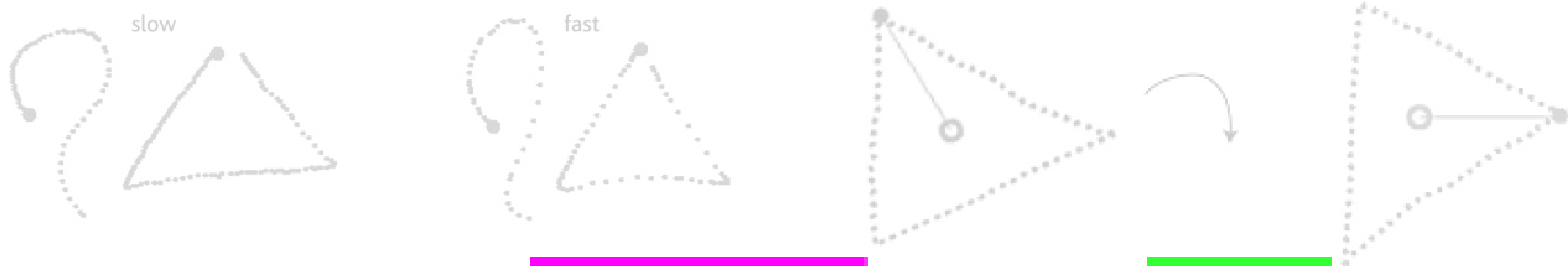


Making possible things easy

The $1 unistroke recognizer (UIST 2007)

UIST 2024 Lasting Impact Award

Jacob O. Wobbrock, Andrew D. Wilson, Yang Li

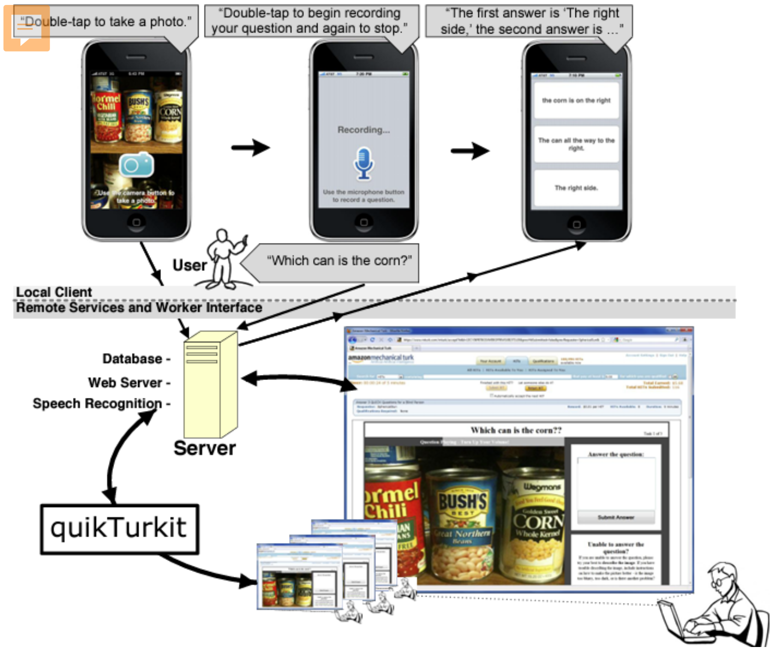
“UIST is about making impossible things **possible**,

and about making possible things **easy**...

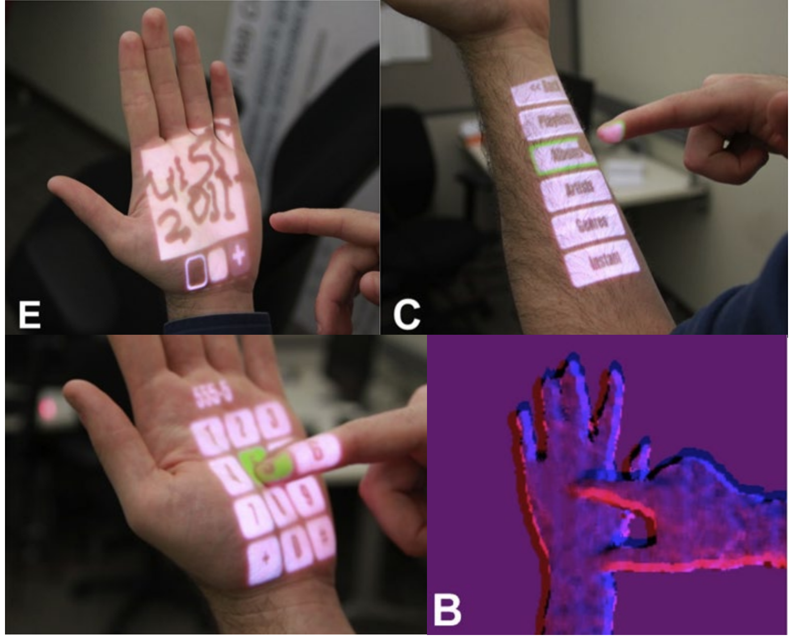
in service of **people**.

