,6,1,5,9,6,-1,0,0,0,0,0,0] \]

4. Advantages of the “KUI” Method

4.1 Applicability

A variety of hand models can be freely downloaded from web sites or purchased from 3D graphics companies (i.e., www.viewpoint.com). Usually the model of the hand is a continuous polygonal mesh which can be imported into different 3D software packages. Once the model has been imported, the creation of the skeletal deformation system is carried out by the animator in the 3D software of choice. There are currently 2 standard skeletal setups that are commonly used for hand animation: setup 1 involves the use of a 24-joint skeleton [9], [10], (see figure 7 on the left); setup 2 involves the use of a 26-joint skeleton [15], [16], (see figure 7 on the right).

![Figure 7. Two standard skeletal setups for hand animation.](image)

Both setups include the 14 phalanges (14 movable joints) and the first and fifth metacarpal bones (2 joints). While setup 2 includes also the 2\(^{nd}\), 3\(^{rd}\), and 4\(^{th}\) metacarpal bones (3 joints), setup 1 connects the 2\(^{nd}\), 3\(^{rd}\) and 4\(^{th}\) metacarpal bones into 1 joint. Setup 2 uses a total of 5 bones (5 joints) for the thumb (2 carpal, 1\(^{st}\) metacarpal, 1\(^{st}\) proximal phalanx, 1\(^{st}\) distal phalanx) and 5 bones (5 joints) for the pinky (1 carpal, 5\(^{th}\) metacarpal, 5\(^{th}\) proximal phalanx, 5\(^{th}\) middle phalanx, 5\(^{th}\) distal phalanx). Setup 1 makes use of 4 bones (4 joints) for the thumb (1 carpal, 1\(^{st}\) metacarpal, 1\(^{st}\) proximal phalanx, 1\(^{st}\) distal phalanx) and 6 bones (6 joints) for the pinky (2 carpal, 5\(^{th}\) metacarpal, 5\(^{th}\) proximal phalanx, 5\(^{th}\) middle phalanx, 5\(^{th}\) distal phalanx).

The advantage of setup 1 lies in the presence of an extra 5\(^{th}\) intermetacarpal joint which allows a more realistic motion of the pinky-cupping of the pinky. The advantage of setup 2 lies in the presence of all the metacarpal bones and the extra 1\(^{st}\) intermetacarpal joint which allow an extremely realistic deformation of the top of the hand and the thumb.
Considered that in a real hand very little or no movement occurs at the intermetacarpal and carpal joints, we can think of these joints as non-movable and so eliminate the differences between the two skeletal setups. Even if the two setups present a different number of joints, because our method assigns a 0 dof to all the intermetacarpal and carpal joints, the total dof of both setups total 26 (see figure 8).

It is advisable to keep the non-movable joints as part of the skeletal setup even if they do not contribute to the motion of the hand. The function of these joints is primarily to facilitate the skinning process by creating a natural distribution of the skin weights thus allowing organic and realistic deformations during motion.

![Figure 8. The 26 letters of the alphabet mapped to the 26 dof of two standard skeletal setups.](image)

Given the above, our KUI method can be used to configure and animate any hand model that uses standard setups 1 or 2 as the skeletal deformation system (size, appearance and construction method—NURBS, Polygons, Subdivided Surfaces—of the hand are irrelevant).

### 4.2 Accuracy

In general the accuracy of the hand configuration is inversely proportional to the magnitude of the quantization steps chosen. It is worth noting, however, that the visual effect can tolerate relatively large quantization steps. Figure 9 shows, on the left, the Q and R manual alphabet hand shapes produced with our method (discreet rotation values) and, on the right, the same configurations produced with the selection and transformation tools of the 3D software (continuous rotation values). For this example all the finger joint pitch rotations have quantized rotations of 10 degree steps. The finger yaw rotations have steps of 5 degrees and the wrist rotation dof have steps of 20 degrees. With such rotation values we were able to achieve very accurate hand configurations (in figure 9 there is no noticeable difference between the configurations on the left and the ones on the right). Figure 10 shows another example of ASL hand configurations (numbers 0 to 9) [17] produced with the same quantized rotations. Table 2 contains the codes corresponding to each number configuration.
Such accuracy may not be surprising if we note, for example, that a quantization of the three wrist angles of 20 degrees results in 5832 possible orientations of the wrist and hence a significant visual discrimination requirement. We also note that this large quantization requires a maximum of only 9 key strokes for x, y and z to move to any desired orientation. (All this also assumes that the wrist is a spherical joint with no limits which in practice is not the case).

