Lexical Gesture Interface

Zhenyao Mo, Ulrich Neumann
CGIT Lab, University of Southern California
{zmo, uneumann}@graphics.usc.edu

Abstract

Gesture interfaces have long been pursued in the context of portable computing and immersive environments. However, such interfaces have been difficult to realize, in part due to a lack of frameworks for their design and implementation. This paper presents a framework for automatically producing a gesture interface based on a simple interface description. Rather than defining hand positions in a low-level high-dimensional joint angle space, we describe and recognize gestures in a "lexical" space, in which each hand pose is decomposed into elements in a finger-pose alphabet. The alphabet and underlying rules are defined as a gesture notation system called GeLex. By implementing a generic hand pose recognition algorithm, and a mechanism to adapt it to a specific application based on a general interface description, developing a gesture interface becomes straightforward.

1. Introduction

While traditional input devices (keyboards, mice, joysticks, etc.) are still widely used in virtual environments and mobile applications, they are abstract and require physical contact with devices. These are barriers for interactions in virtual environments and mobile settings, where gestures have been recognized and pursued as a natural and effective mechanism for human computer interaction [1]; however the difficulty in creating gesture interfaces impedes their further development and adoption.

One of the main challenges is the gesture recognition problem. Digital gloves or marker-based methods provide feasible solutions, but wearing extra equipment on the hands lessens the natural and general character of gesture interfaces. Vision-based methods are non-invasive; however, low computational-efficiency and lack of robustness has impeded the adoption of vision methods for practical applications up to now. We employ a new vision sensor technology that does leads to robust vision performance.

Figure 1: two pointing poses

However the sensing and recognition problems are solved, the next challenge is to design an interface satisfying specific application requirements. This is the focus of our work and that domain includes the following:

1. How to define and express the desired gesture set for a system;
2. How to define the constraints of the interface, i.e., are both poses in Figure 1 treated as a pointing gesture, or is just the left pose defined as pointing?
3. How to enable users to configure a gesture interface to suit their own preference. For example, some people prefer to navigate in a 3D world using one finger, and others prefer to use a whole hand. User preferences can vary widely and it is impractical to anticipate all possible alternatives.

While most prior gesture-related research concentrates on gesture recognition, the design problem is widely ignored. However, we believe the design problem is also a challenge and deserves more attention.

Traditional gesture interface designs are directly integrated into a specific application, as shown in Figure 2 (a). In this approach, developing the gesture interface is both technically challenging and time consuming; in addition, a user must use the interface exactly as it is designed.

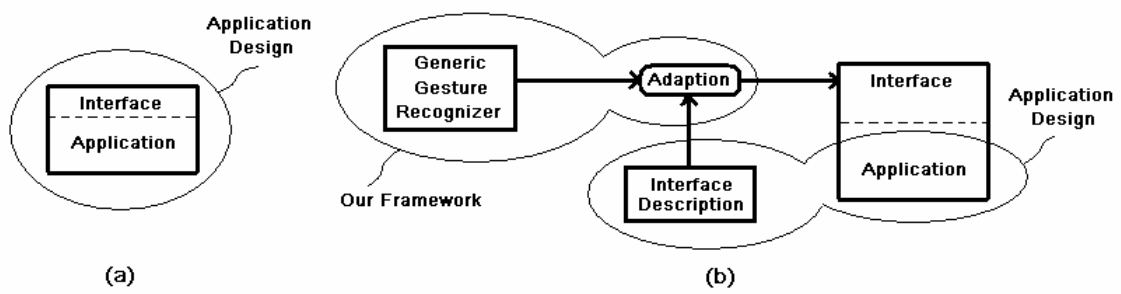

Figure 2: (a) is the current gesture interface design framework: the gesture interface is hard-wired into application system; (b) illustrate our framework: designers specify a description of the desired gesture interface, and our framework automatically configures a generic gesture recognizer to the description, producing the desired specific interface.

