Efficient Hand Gesture Rendering and Decoding Using A Simple Gesture Library

Jason A. Smith and Lijun Yin

Department of Computer Science, State University of New York at Binghamton
Binghamton, NY, 13902

ABSTRACT

Recent work in hand gesture rendering and decoding has treated the two fields as separate and distinct. As the work of rendering evolves, it emphasizes exact movement replication, including more muscle and skeletal parameterization. The work in gesture decoding is largely centered on trained systems, which require large amounts of time in front of a camera rendering a gesture in order to decode movement. This paper presents a new scheme which more tightly couples the gesture rendering and decoding processes. While this scheme is simpler than existing techniques, the rendering remains natural looking, and decoding a new gesture does not require extensive training.

1. INTRODUCTION

Interest in the field of gesture rendering and decoding has grown in the past several years with the advent of powerful graphics processors, and camera based hands-off user interfaces – interfaces that use cameras to decode gestures and movement. Work done in gesture rendering has been largely un-related to work done in gesture decoding, and vice versa. These two fields have taken dissimilar approaches to a similar problem.

As the work in gesture rendering evolves, it produces more realistic motion, modeling, and animation[1][2]. This increased realism comes at the expense of increased complexity. Describing muscle and skeletal motion used to encode a simple gesture now contains several intermediate steps to define each minutia of movement.

As the work in gesture decoding evolves, it has moved from basic computer peripheral input decoding[3][4], to glove based decoding[5][6], to visual decoding[7][8]. Today's systems lean more heavily on glove based systems or purely visual systems that require extensive and specific training [5][6][7][8]. These visual systems attempt to decode and describe motion. This work requires hours of manually training a complex system for each gesture added to the list of known gestures, although current work is making strides in reducing and eliminating extensive training[9].

As the two bodies of work continue to advance, there is no apparent middle ground. Additionally, not all rendering schemes will require a complex and vividly realistic motion model. Video games or simple motion capture systems do not need excessive realism, and would benefit from a reduced data set used to describe motion. Decoding methods should be able to leverage work done in motion rendering. This paper presents a scheme that more closely couples the work of gesture rendering and decoding.

The scheme presented here is a joint angle based system that can be used to render simple hand gestures. The system is adaptable for most skeletal based motion, but this paper will focus on hand based gestures. A simple gesture is defined as having a predetermined end point, one that does not require crossing finger patterns, and one that can be completed in a single motion – that is, a finger that begins moving by curling or closing into the palm, will continue to curl until the end of the gesture. Examples of a simple gesture include the gestures "one" "three" and "horns" as shown in Figure 1.

In this system, both gesture rendering and decoding are driven from this end-position joint angle model. A joint angle structure defines the angle of rotation for each finger, as well as the final joint angle for each joint in a finger. A library of simple gestures is created, each entry consisting of one joint angle structure.

Once this library is constructed, a gesture is rendered by moving between any two of these joint-angle structures. Any two simple gestures can be moved between in any order.

Decoding the gesture requires taking a snapshot of the overall joint positions once a gesture has been rendered. Determining when a gesture has been rendered is discussed later in this paper, as well as the algorithm to determine which gesture was rendered.

It is not the goal of this paper to provide a complete framework to render and decode complex gestures. The goal of this scheme is to more tightly couple the work of rendering and decoding gestures; provide simple, yet realistic hand gesture motions based on final joint positions; and to provide a simple angle based decoding scheme for rendered gestures.

2. GESTURE DEFINITION

There are two methods used to define a gesture. A gesture can either be described as the motion of fingers, or as an end position of the fingers. Most common gestures are ones that can be defined by their end position. For example, the "number one," "OK," and "Peace," are gestures that are described by the position of their fingers. An example of a motion based gesture would be using the index finger to "trace" a letter or figure in the air, such as moving the tip of the index finger in a circular motion to trace a "zero" or "O."

The methods presented in this paper work on gestures that are described by finger positions, not motion. The methods described also are intended to work off "simple" gestures – gestures that do not require awkward or unnatural motion. This is motion that the finger cannot support physically by itself. The act of "crossing your fingers" is such an example. The index finger is required to restrain the middle finger from returning to its natural position. This crossing is considered awkward for the sake of this paper. The extension of the middle finger across the index finger, and the use of a restraining index finger make this motion complex.