However, for hand configurations that require very careful positioning of the fingers to avoid obstacles and prevent collisions, such as fingers that need to fit in tight spaces (as in Figure 13), the magnitude of the quantization steps can be easily reduced to accommodate the complexity of the hand shape. It is clear how the magnitude of the quantization steps is inversely proportional to the amount of time required to reach a particular configuration. While small quantization steps allow a high level of precision, a large number of keystrokes is required to reach the desired configuration.

*Figure 9. Comparison between Q and R ASL hand shapes produced with the “KUI” method (right) and the traditional 3D software method (left).*

*Figure 10. Code-based ASL ten digits configurations.*
Configuration | Code
--- | ---
Zero | ac2e2f4g3i4j5k2lm3n4o2p2q2r2s2t2U4
One | a3b2c2D2i4j10k5m6n9o6pq6r8s6
Two | a3b3c3DhLm6n9o6pq6r8s6T
Three | hLm6n9o6pq6r8s6T
Four | a3b4c2D2eT
Five | Ah2PT3
Six | a3b5c3h4q9r9s4t3
Seven | a3b5c3h2m3n11o3p
Eight | a5b4c2dh2i4j10k4T2
Nine | a5b2ce7f8g2PT3
Ten * | A2D4e3f9g6H15j9k6m4n9o6pq5r9s6Tu4w3
      | A2D4e3f9g6H15j9k6m4n9o6pq5r9s6Tu6w3
      | A2D4e3f9g6H15j9k6m4n9o6pq5r9s6Tu2w3

* (since the handshape is animated, it requires 3 codes, one for each position)

Table 2. “Compact” codes corresponding to the ASL number (0-10) configurations.

As explained below, the accuracy and smoothness of the animation is not only inversely proportional to the magnitude of the quantization steps chosen, but also directly proportional to the number of hand configurations (hand codes) used in the sequence.

4.3 Code-based animation versus traditional computer animation methods

In traditional keyframe animation, to animate a hand gesture, the animator selects the appropriate joints, transforms them to attain a particular hand pose and sets a keyframe (to set a keyframe means to save the transformation values of the joints at a particular point in time). After a keyframe has been set, the animator manipulates the joints to reach a different pose and sets another keyframe (at a different point in time). The process is repeated until the desired animation is accomplished. Once the keyframes for the sequence have been defined, the 3D software calculates the in-between frames and the animator decides which interpolation (linear, spline, flat tangents, stepped tangents etc.) the software should use to calculate the intermediate transformation values of the joints between the keyframes [18]. All 3D programs allow the animator to edit the animation curves which are usually Bezier curves representing the relationship between time, expressed in frames (x axis), and transformation values (y axis). By editing the curves (scaling/rotating the tangents, adding or removing keyframes), the animator can tweak the animation with high level of precision.

With our method, the user enters a sequence of hand codes in the “Bookmarks” window and sets keyframes for each hand configuration. However, after recording the keyframes, the user does not have direct access to the curve tangents (set to flat). We could have included this as an option in the “Bookmarks” window but it would have defeated the object of the simplicity of the method. Being unable to access the animation curves might seem a severe limitation to the ability to refine the animation but actually the user can step through the animation and observe the hand. If the hand is not in the desired configuration at the observed frame, she can simply code in and keyframe that configuration.
The method involves the following steps: (1) in the lower field of the “Hand Configuration” window the user enters the code of the keyframe closest and preceding the frame to change; (2) after returning the hand to neutral position, she presses ‘execute code’; (3) the user clears the input (upper) field of the “Hand Configuration” window; (4) using the hotkeys she brings the hand in the desired configuration; (5) she presses “compact code” -- this will produce (in the lower field) the code for the desired configuration--; (6) she uses this code in the “Bookmarks” window to set an additional keyframe.

By keyframing additional hand codes, the user can refine the animation with high level of precision. Figure 11 shows the 5 code-based keyframes used to produce the animation illustrating the right hand configuration and motion in infant CPR (chest compression phase) [13]. Figure 12 shows the 3 standard keyframes used to produce the same sequence.