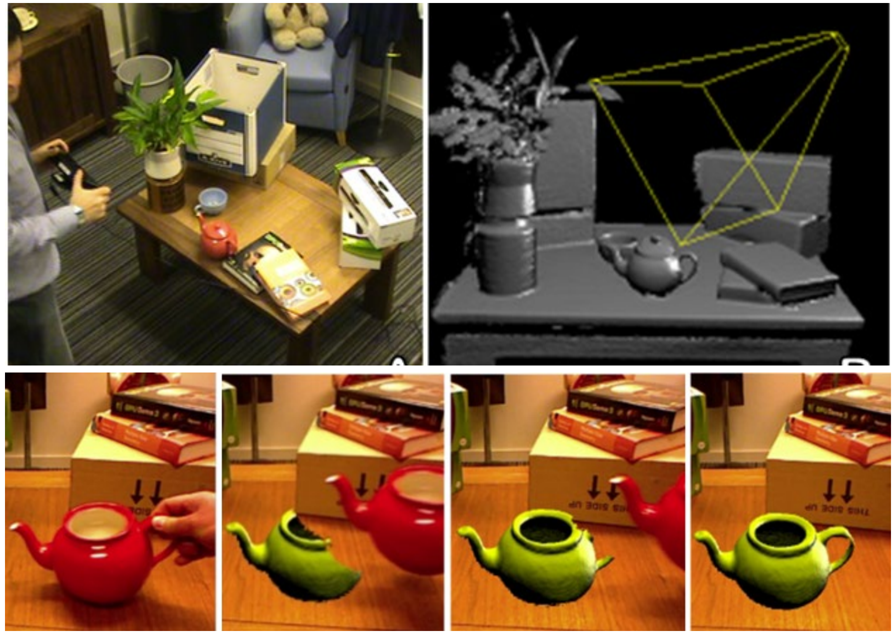
– Scott Hudson



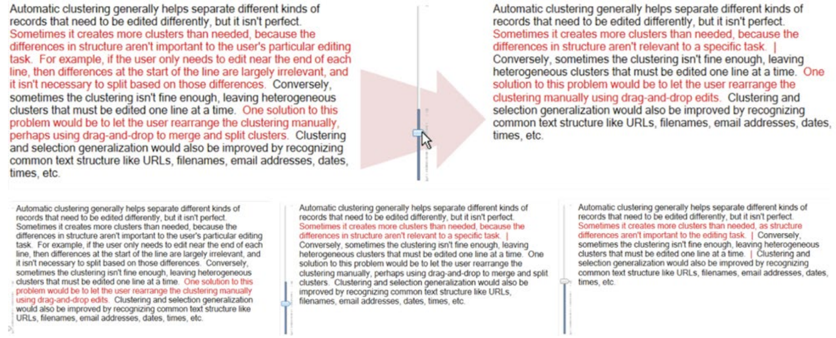
VizWiz, Bigham et al. (2010)



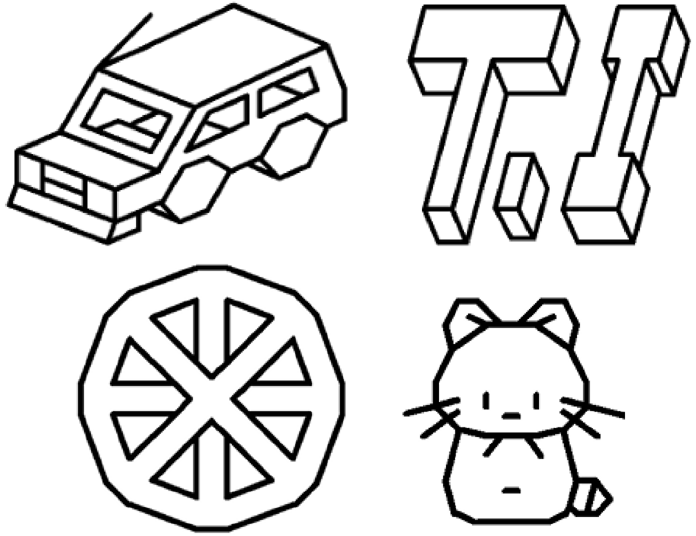
OmniTouch, Harrison et al. (2011)



KinectFusion, Izadi et al. (2011)



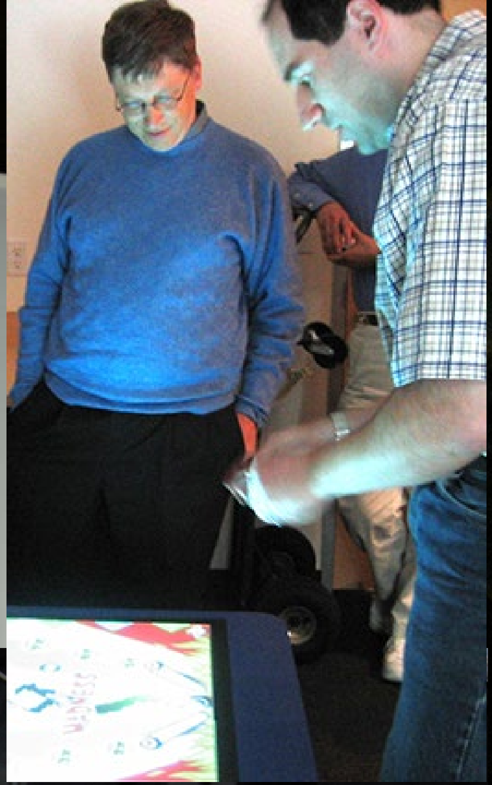
Soylent, Bernstein et al. (2010)



Interactive Beautification, Igarashi et al. (2007)