This paper presents a new design framework, which models the creation of a gesture interface as an interface description that automatically configures the implementation. Designers need only to provide the interface description, and our framework automatically produces the desired interface.

After reviewing related work, section 3 describes our framework and its components; section 4 illustrates an example of a gesture-driven application: the "Scissor-Rock-Paper" game, which is built with the framework; finally, conclusions and future work are discussed in section 5.

2. Related Work

Gestures, in their most general sense, are composed of movements, such as a hand-wave or circular motion, and poses such as the finger configurations shown in Figure 1. Our focus is on hand poses, and specifically on 3D pose representations. We survey vision methods since these are non-invasive and represent the majority of relevant work. Prior 3D hand pose estimation techniques fall into two categories: appearance-based and model-based. A review of early work in gesture recognition is provided in [2].

Model-based methods match a 3D hand model to features extracted from input images. In [3], a 27 degree-of-freedom (DOF) hand-model is tracked in real-time using two cameras. In [4], a model is built from truncated quadrics, and techniques from projective geometry are applied. Visual feature sets can be ambiguous, and the work in [5], [6], and [7] obtain correct poses by integrating constraints and statistical tracking methods.

Appearance-based methods are appealing for recognizing a small set of gestures. However, for larger gesture sets, such methods lack robustness and require extensive training data. In [8], the training data problem is addressed by combining a few labeled images with a large set of unlabeled images.

"Estimation by Synthesis" is a combination of model-based and appearance-based methods. 3D graphical hand models are controlled to create images, which are then compared with input images to estimate 3D poses. These methods often need to produce a large number of generated images to account for the variety of possible hand poses and views of them. In [9] an adjacency map is used to reduce the search space, achieving real-time performance with a cluster of six personal computers. In [10], a hierarchical retrieval based on combined measures is used to reduce the processing time for one frame to a few seconds.

3. Gesture Interface Design: a Framework

We observe that although gesture space is very high-dimensional, the underlying discernable finger poses are relatively limited. We also observe that gestures are compared and recognized more qualitatively than quantitatively; i.e., gesture space can be discretized without sacrificing recognition performance.

Based on these observations, we decompose a hand pose into individual finger poses, each represented by an alphabet including elements such as "bend" and "cross". Thus, gestures can be described and recognized in this lexical space instead of angle space.

As shown in Figure 2 (b), our framework has three components: a generic gesture recognition algorithm, a gesture notation system, and a mechanism to adapt the generic gesture recognizer to a specified interface description.

Any of the gesture recognition algorithms surveyed in section 2 would serve as a generic recognizer in our framework, since they all obtain 3D hand poses. This function is not our main research focus; yet we implemented our own 3D hand-pose estimation

algorithm using a new depth-image camera, due to its robustness (see section 3.1).

Application designers provide the interface description that contains a set of desired gestures using a gesture notation system called GeLex (see section 3.2 and Appendix A).

The hand poses obtained from the generic recognizer are further decomposed into finger poses, and a specified gesture is recognized if finger poses match the design descriptions (see section 3.3).

3.1. Generic Hand Pose Recognition

This section describes a generic hand pose recognition algorithm that uses a new laser-illuminated camera made by Canesta, Model DP-205. By "generic" we mean that our algorithm is not trained to recognize any specific set of poses, or views of poses, but hand poses in general.

The camera outputs depth images at a resolution of 64x64 pixels, at video rates. We observe it to be insensitive to ambient illumination and quite robust in many conditions encountered in our experiments. In current implementation, however, we constrain that a user's palm faces the camera (rotation with 30 degrees is fine). This is because with low-resolution inputs, side views may not provide not enough information for identifying hand poses.

Figure 3: a hand model

Hands can be modeled at different levels of detail, according to the application needs. Our system uses a representation of finger poses, thus, our model is a right hand with a palm and 14 phalanxes (each modeled as a cylinder), as shown in Figure 3. Since we only model the structure of a hand and not the actual sizes, our algorithm works for different users with various hand and finger sizes, without problems.