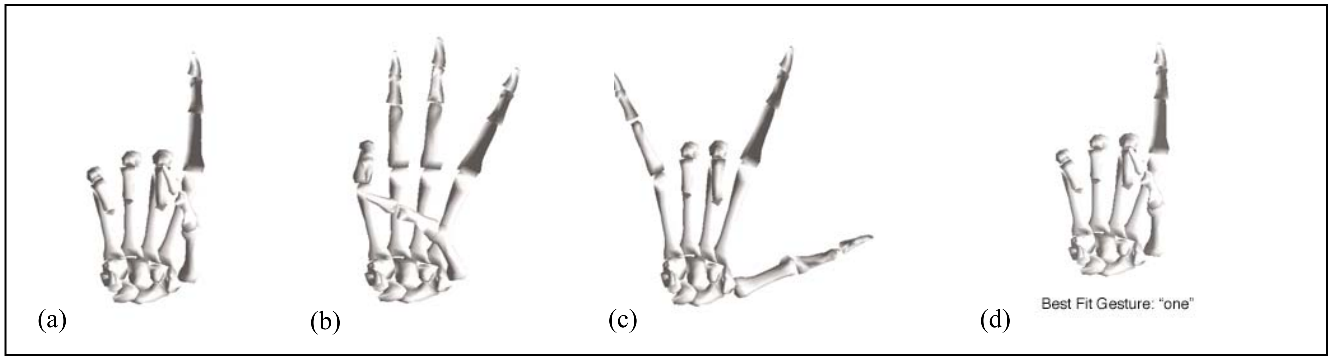

Figure 1 Simple gestures "one" (a) "three" (b) and "horns." (c), and the gesture decoding algorithm detecting the gesture "one" (d)

There are four parameters that must be defined per finger. They are the two joints above the base joint of a finger, the tip of the finger, and the angle of rotation of the finger. These parameters are shown in Figure 2. An entry in the gesture library is structured as :

```
gesture {
        float [0][0]        // Thumb joint 0 position
        float [0][1]        // Thumb joint 1 position
        float [0][2]        // Thumb joint 2 position
        float thumb_rot     // Thumb angle of rotation
        float [1][0-2]      // Index joint positions
        float index_rot     // Index angle of rotation
        …
        float [4][0-2]      // Pinky joint positions
        float pinky_rot     // Pinky angle of rotation
}
```

By defining each value, we define a specific gesture. The gesture library is then an array of gestures. The library that was built contains gestures for the numbers one through 5, the peace sign - extending the index and middle fingers with 35 degrees separation, the letter L – with the index finger and the thumb fully extended with 90 degrees separation, and several other simple gestures.

3. RENDERING MOTION

Once two gestures have been defined, the end positions are fed into the motion model. The motion model determines how the skeleton must move between any two gestures. Each joint has a starting and end position, but the joint cannot simply move to the new position through simple methods like standard linear interpolation. There are a series of requirements imposed on skeletal motion such as, the skeleton cannot change shape while in motion, the motion of each subsequent bone and joint is dependent on the motion and position of the preceding joint, the motion of all joints and bones must be coordinated so that all bones and joints arrive at their final position at the same time.

As an example, the index finger in a "number one" position will be curled into a "number zero" position, as shown in Figure 2. In this example, the index finger is curled about the x-axis. The starting angles of each joint (in this paper, the tip of a finger is also referred to as a joint for simplicity of presentation) *A, B* and *C* are $\pi/2$. The final angle for joint **A** is 0, *B* is $-\pi/2$, and *C* is $-(3\pi)/2$. In order for all three joints to reach their final position together, the motion of each joint must be scaled. Joint *C* has to move three times farther than joint A, so it will move 3 times faster than joint *A*. Joint B has to move twice as far as joint *A*, so it will move twice as fast.