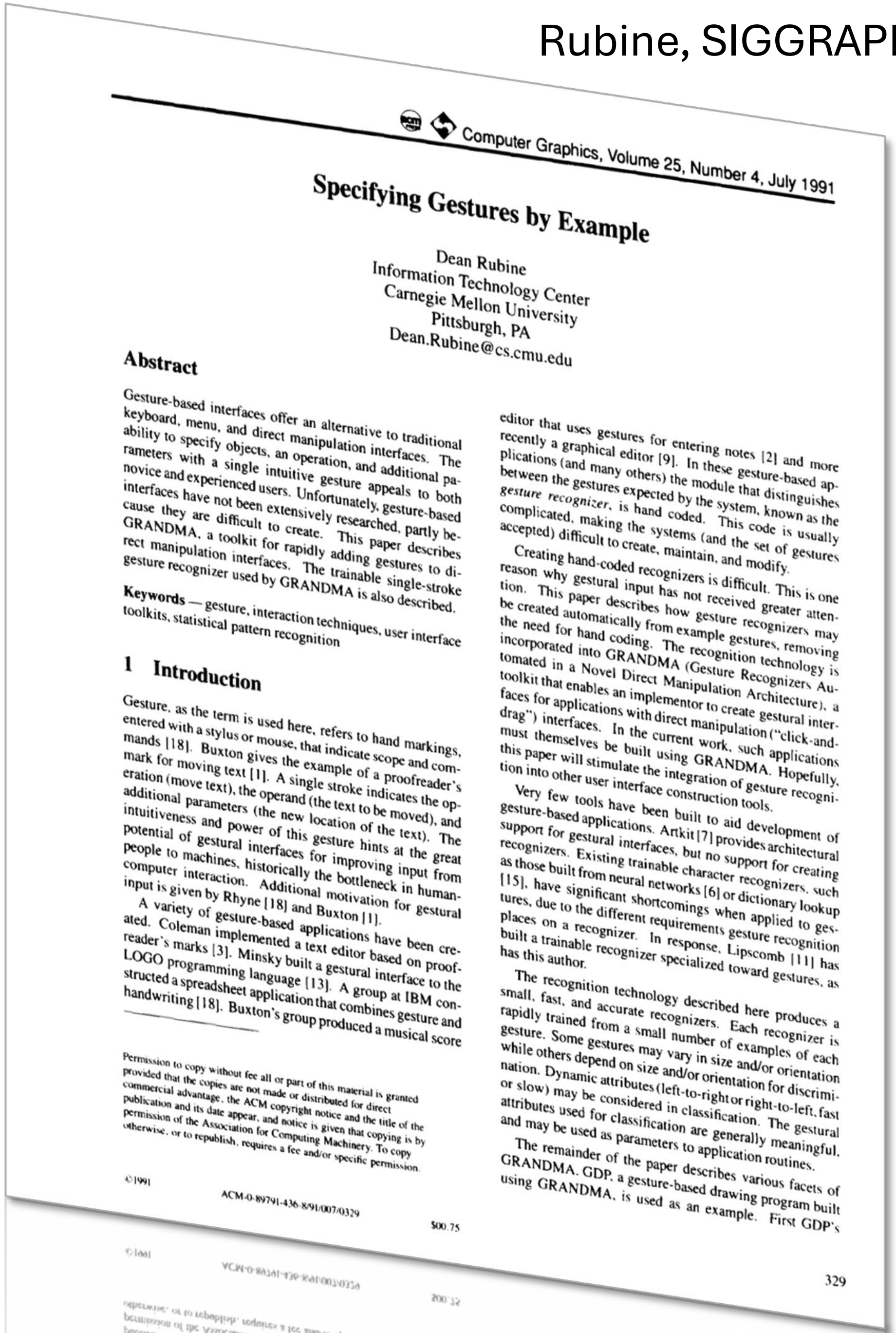
SearchTogether, Morris & Horvitz (2007)

How can we make 2-D gesture recognition **easy**?

go green


Rubine, SIGGRAPH 1991



Computer Graphics, Volume 25, Number 4, July 1991

Specifying Gestures by Example

Dean Rubine
Information Technology Center
Carnegie Mellon University
Pittsburgh, PA
Dean.Rubine@cs.cmu.edu

Abstract

Gesture-based interfaces offer an alternative to traditional keyboard, menu, and direct manipulation interfaces. The ability to specify objects, an operation, and additional parameters with a single intuitive gesture appeals to both novice and experienced users. Unfortunately, gesture-based interfaces have not been extensively researched, partly because they are difficult to create. This paper describes GRANDMA, a toolkit for rapidly adding gestures to direct manipulation interfaces. The trainable single-stroke gesture recognizer used by GRANDMA is also described.

Keywords — gesture, interaction techniques, user interface toolkits, statistical pattern recognition

1 Introduction

Gesture, as the term is used here, refers to hand markings, entered with a stylus or mouse, that indicate scope and commands [18]. Buxton gives the example of a proofreader's mark for moving text [1]. A single stroke indicates the operation (move text), the operand (the text to be moved), and additional parameters (the new location of the text). The intuitiveness and power of this gesture hints at the great potential of gestural interfaces for improving input from people to machines, historically the bottleneck in human-computer interaction. Additional motivation for gestural input is given by Rhyne [18] and Buxton [1].

A variety of gesture-based applications have been created. Coleman implemented a text editor based on proofreader's marks [3]. Minsky built a gestural interface to the LOGO programming language [13]. A group at IBM constructed a spreadsheet application that combines gesture and handwriting [18]. Buxton's group produced a musical score editor that uses gestures for entering notes [2] and more recently a graphical editor [9]. In these gesture-based applications (and many others) the module that distinguishes between the gestures expected by the system, known as the *gesture recognizer*, is hand coded. This code is usually complicated, making the systems (and the set of gestures accepted) difficult to create, maintain, and modify.