A hand pose is recognized in following steps:

1. The depth image is segmented into hand and non-hand regions. This is trivial once we assume that the user's right hand is the closest object to the camera.

2. For each pixel p that is in hand region, calculate $r(p)$ as

$$r(p) = \max(r \mid circle(p, r))$$

where $circle(p, r)$ is true if all pixels within r distance from p are in hand region and are of similar depth values. $S(p)$ is defined as all pixels within $r(p)$ from p.

3. Palm center p_0 (Figure 4-c) is located as

$$p_0 = \arg\max_p(r'(p))$$

where $r'(p)$ is the same as $r(p)$, except that depth similarity is not considered.

4. Candidate finger pixel set F are selected as

$$F = \{p \mid r(p) \in [\underline{r}, \bar{r}]\}$$

where range [0.5, 2] is used in our experiments.

5. Select the farthest pixel p_t in F as a possible candidate of fingertip. Starting from p_t, connect with neighboring pixels in F until too many pixels in F are encountered for the next step (no longer in a narrow band region as a finger should be). Fitting one to three cylinders to cover connected pixels $\{p\}$ and $\cup S(p)$ (Figire 4-e).

6. Remove the fitted area from hand region, and re-calculate $r(p)$ for those neighboring pixels. Then repeat step 5 until all obtrusive regions are fitted or the farthest pixel p_t is close to palm.

7. If less then 5 fingers are identified, we assume that the missing fingers are merged with the palm. Locate candidate fingertip as the nearest pixel to palm center in F whose depth is discontinuous from surroundings (Figure 4-g). The fitting process is the same as step 5.

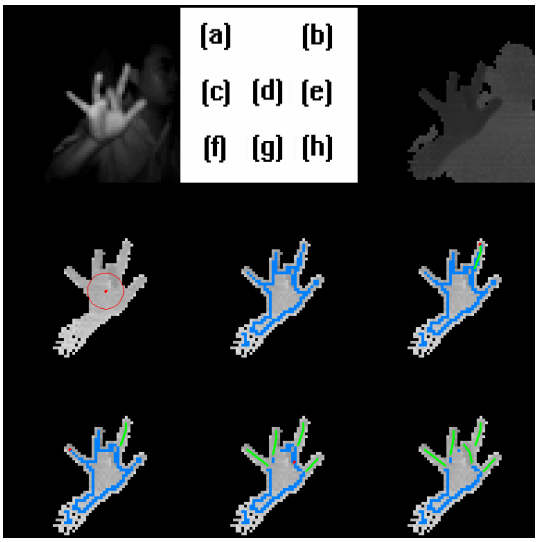

Figure 4: (a) brightness image; (b) depth image; (c) palm center; (d) candidate finger pixels; (e) fingertip candidate and fitted finger (green line as cylinder axis); (f) next candidate; (g) fingertips merged with the palm; (h) all 5 fingers located.

If more then 5 fingers are identified, combine them based on geometrical continuity. This will only happen when self-occlusion is present.

Quantitatively, our algorithm suffers from errors due to the low-resolution input. However, finger configurations are always correctly recognized, which satisfies our framework requirements.

3.2. GeLex: the Language

The power and flexibility of our framework arises from the gesture language. We constructed a suitable language based on two simple desired attributes:

Human friendly: there should be a limited set of characters in the pose alphabet and the alphabet should intuitively relate to hand poses.

Computer friendly: pose spellings should be mapped into hand poses without ambiguity.

Since the pioneering Stokoe Notations [11], several gesture notation systems were developed for describing sign languages. Among them are SignWriting [12], which is recognized as the most expressive [13]. However, SignWriting contains over 600 symbols and takes significant effort to learn.

SignWriting can represent all possible gestures, but only a subset is suitable for human computer interaction. Thus, instead of using SignWriting, we developed a simpler notation system called GeLex.

GeLex can express two types of hand poses, full or partial. For full hand poses, each finger is specified; for partial hand poses, some finger states are undefined or unknown. For example, a "fist" is a full hand pose, whereas an "index finger pointing" is a partial hand pose. Partial poses greatly simplify interface specifications and provide flexibility for customizing the interface to user preferences.