In order to achieve the level of smoothness and precision as in the sequence in figure 12 and avoid the intersection of the middle finger with the infant’s chest, we were required to add 2 extra keyframes (frames 17 and 20, figure 11). The two additional hand codes cause the y translation of the wrist joint to occur before the pitch rotation hence avoiding the intersection of finger and body. The same problem could not have been solved by simply decreasing the magnitude of the quantized rotations and translations of the wrist joint. In sequence 2 (figure 12) no additional keyframes were required since the
intersection was prevented by editing the curve tangents of the wrist joint rotation and translation.

Figure 13. Example of code-based hand configuration and animation in low clearance and occluded areas.

The sequence in figure 13 [19] is another example of how the animation can be refined and precisely controlled by adding keyframes and/or adjusting the resolution of the quantization steps. Figure 13 illustrates a case of configuration and animation in low clearance and occluded areas. Such areas present several challenges since they require precise positioning to avoid obstacles and careful animation to prevent collisions. In order to fit the fingers in such tight spaces and avoid intersection with the bowling ball, 3 additional keyframes (frames 3, 6 and 12, figure 13) were used and the quantized rotations for the wrist joint were decreased to 10 degree steps.

4.4 Storage and transmission of poses and animation

Poses, as we have seen, are represented as 26 dimensional vectors of integers. We use the term vector in the mathematical sense, not as defined in MEL (Maya Embedded Language) where ‘vector’ is a term reserved for mathematical 3 dimensional vector. The MEL language would describe what we call a 26 dimensional vector as an ‘array of size 26’.

Since a pose is a set of 26 integers it is straightforward to store it. For convenience, the window “Bookmarks” contains the menu “File” with two menu items: “Save bookmarks” and “Load bookmarks” which, as described above, allow storage and retrieval of poses from simple text files. Transmission of poses is then reduced to transmitting sets of 26 integers (for example in text files but also directly). A comparison can be made with exporting and importing clips of poses. This process requires typically at least twenty times more memory.

For animations, our method allows the encoding of animations as keyframes described as pairs composed of an integer for the time and the 26 dimensional vector describing the pose at that time. Thus sets of 27 integers form the description of an animation. However this process is not, at the moment, independent of the Maya interface since it relies on the tweening done by Maya between keyframes. The method can be extended and
generalized to be applicable across different animation packages; it will be the subject of a future publication.

5. Conclusions

In this paper we have developed a new method of configuring and animating hands. The method utilizes the user’s typing skills to control the configuration and animation of a computer generated three-dimensional hand. The idea is based on the realization that a hand gesture path requires the same number (26) of parameters as the letters of the English alphabet, hence a one-to-one relationship between hand dof and letter keys can be established.

The method presents several advantages. (1) Flexibility: it can be applied to any three dimensional hand model with a standard skeletal setup (26 dof); (2) Accuracy: although the method uses a discretized representation of hand position and orientation, the resolution of the quantization can be easily adjusted to configure the hand with high precision; (3) Simplicity of code-based animation: the method allows the user to quickly and easily produce complex sequences without prior knowledge of the software; it also enables the user to tweak the animation by simply adding code-based keyframes (no parameter curve editing is required); (4) Ease of Storage and transmission: configurations and animation can be stored in text files of minimum size, exported cross platform or transmitted via internet.

The method uses a CHI entirely based on keyboard entry. The interface is graphically very simple and contains only two windows: the “Hand configuration” window and the “Bookmarks” window. The first window is used to select the character parts, record the keystrokes and reduce them to an alphanumeric “compact code”. The desired hand configuration can be reached starting from the hand neutral position or from an ASL manual alphabet handshape. The second window is used to write the hand configuration codes necessary to produce the animation and to set keyframes. The codes can be cut and pasted, read from a text file and also saved to a text file for later retrieval.

Traditional hand animation methods are tedious, time-consuming and require 3D software expertise. We have developed a quick and easy keyboard-based method of configuring and animating hands that could find extensive application in many fields such as: (1) signed communication; (2) technical/medical animation; (3) technical illustration; (4) art education; (5) character animation.

There are several limitations to the method presented here. First the method does not allow arbitrary reconfigurations of the hand. This is intrinsic to any discretized representation of position and orientation. Discretization is a matter of degree and clearly there is discretization in any digital representation but what we are concerned with here is a significant degree of discretization, perhaps better referred to as quantization of the motion. The size of the quanta of motion can be chosen and this choice results in a basic tradeoff. As discussed in the paper, the smaller the step the higher the accuracy of reconfiguration, and the lower the speed of achieving it. Once the position has been
reached and the code for the configuration has been determined there is no further cost in higher precision, i.e. in using smaller quanta of motion.