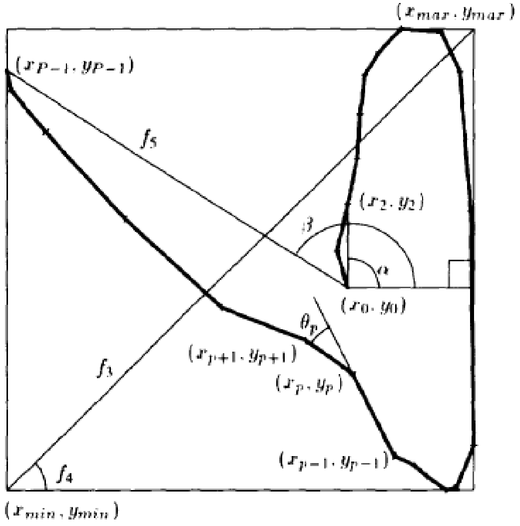
Creating hand-coded recognizers is difficult. This is one reason why gestural input has not received greater attention. This paper describes how gesture recognizers may be created automatically from example gestures, removing the need for hand coding. The recognition technology is incorporated into GRANDMA (Gesture Recognizers Automated in a Novel Direct Manipulation Architecture), a toolkit that enables an implementor to create gestural interfaces for applications with direct manipulation ("click-and-drag") interfaces. In the current work, such applications must themselves be built using GRANDMA. Hopefully, this paper will stimulate the integration of gesture recognition into other user interface construction tools.

Very few tools have been built to aid development of gesture-based applications. Artkit [7] provides architectural support for gestural interfaces, but no support for creating recognizers. Existing trainable character recognizers, such as those built from neural networks [6] or dictionary lookup [15], have significant shortcomings when applied to gestures, due to the different requirements gesture recognition places on a recognizer. In response, Lipscomb [11] has built a trainable recognizer specialized toward gestures, as has this author.

The recognition technology described here produces a small, fast, and accurate recognizers. Each recognizer is rapidly trained from a small number of examples of each gesture. Some gestures may vary in size and/or orientation while others depend on size and/or orientation for discrimination. Dynamic attributes (left-to-right or right-to-left, fast or slow) may be considered in classification. The gestural attributes used for classification are generally meaningful, and may be used as parameters to application routines.

The remainder of the paper describes various facets of GRANDMA. GDP, a gesture-based drawing program built using GRANDMA, is used as an example. First GDP's

ACM-0-89791-436-8/91/007/0329 $00.75



$$f_1 = \cos\alpha = (x_2 - x_0)/\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}$$

$$f_2 = \sin\alpha = (y_2 - y_0)/\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}$$

$$f_3 = \sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2}$$

$$f_4 = \arctan\frac{y_{max} - y_{min}}{x_{max} - x_{min}}$$

$$f_5 = \sqrt{(x_{P-1} - x_0)^2 + (y_{P-1} - y_0)^2}$$

$$f_6 = \cos\beta = (x_{P-1} - x_0)/f_5$$

$$f_7 = \sin\beta = (y_{P-1} - y_0)/f_5$$

$$\text{Let } \Delta x_p = x_{p+1} - x_p \qquad \Delta y_p = y_{p+1} - y_p$$

$$f_8 = \sum_{p=0}^{P-2}\sqrt{\Delta x_p^2 + \Delta y_p^2}$$

$$\text{Let } \theta_p = \arctan\frac{\Delta x_p \Delta y_{p-1} - \Delta x_{p-1}\Delta y_p}{\Delta x_p \Delta x_{p-1} + \Delta y_p \Delta y_{p-1}}$$

$$f_9 = \sum_{p=1}^{P-2}\theta_p$$

$$f_{10} = \sum_{p=1}^{P-2}|\theta_p|$$

$$f_{11} = \sum_{p=1}^{P-2}\theta_p^2$$

$$\text{Let } \Delta t_p = t_{p+1} - t_p$$

$$f_{12} = \max_{p=0}^{P-2}\frac{\Delta x_p^2 + \Delta y_p^2}{\Delta t_p^2}$$

$$f_{13} = t_{P-1} - t_0$$

Figure 6: Features used to identify strokes

Tappert, IBM 1982

C. C. Tappert

Cursive Script Recognition by Elastic Matching

Dynamic programming has been found useful for performing nonlinear time warping for matching patterns in automatic speech recognition. Here, this technique is applied to the problem of recognizing cursive script. The parameters used in the matching are derived from time sequences of x-y coordinate data of words handwritten on an electronic tablet. Chosen for their properties of invariance with respect to size and translation of the writing, these parameters are found particularly suitable for the elastic matching technique. A salient feature of the recognition system is the establishment, in a training procedure, of prototypes by each writer using the system. In this manner, the system is tailored to the user. Processing is performed on a word-by-word basis after the writing is separated into words. Using prototypes for each letter, the matching procedure allows any letter to follow any letter and finds the letter sequence which best fits the unknown word. A major advantage of this procedure is that it combines letter segmentation and recognition in one operation by, in essence, evaluating recognition at all possible segmentations, thus avoiding the usual segmentation-then-recognition philosophy. Results on cursive writing are presented where the alphabet is restricted to the lower-case letters. Letter recognition accuracy is over 95 percent for each of three writers.

1. Introduction

The advent in the early 1960s of electronic tablets capable of accurately capturing the *x-y* coordinate data of pen movement precipitated activity on cursive writing recognition [1, 2]. Except for an occasional thesis, activity in this area has decreased in recent years. Almost all efforts have been restricted to noncapital roman script [1, 2]. Two major approaches have been described in the literature. The approach undertaken at MIT by Eden and his students was basically one of "analysis by synthesis," where a model is created for the handwriting process and decoding is performed by fitting the model's production parameters [3]. Several more direct approaches were used by Harmon and his colleagues at Bell Laboratories, the most successful being an analysis on a letter-by-letter basis following explicit segmentation. Letter accuracy for this technique was first reported at 60 percent [4] and later, after improvements, at 90 percent [2] for carefully formed script.