There are four types of alphabet elements in the GeLex system:

Entities: hands, fingers, phalanxes, joints;

Orientations: local orientations for view-independent hand poses; global orientations for hand-camera relations;

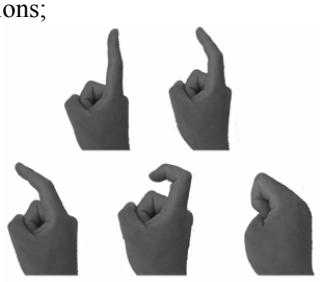

Figure 5: five finger states. In upper row: point, half bend; in lower row: bend, half closed, and closed.

Finger states: point, bend, half bend, closed, half closed (see Figure 5);

Finger inter-relations: group/separate, cross, touch (see Figure 6).

Figure 6: finger inter-relations: group, separate, cross, and touch.

Each hand pose notation starts with a distinctive ID (a positive integer) and a colon, followed by factors, and ending with a period. Each factor contains entities and entity states. Factors are separated by semicolons, and alphabet elements are separated by commas.

For convenience, we define four base poses: Fist, Flat, OpenHand, and Ball (see Figure 7). A full hand pose can be expressed directly using components, or as deviation from one of the four base-poses.

Figure 7: four base-poses in GeLex

For a detailed description of GeLex, see Appendix A.

GeLex is not comprehensive, but most natural hand poses used in prior HCI work are covered. There are 41 basic hand shapes in SignWriting, and all of them can be notated using GeLex. See Appendix B.

We have not done an extensive user study, however, two people with no expertise in either computer science or gesture interfaces were asked to learn GeLex for 30 minutes. After that, both of them are able to create hand poses correctly and fluently.

3.3. Pose Template Matching

This section explains the mechanism for adapting the generic pose recognizer to a specific application.

Each pose from the interface description is considered as a template. To compare template poses (as GeLex notations) and input poses (represented as cylinders), a unified representation is needed. In our framework, state vectors are defined to represent hand poses, and both GeLex notations and cylinders are mapped to the vector space and then compared.

The elements of a state vector are defined as follows:

- 25 elements for finger states. Element p_{ij} (i=1~5, j=1~5) indicates the probability of finger i in state j (j as enumerate of *Point*, *Bend*, *HalfBend*, *Close*, and

HalfClose). For example, (p_{11}, p_{12}, p_{13}, p_{14}, p_{15})=(0, 0, 0, 1, 0) indicates that the thumb is closed; (p_{21}, p_{22}, p_{23}, p_{24}, p_{25})=(0, 0.5, 0.5, 0, 0) indicates that the index finger is between bend and half-bend.

- 4 elements for finger grouping (see Figure 6). Element g_i=1 (i=1~4) indicates that finger i and i+1 are grouped together; g_i=0 indicates two fingers are separated.
- 15 elements for finger orientation. Elements (o_{i1}, o_{i2}, o_{i3}) (i=1~5) are defined as $_{(\alpha/\frac{\pi}{2},\beta/\frac{\pi}{2},\gamma/\frac{\pi}{2})}$ where (α,β,γ) are the Euler angles of finger i in the local orientation system as shown in Figure 10 left. If finger i's state is not pointing, then (o_{i1}, o_{i2}, o_{i3})=(0, 0, 0).
- 4 elements for finger touch. Elements t_i (i=2~5) defines how the thumb is touched with finger i: t_i=0 indicating no touch, t_i=1 indicating touched at fingertips, t_i=2 indicating touched at other parts. There are other possible finger touches, for example, index finger and ring finger touch at fingertips. However, such poses are rarely used in human computer interaction and thus ignored in the current implementation.

The above totals up to a forty-eight element state vector.

Mapping from GeLex notations to state vectors is simple. Given a specific hand pose, its GeLex notation may not be unique. However, its vector representation in our framework is unique.

Mapping from cylinder representations to state vectors is more complicated.