Thus an improvement to our method can be carried out by tailoring the quantization level and using it in the standard fashion of progressive refinements; that is, setting a low resolution (large steps) initially and, after reaching the proximity of the desired configuration, setting a high resolution (small steps) for fine tuning the position. Although conceptually straightforward and practically conveniently usable, we have not implemented this more complete approach yet but we have left it for a future development.

A second limitation at the moment is the restriction to one particular skeleton structure and geometry. As discussed previously the skeleton structure is common to a large group of commercially available hand models, and extendable to a second large class of skeleton structures. However, this extension is also not implemented here and it is left to future developments.

Similarly we have left for future development the portability of the method across modeling/animation platform. So far our application has been limited to Maya 4.5. As we have indicated there are several cross platform extensions that could be implemented. The simplest is sharing hand configuration codes (poses and animations) across platforms. This would require an implementation of a similar interface (corresponding to the functions of the two windows in this work) in the new platform. No additional transfer software is required since the codes are in simple text.

Finally, a further current limitation of the method is the geometrical parameters of a given skeleton structure. The configuration depends on the length of the links (bones) of the skeleton. These can be rescaled and the method applied to skeletons with similar structures but different bone lengths. However this rescaling capability is not yet implemented.

Apart from the development of these current limitations, future applications of the method can conceivably include client-server operation via the internet. Such internet implementations are a very promising avenue for future research.

References


Nicoletta Adamo -Villani

Nicoletta Adamo-Villani is Assistant Professor in the Department of Computer Graphics Technology at Purdue University. She received her Master in Architecture from the University of Florence, Italy and she is a Certified Animator and Instructor for Alias Wavefront.

Prior to joining Purdue, Prof. Adamo-Villani has taught courses in the fields of 3D Modeling, 2D and 3D Animation, Multimedia and Web Design at various institutions including: the University of California, Riverside; DeVry Institute of Technology, Pomona, CA; Augustana College, Rock Island, IL and Marycrest International University, Davenport, IA.

She has produced graphics and animations (aired on national television) for companies such as Marvel Comics, Genesis Designs and Endocyte Inc. and she has received public recognition for the design of one of her award-winning web sites. Her special area of expertise is in Character Animation and Character Design. Her current research interest is in the application of 3D Animation to teaching communicative gestures, e.g., American Sign Language, fine manipulative skills, and dynamic manipulative tasks. She is also interested in the application of 3D Animation to scientific visualization.

Nicoletta Adamo-Villani
Department of Computer Graphics Technology
Purdue University
Knoy Hall of Technology
401 N. Grant St.
West Lafayette, IN 47907-2021

Tel: 765-496-1297
Fax: 765-494-9267
E-mail:nvillani@tech.purdue.edu

Gerardo Beni

Dr. Beni received the Ph.D. in Theoretical Physics at the University of California, Los Angeles in 1974. He was a Distinguished Member of the Technical Staff at AT&T Bell Laboratories where he conducted research in solid state physics for 10 years. From 1984 to 1991, as a professor of Electrical and Computer Engineering at the University of California, Santa Barbara, he co-founded and co-directed the NSF Center for Robotic Systems in Microelectronics. Dr. Beni has over 150 publications and 15 patents. He is the co-
founder and co-editor of the *Journal of Robotic Systems*. Dr. Beni became a fellow of the American Physical Society in 1982, and is listed in Who’s Who in America and Who’s who in the World. He has been a Professor of Electrical Engineering at the University of California, Riverside since 1991 where he directed the Distributed Robotics Systems Lab and the MultiMedia Lab&Studio. He is also a certified instructor in computer animation for AliasWavefront. He was elected fellow of the American Association for the Advancement of Science (AAAS) in 1999.

Gerardo Beni  
Department of Electrical Engineering  
University of California Riverside  
900 University Ave  
Riverside, CA 92521  
E-mail: beni@ee.ucr.edu
Hand gesture recognition (HGR) is an important research topic because some situations require silent communication with sign languages. Computational HGR systems assist silent communication, and help people learn a sign language. In this thesis, a novel method for contact-less HGR using Microsoft Kinect for Xbox is described, and a real-time HCR system is implemented with Microsoft Visual Studio 2010. Previous researches on computer vision and hand detection have established solid groundwork for gesture recognition. However, only a few Kinect-based systems were developed for HGR, and only a few gestures were recognized. Belated work will be reviewed in Chapter 2. Another traditional approach for hand-tracking is known as glove-based technologies.