However, with the increased interest in office systems, particularly those that reduce the principal's dependence on secretarial support, there is also a growing interest in communicating with machines in a person's natural modalities, such as speech and handwriting. For direct written input by principals, perhaps the most difficult technical problem is that of cursive script recognition.

The dynamic programming technique of elastic matching (dynamic time warping) was applied to speech recognition problems over a decade ago and has since become widespread [5–7]. Recently, elastic matching has been successfully applied to the recognition of discrete handwritten characters, where an input character is matched against each of a set of prototypes and assigned the name corresponding to the prototype yielding the best match [8]. Here, elastic matching is extended to the recognition of cursive writing. The procedure decodes handwritten words into estimated strings of letters. Operating on a word at a time, using letter prototypes, and allowing any letter to follow any letter, the decoder uses elastic matching to find the prototype sequence which best fits the unknown word, yielding the corresponding letter sequence as the estimated letter string. Elastic matching provides the decoder with the essential feature of being insensitive to minor perturbations of input letter shapes relative to prototype letter shapes. Thus, elastic matching is

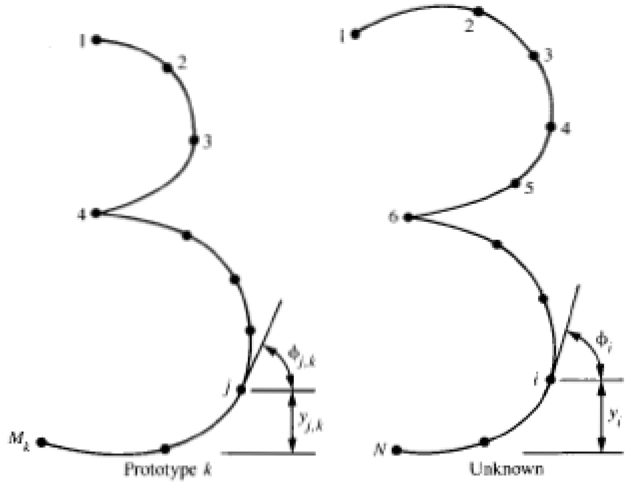


Figure 3 Model parameters: angle ϕ and height y.

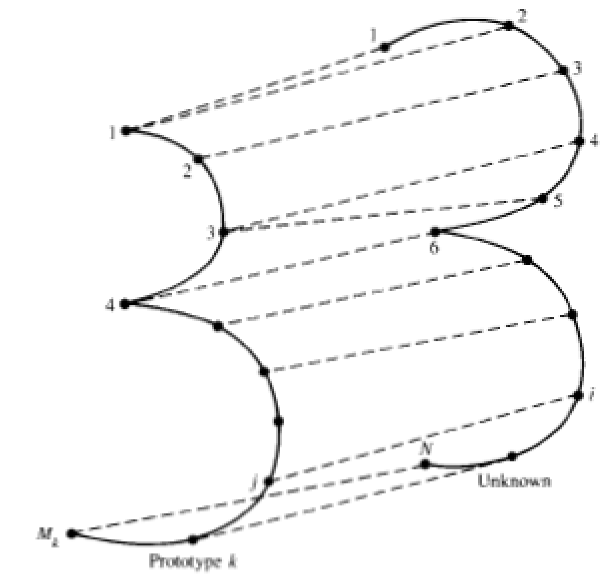


Figure 4 Elastic matching.