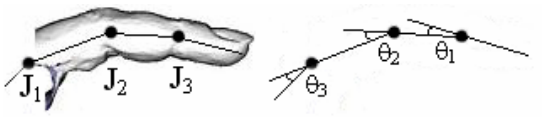

Figure 8: three angles specifying a finger's state.

As shown in Figure 8, each finger state is defined by three angles, which can be inferred from cylinder representations. As in most natural hand poses, the constraint $\theta_1 = \frac{2}{3}\theta_2$ holds (see [2]), so θ_1 could be ignored.

The probability of a finger in a certain state is defined as:

$$p_j = k_j \cdot \exp\left(\frac{-(\theta_2 - \mu_{2j})^2}{2\cdot(\pi/16)^2}\right) \cdot \exp\left(\frac{-(\theta_3 - \mu_{3j})^2}{2\cdot(\pi/16)^2}\right) + c_j$$

(for $|\theta_2 - \mu_{2j}| < \sigma_{2j}$, $|\theta_3 - \mu_{3j}| < \sigma_{3j}$)

$$p_j = 0$$

(otherwise)

where μ_{2j}, δ_{2j}, μ_{3j}, and δ_{3j} are predefined for each finger state j, as listed in Table 1; and k_j and c_j are constants to scale p_j to the range of [0, 1].

	μ_{2j}	δ_{2j}	μ_{3j}	δ_{3j}
Point	0	π/8	-	-
Bend	π/4	π/6	π/4	π/6
HalfBend	π/4	π/6	0	π/8
Close	π/2	π/8	π/2	π/8
HalfClose	π/2	π/8	0	π/8

Table 1: predefined values for each finger state *j*.

Generally speaking, μ_{2j} and μ_{3j} defines the "perfect pose" for state j. As a pose deviates from the "perfect pose," the probability of a hand in that state decreases correspondingly.

One exception for the above formulation is the pointing state, where only θ_2 is considered. Another exception is the thumb, where only two (instead of three) joints exist. However, the computation of p_j is similar.

Each template pose i is mapped to a state vector v_i off-line, thus forming a match candidate set $\{v_i\}$.

Given a new pose recognized from a depth-image, it is mapped to a state vector v that is compared to all templates $\{v_i\}$, and pose i is recognized if the difference between v and v_i is below a threshold. The threshold is set to control how "precise" the gesture interface should be. This could be set by users for their comfort. A "less-strict" threshold is suggested considering the pose recognition errors due to low-resolution camera input. As for partial poses, un-identified fingers and corresponding vector elements are ignored and comparison is performed on sub-vectors.

Since pose matching is performed as a vector comparison, it can be processed efficiently. All candidate vectors are pre-computed off-line, so recognition speed is not affected significantly as the candidate set size increases to include more gestures in an application system.

The generated interface is in the form of a set of API functions as follows (in the C programming language):

```
int GetCurrentHandPose();
Coord3D GetPalmCenterPosition();
Coord3D GetPalmOrientation();
Coord3D GetFingerTipPosition(int FingerIndex);
Coord3D GetFingerOrientation(int FingerIndex);
```

4. Application: an Example

To show the effectiveness of the framework, we built a gesture-driven system to implement the "Scissor-Rock-Paper" game by defining an interface within the framework.

Figure 9: three hand poses in the "Scissor-Rock-Paper" game.

For each round of the game, a user is asked to present one of the three gesture poses as shown in Figure 9. The program randomly produces its own selection of "Scissor," "Rock," or "Paper." The winner is based on the simple rules: "Scissor<Rock," "Rock<Paper," "Paper<Scissor," and "Scissor<Rock."

The GeLex script for the gesture interface of this game is only three lines, as follows:

```
1 : Fist. /* Rock */
2 : Fist; F2, Point(Up); F3, Point(Up).
    /* Scissor */
3 : OpenHand. /* Paper */
```

Pseudo-code for one game round is:

```
Recognize user's gesture uBid;
If (uBid is not "Rock," "Scissor," or "Paper") {
   Display: "invalid bid";
} Else {
   Generate computer's random selection cBid;
   If (uBid>cBid)
      Display: "you win!";
   Else
      Display: "computer wins!";
}
End-of-Round;
```

The whole game was built within thirty minutes, and the gesture interface was built within ten minutes.