More specifically, a gesture is rendered in one "gesture frame" which is composed of 's' smaller "minor frames." That is, the motion of rendering a gesture is broken into 's' smaller movements. The angle through which a joint will move through for each minor frame ($\Delta\angle$) is then:

$$\Delta\angle = (\angle_{final} - \angle_{starting}) / s$$

Where $\angle_{final}$ is the final joint angle, and $\angle_{starting}$ is the starting joint angle. These values are obtained from the gesture library, using the previous and current gesture entries. Likewise, the angle of rotation is computed for each finger :

$$\Delta\partial = (\partial_{final} - \partial_{starting}) / s$$

Where $\Delta\partial$ is the delta angle of rotation, ∂_{final} is the final angle of rotation, and $\partial_{starting}$ is the starting angle of rotation. These values are obtained from the gesture library, using the previous and current gesture entries.

We can then apply standard transformation formulas to rotate a point in space about an arbitrary axis, using ($\Delta\partial + \partial_{previous}$) as the arbitrary axis and $\Delta\angle$ as the distance to rotate that point. Applying these transformations to each joint each minor frame, the hand gesture will be completely rendered, in a smooth manner - each joint reaching the end position at the same time, in s minor frames.

The basic motion algorithm for any given joint is then:

```
$\Delta\angle = ( \angle_{final} - \angle_{starting} ) / s$
$\Delta\partial = ( \partial_{final} - \partial_{starting} ) / s$
for (gesture_frame = 0; gesture_frame < s; gesture_frame++)
{

    move joint to origin.
    $\angle_{current}$ += $\Delta\angle$
    $\partial_{current}$ += $\Delta\partial$


   rotate joint "$\angle_{current}$ degrees" about "angle $\partial_{current}$"
   move joint off origin
}
```

This simple example is the basis for all joint motion in this system. A finger will move from a beginning angle to a final angle

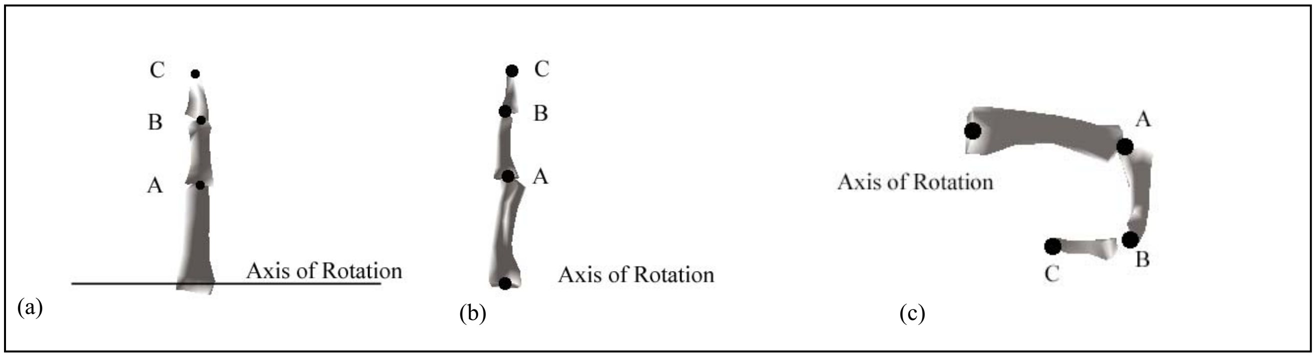

Figure 2: The index finger as seen from the front extended (a), side extended (b), and side curled (c), showing the axis of rotation, and the joints A, B and C with their relationship to each other.

about a common axis of rotation which runs through the preceding joint.

Similarly, a starting and ending axis of rotation is given for each finger. The finger will move to its new axis of rotation in the same amount of time as the individual joints need to move to their locations. This allows for natural appearing motion - having a gesture rendered with one continuous and smooth motion.

4. GESTURE RECOGNITION

In order to decode a gesture, the system must first recognize a gesture that has been rendered. There are two simple ways to determine if a gesture has been rendered – either a finger changes its direction, or a finger stops moving. As an example, the gesture "one" is our starting point. When we start to render the "two" gesture, the middle finger begins to move. The gesture "two" is complete when the middle finger is fully extended and stops moving.