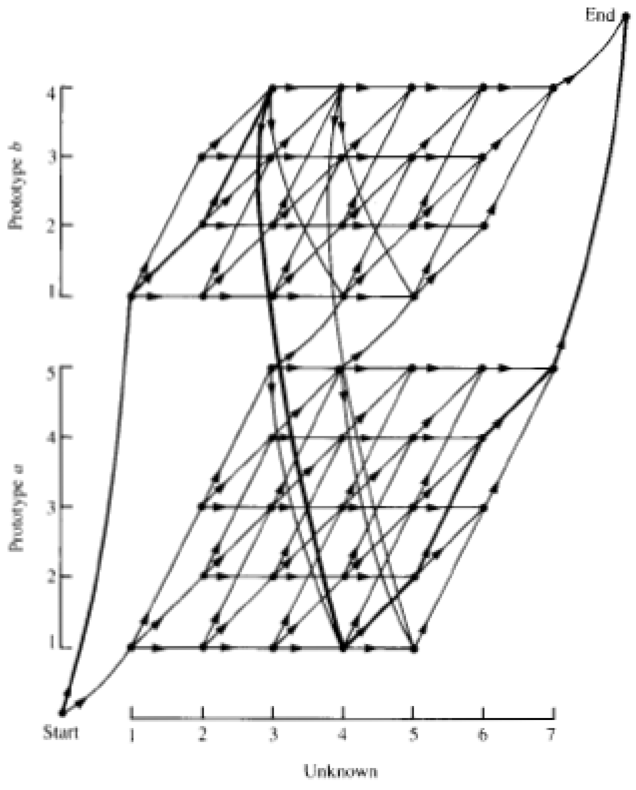
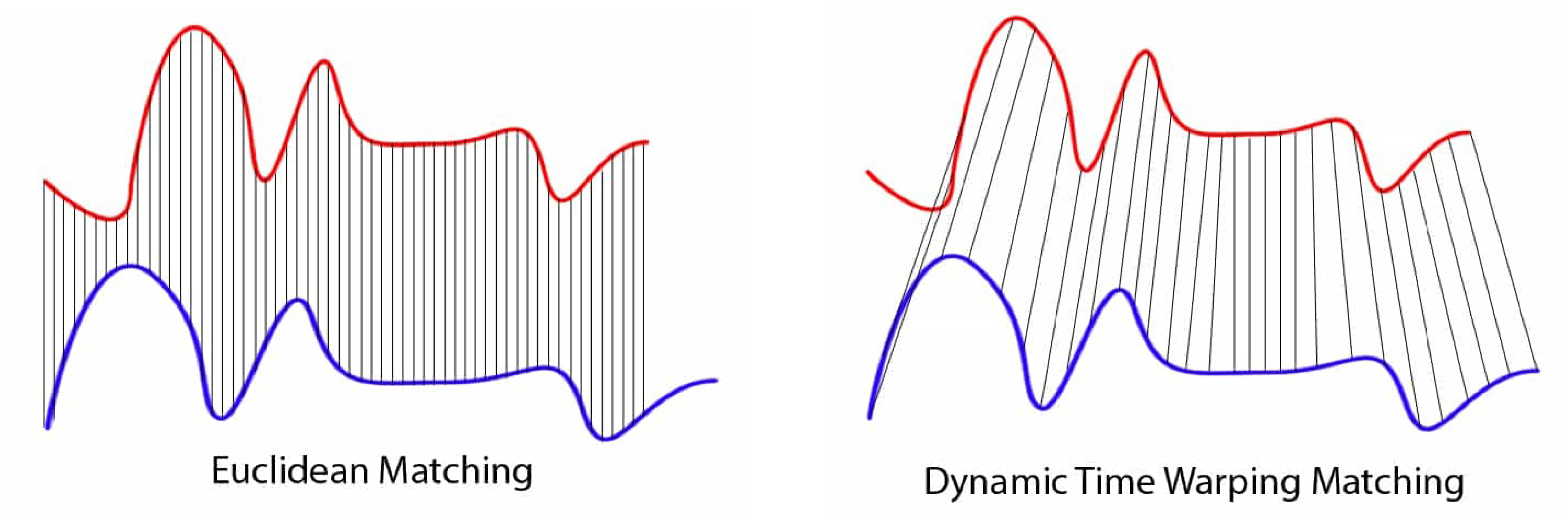
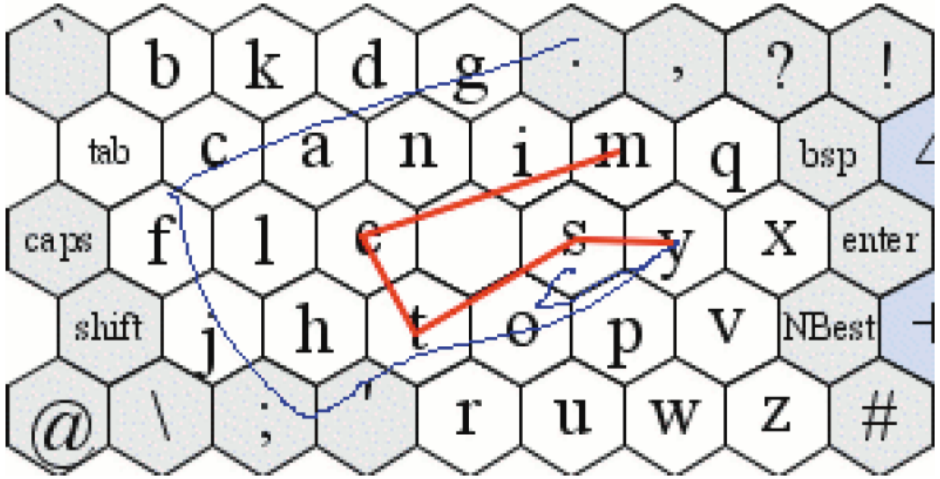


Figure 5 Lattice of simple, two-prototype model.

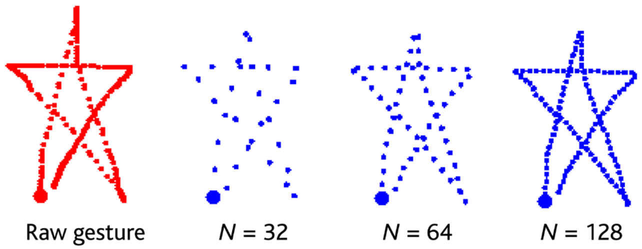

Euclidean Matching
Dynamic Time Warping Matching

SHARK², Kristensson & Zhai (2004)