5. Conclusion

In this paper, we present a framework that enables designers to rapidly and easily build a gesture interface for an application. Our framework serves as a black box, which conceals the implementation details of gesture recognition from designers. The input is the interface description provided by designers, and the output is the desired gesture interface. All designers need to do is to provide a text description of the poses desired in the interface. The application end users could also configure the interface for their preferences by changing the description.

Due to the low-resolution images, the generic pose recognition algorithm is limited to front views within a +/-30degree rotation. This limitation can be alleviated by using a higher resolution laser-illuminated camera.

Our goal is to enable the rapid simple building of any gesture interface through our framework. The current implementation of GeLex is limited to static hand poses. In many situations, dynamic gestures are also desirable as user inputs. Thus, extending GeLex to dynamic gestures is our next research goal.

Another extension is from single-hand poses to two-hand poses. Inter-hand interaction is common in sign language, and may also be desirable in certain systems.

Acknowledgement

This project is funded by the National Science Foundation through its ERC program grant to the Integrated Media Systems Center (IMSC) at USC.

Thank J.P. Lewis for insightful discussion.

Reference

[1] T. Starner, B. Leibe, D. Minnen, T. Westeyn, A. Hurst, and J. Weeks. Integration of wireless gesture tracking, object tracking, and 3D reconstruction in the Perceptive Workbench. In *Machine Vision and Applications*, Vol. 14, No. 1, pages 59-71, 2003.

[2] V.I. Pavlovic, R. Sharma, and T.S. Huang. Visual interpretation of hand gestures for human-computer interaction: a review. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol.19, No.7, July 1997.

[3] J. Rehg and T. Kanade. Visual tracking of high DOF articulated structures: an application to human hand tracking. In *3rd European Conference on Computer Vision (ECCV'94)*, pages 35-46, Springer-Verlag, Stockholm, Sweden, May 1994.

[4] B. Stenger, P. R. S. Mendonça, and R. Cipolla. Model-based 3D tracking of an articulated hand. *Proc. Conference on Computer Vision and Pattern Recognition*, Vol. II, pages 310-315, Kauai, USA, December 2001.

[5] Y. Wu, Lin J. Y., and T.S. Huang. Capturing natural hand articulation. In *Proc. 8th Int. Conf. on Computer Vision*, volume II, pages 426-432, Vancouver, Canada, July 2001.

[6] N. Shimada, Y. Shirai, Y. Kuno, and J. Miura. Hand gesture estimation and model refinement using monocular camera - ambiguity limitation by inequality constraints. In

Proc. of 3rd Int'l Conf. On Automatic Face and Gesture Recognition, pages 268-273, 1998.

[7] S.U. Lee and I. Cohen. 3D hand reconstruction from a monocular view. In *Proc. of the 7th Int'l Conf. On Pattern Recognition (ICPR'04)*, Cambridge, United Kingdom, August 2004.

[8] Y. Wu and T.S. Huang. View-independent recognition of hand postures. In *CVPR*, volume 2, pages 88-94, 2000.

[9] N. Shimada, K. Kimura, and Y. Shirai. Real-time 3-D hand estimation on 2-D appearance retrieval using monocular camera. In *Proc. of the IEEE ICCV Workshop on RATFG-RTS'01*, pages 23-30, 2001.

[10] V. Athitsos and S. Sclaroff. An appearance-based framework for 3d hand shape classification and camera viewpoint estimation. In *Proc. 5th Int'l Conf. On Automatic Face and Gesture Recognition*, pages 40-45, 2002.

[11] W.C. Stokoe, D.C. Casterline, and C.G. Croneberg. A dictionary of American Sign Language on linguistic principles. Gallaudet College Press, Washington D.C., 1965.