Alternately, rendering the gestures "one, two, one" in sequence without pauses is detectable by detecting direction changes in a given finger. Starting with gesture one, and moving to two, the middle finger becomes fully extended. Then to render the gesture "one" without a pause, the middle finger begins to curl. The system detects a direction change in the middle finger from "opening" to "closing" and realizes a gesture that has been rendered.

Once the system realizes a gesture has been rendered, a snapshot of joint positions is taken.

Once a gesture snapshot has been taken, the system runs through a "best fit" algorithm to determine which gesture has been recorded. A library of gestures is built up, similar to the gesture library used to render gestures above. This library only contains required joint positions, and has a counter for the number of essential fingers for a given gesture. The system runs through the library, and matches as many joints angles as it can to find a best fit. The library is organized according to the number of essential fingers. Gestures that require only 1 finger are entered in the library first, then gestures that require 2 fingers, etc. Once n fingers from the snapshot have been matched, the algorithm will jump to the position in the library containing n+1 essential fingers until the gesture library has been exhausted. Although this algorithm has a worst case linear runtime, the average runtime is not linear since the entire library is not normally examined. In the following, we will use the "horns" gesture to illustrate this concept.

The algorithm first tries to match one finger of the gesture being rendered to the gestures in the gesture library. If 1 finger can be matched, the first guess is recorded. The algorithm would skip over the remainder of the "1 finger" gestures since a best guess for 1 essential finger has been made. The algorithm would then attempt to match 2 fingers, and continue on this way until the library is exhausted, or 5 fingers have been matched. The last recorded gesture is returned as the best guess.

Checking a single finger, the equation below is used, where FIT is a Boolean indicating if the finger has been matched, $\angle A^n$ is the angle of the *A* joint of the gesture in the gesture library currently being matched against, $\angle A^c$ is the angle of joint *A* that is currently being rendered, T is the tolerance applied to this gesture. If joints *A, B,* and *C,* and angle of rotation R, are within tolerance for the current gesture for the given finger, that finger is matched.

$$\begin{aligned}\text{FIT} = (\,|\angle A^n - \angle A^c| < T) \\ \&\ (|\angle B^n - \angle B^c| < T) \\ \&\ (|\angle C^n - \angle C^c| < T) \\ \&\ (|\angle R^n - \angle R^c| < T)\end{aligned}$$

The algorithm becomes

```
0: n = 1, best_guess = NULL
1: IF a gesture has been rendered, compare the snapshot against the first record for n essential fingers
2: if FIT is true, record best guess. Increment n and go to step 1.
3: if FIT is false, move to next gesture with n essential fingers
```

The algorithm exits when the library is exhausted, and the best guess is returned.

5. EXPERIMENTS AND ANALYSIS

The gesture library was built up using snapshots of a human hand posing for various gestures. Joint angles were calculated and entered into the gesture library. The motion system randomly moves between any of these gestures in a natural looking fashion.

The rendering system uses a set of static hand bone models. The individual bones are mapped to the base position of each corresponding joint, and use the output of the motion model to determine rotation.

A separate gesture library was created for the purposes of gesture decoding. This library only identifies essential fingers and bone angles, and is purposefully smaller than the encoding library.

The only hook between the motion layer and the decoding layer is a call from the decoding layer to get the current joint positions. The decoding layer has no knowledge of the encoding library, the gesture being rendered, or when a gesture has been completed. The decoding layer compares current and previous joint positions to determine direction of motion for each joint, and determines when that direction changes. Once direction for any joint changes, a snap shot is taken, and the decoding algorithm attempts to decode the gesture. The graphic in Figure 1-d shows the decoding system identifying the "one" gesture. The output of the decoding algorithm is shown as text below the figure being rendered. The decoding library is still in its infancy, and at the time of this paper contains approximately 10 gestures in total. A gesture has never been misidentified. Additional graphics and a video clip of the rendering and decoding system randomly counting and identifying gestures one through five can be seen on the project website.[1]

Gestures were added to the libraries in minutes, the gestures were rendered in real time, and the gestures were accurately decoded within the space of a few gesture frames – a matter of milliseconds.