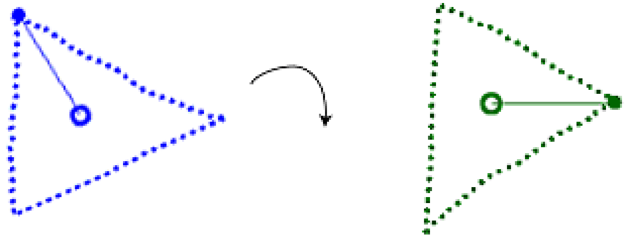


UIST Lasting Impact Award 2014

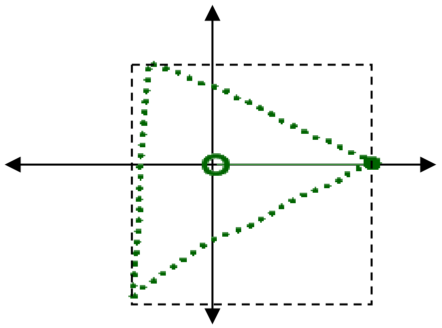
1. Resample



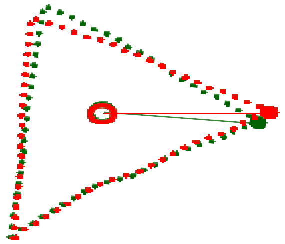
2. Rotate to 0°



3. Scale, translate



4. Compare to templates



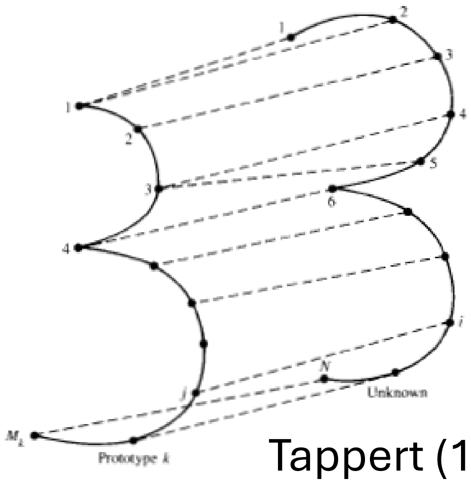
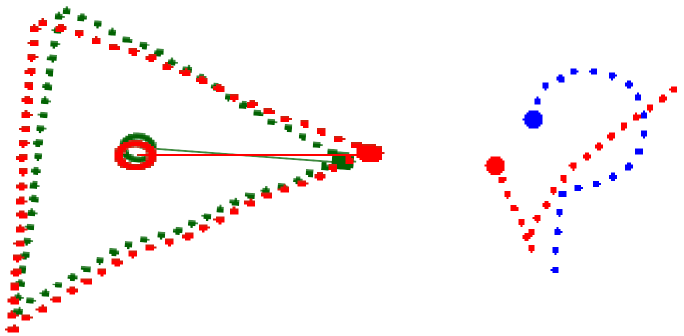


Figure 4 Elastic matching.

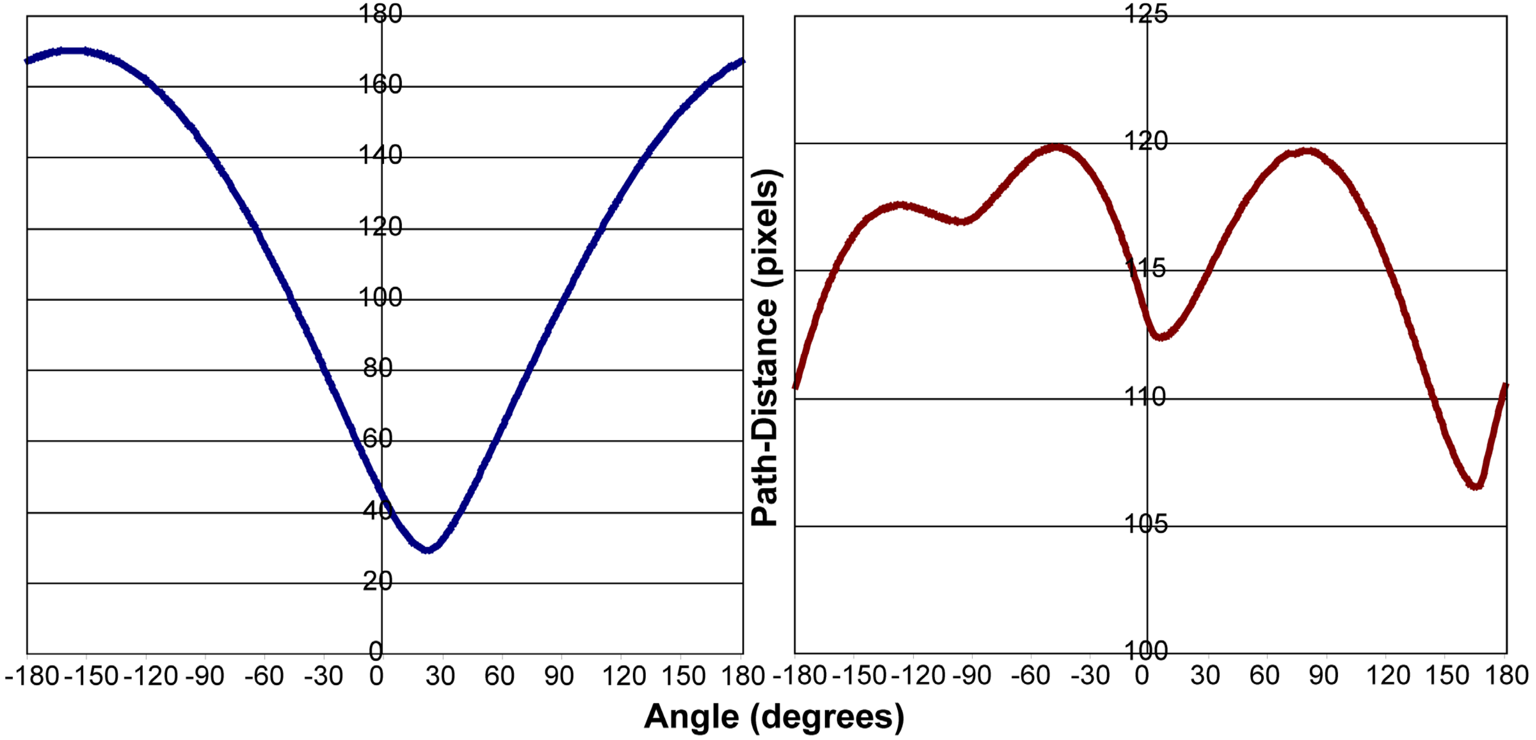
Tappert (1982) with “zero look-ahead”