[12] Http://www.signwriting.org/

[13] A. Rosenberg. Writing signed languages: in support of adopting an ASL writing system. Matser's thesis, University of Kansas, Department of Linguistics, 1999.

Appendix A: GeLex Notation System

1. Entities

- Two hands: *LeftHand* and *RightHand*.
- Five finger digits: F_1 (thumb), F_2 (index finger), F_3 (middle finger), F_4 (ring finger), and F_5 (baby finger).
- Phalanxes: P_1 (top phalanx, closest to fingertip), P_2, and P_3. Thumb contains 2 phalanxes and others contain 3. For each phalanx, there are *Face*, *Back*, *Side1* (as closer to Thumb), and *Side2*.
- Finger joints: J_0 (fingertip, a virtual joint), J_1 (connecting P_1 and P_2), J_2, and J_3. Thumb contains 3 joints and others contain 4.

Entities are expressed in hierarchy. For example, fingertip of index finger of right hand is expressed as: $RightHand.F_2.J_0$. Right hand is assumed by default: $RightHand.F_2$ is equal to F_2.

2. Orientation systems

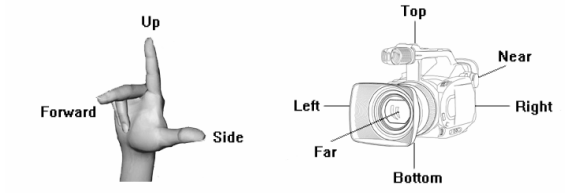

Figure 10: a local hand orientation system, and a view-dependent global orientation system.

- Local system to describe view-independent hand poses. As shown in Figure 10 left, three orientations are specified: *Forward*, *Up*, and *Side*.
- Global system to describe the relation between a hand and the viewing camera. As shown in Figure 10 right, six directions are specified in a global system: *Near*, *Far*, *Left*, *Right*, *Top*, *Bottom*.

Combinations are allowed in each system, for example, *Forward-Up* or *Near-Left-Top*.

3. Finger states (see Figure 5)

- ***Point***: a parameter could be used to specify pointing orientation.
- ***BendHalf***.
- ***Bend***.
- ***CloseHalf***.
- ***Close***.

4. Finger inter-relations (see Figure 6)

- ***Group***: 2 or more parameters, specifying a set of fingers grouped together. By default, fingers are separated.
- ***Cross***: 2 parameters, first one specifying the back finger, second one specifying the front finger.
- ***Touch***: 2 parameters, specifying 2 touching entities. By default, fingers touch at fingertips, i.e., *Touch*(F_1, F_2) is equal as *Touch*($F_1.J_0$, $F_2.J_0$).

5. Base-poses (see Figure 7)

- ***Fist***: *Group*(F_2, F_3, F_4, F_5), *Close*; F_1, *Close*.
- ***Flat***: *Group*(F_1, F_2, F_3, F_4, F_5), *Point*(*Up*).
- ***Ball***: a parameter is to specify which finger the thumb touches. For example, *Ball*(3) is equal as: *Group*(F_2, F_3, F_4, F_5), *Bend*; F_1, *Bend*; *Touch*(F_1, F_3).
- ***OpenHand***: F_1, *Point*(*Up-Side*); F_2, *Point*(*Up*); F_3, *Point*(*Up*), F_4, *Point*(*Up*), F_5, *Point*(*Up*).