The gesture library was built with variations on several gestures. Because there is no absolute way to render the gesture "one" several variations were given with varying joint angles for each finger. The decoding system was always capable of correctly identifying a gesture. The rendering library had several gestures that were not in the decoding library. In these cases, the decoding system decoded these as "unknown gestures" or returned a best guess which matched as many fingers as possible. So while the decoding system was not, in the strictest terms, correct in identifying the given gesture, it was correct in that it most closely matched the gestures present in its library to the one being rendered.

The gesture rendering system is simple. This system allows for the easy rendering of simple gestures, and allows realistic looking motion between any two simple gestures. Although the motion appears natural and realistic, it is not 100% anatomically correct.

It is not meant to be a perfect replication of human motion. It does, however, allow for natural looking motion with minimal data. The trade off is a gain in simplicity for a loss in absolute realism.

The simplicity of the decoding system allows for quick addition of gestures, in comparison to trained systems. Adding a gesture can take minutes, opposed to hours. By decoding gestures solely based on joint angle, there is no additional work to decode gestures on other hands, as there is on trained systems that must account for an individual's hand position and finger length.

6. CONCLUSION AND FUTURE WORK

The goal of this work was to simplify and couple the gesture rendering and decoding processes. In that regard, the program is a success – it allows for similar libraries to be used in the rendering and decoding processes. The system also allows for natural looking motion between any two simple gestures using a relatively small data set. The decoding system takes extensive training out of the equation, and allows for quick addition of gestures to the library. Both the rendering and decoding system work with any arbitrarily sized skeleton. This allows the motion system to couple to any skeletal models without extensive rework. This also allows the decoding system to work for any given hand – not having to be re-trained for different sized hands. While more work needs to be done in this system to further enhance the natural appearing motion, and to allow for more complex gestures, it successfully found middle ground between gesture rendering and decoding, and simplified both processes.

For the decoding system to be truly worthwhile, it must be taken out of the computer simulation. Adding a video capture front end to capture the joint position rendered by a live person is the next step in the decoding work. This translation from video to gesture recognition is the next major focus of this work. Currently, techniques are available which process live video feed and can find and track human features. The most readily available and reliable technique is used to track the position of the human eye – finding the eyes on the face, and tracking their motion. It is believed that this video processing technique can be applied to tracking joint position. Determining joint position using this technique will quickly make the existing gesture decoding system applicable.

[1] http://www.cs.binghamton.edu/~lijun/decode_demo.mov

REFERENCES

[1] Horace Ip, Sam C.S. Chan, Maria S.W. Lam, "Hand Gesture Animation from Static Postures Using an Anatomy-Based Model," cgi, p. 29, Computer Graphics International 2000 (CGI'00), 2000.

[2] Irene Albrecht , Jörg Haber , Hans-Peter Seidel, "Construction and animation of anatomically based human hand models," Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2003

[3] K. Mardia, N. Ghali, T. Hainsworth, M. Howes, and N. Sheehy. "Techniques for online gesture recognition on workstations." Image and Vision Computing, 11(5):283-294, June 1993.

[4] Dean Rubine. "Specifying gestures by example." Computer Graphics, 25(4):329-337, July 1991.

[5] Kouichi Murakami and Hitomi Taguchi. "Gesture recognition using recurrent neural networks." Journal of the ACM, 1(1):237-242, January 1991.

[6] Mohammed Waleed Kadous. "Machine Recognition of Auslan Signs Using Power Gloves: Towards Large-Lexicon Recognition of Sign Languages." *Workshop on the Integration of Gestures in Language and Speech, Wilmington Delaware*, 1996.

[7] Pattie Maes, Trevor Darrell, Bruce Blumberg, and Alex Pentland. "The Alive System: Full-body interaction with autonomous agents." In Computer Animation '95 Conference, IEEE Press, Geneva, Switzerland, April 1995.

[8] Thad Starner and Alex Pentland. "Visual recognition of American Sigh Language using Hidden Markov Models." IEEE International Symposium on Computer Vision, November 1995.

[9]Mathias Kölsch and Matthew Turk. Robust Hand Detection. In Proc. IEEE Intl. Conference on Automatic Face and Gesture Recognition, May 2004.