Appendix B: 41 Basic Hand Poses

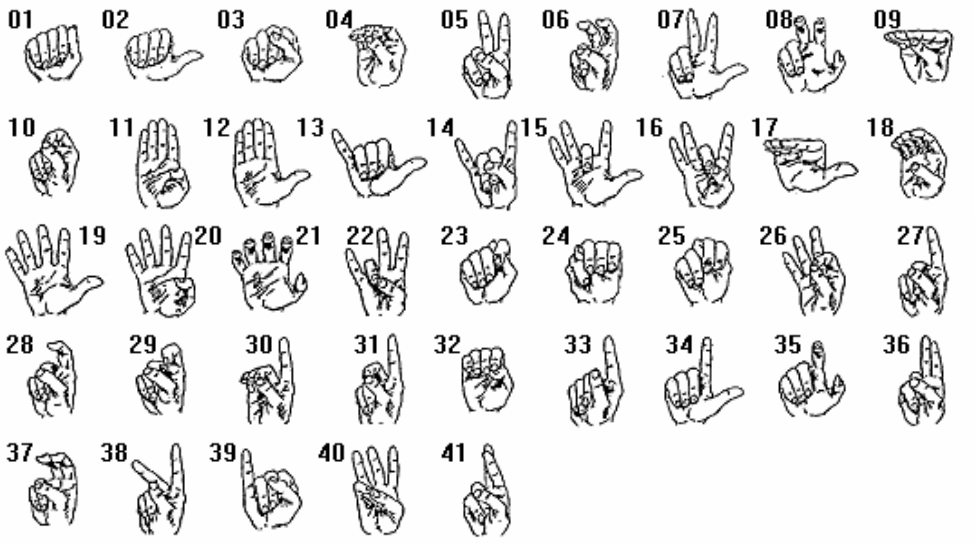

Figure 11: 41 basic hand poses in SignWriting (copied from [11])

01: Fist; F1, Point; Touch(F1.P1.Face, F2.P3.Side1).
02: Fist; F1, Point(Side).
03: Fist.
04: Ball(3).
05: Fist; F2, Point(Up); F3, Point(Up).
06: Fist; F2, BendHalf; F3, BendHalf.
07: OpenHand; Group(F4, F5), Close.
08: F1, BendHalf; F2, BendHalf; F3, BendHalf; Group(F4, F5), Close.
09: Group(F2, F3, F4, F5), Point(Forward); F1, Point; Touch(F1.P1.Face, F2.P1.Face).
10: Ball(2); Group(F3, F4, F5), Close.
11: Flat; F1, CloseHalf.
12: Flat; F1, Point(Side).
13: OpenHand; Group(F2, F3, F4), Close.
14: Fist; F2, Point(Up); F5, Point(Up).
15: OpenHand; F3, Close.
16: Ball(3); F2, Point(Up); F4, Point(Up); F5, Point(Up).
17: Group(F2, F3, F4, F5), Point(Forward); F1, Point(Side).
18: Group(F2, F3, F4, F5), BendHalf; F1, Bend.
19: OpenHand.
20: OpenHand; F1, CloseHalf.
21: F1, BendHalf; F2, BendHalf; F3, BendHalf; F4, BendHalf; F5, BendHalf.
22: Ball(4); F2, Point(Up); F3, Point(Up); F5, Point(Up).
23: Group(F3, F4, F5), Close; Touch(F1.P1.Face, F3.P3.Side1); F2, Close.
24: F5, Close; Touch(F1.P1.Face, F5.P2.Back); Group(F2, F3, F4), Close.
25: Group(F4, F5), Close; Touch(F1.P1.Face, F4.P2.Side1); Group(F2, F3), Close.
26: Ball(2); F3, Point(Up); F4, Point(Up); F5, Point(Up).
27: Fist; F2, Point(Up).
28: Fist; F2, BendHalf.
29: Fist; F2, CloseHalf.
30: Ball(3); F2, Point(Up).
31: Ball(3); Group(F4, F5), Close; F2, Point(Up).
32: Fist; F1, CloseHalf.
33: Fist; F1, Point(Up); F2, Point(Up).
34: Fist; F1, Point(Side); F2, Point(Up).
35: Fist; F1, BendHalf; F2, BendHalf.
36: Fist; Group(F2, F3), Point(Up).
37: Fist; Group(F2, F3), BendHalf.
38: Group(F4, F5), Close; F3, Point(Up-Forward).
39: Fist; F5, Point(Up).
40: Ball(5); F2, Point(Up); F3, Point(Up); F4, Point(Up).