Rotation invariance



Similar gestures | **Dissimilar gestures**

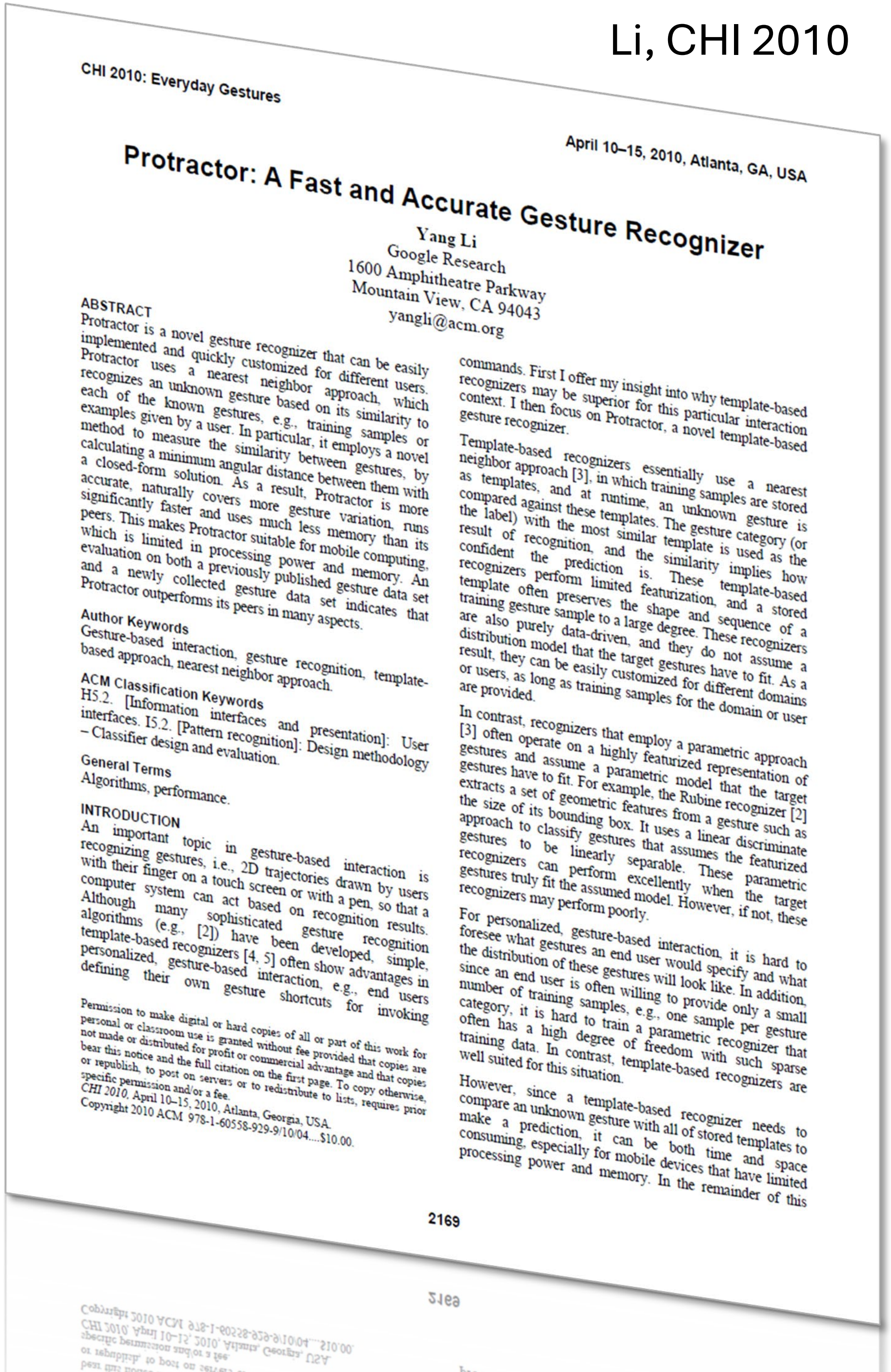
(a) question mark08 → question mark04 | **(b) v03 → caret01**



Search Method	***Rotations***
Brute force (+1° for 360°)	57,600
Hill climbing	8097
Golden Section Search	1600

*For a system with 10 templates for 16 gesture types (160 templates).

Li, CHI 2010



Protractor: A Fast and Accurate Gesture Recognizer

Yang Li
Google Research
1600 Amphitheatre Parkway
Mountain View, CA 94043
yangli@acm.org

ABSTRACT
Protractor is a novel gesture recognizer that can be easily implemented and quickly customized for different users. Protractor uses a nearest neighbor approach, which recognizes an unknown gesture based on its similarity to each of the known gestures, e.g., training samples or examples given by a user. In particular, it employs a novel method to measure the similarity between gestures, by calculating a minimum angular distance between them with a closed-form solution. As a result, Protractor is more accurate, naturally covers more gesture variation, runs significantly faster and uses much less memory than its peers. This makes Protractor suitable for mobile computing, which is limited in processing power and memory. An evaluation on both a previously published gesture data set and a newly collected gesture data set indicates that Protractor outperforms its peers in many aspects.

Author Keywords
Gesture-based interaction, gesture recognition, template-based approach, nearest neighbor approach.

ACM Classification Keywords
H5.2. [Information interfaces and presentation]: User interfaces. I5.2. [Pattern recognition]: Design methodology – Classifier design and evaluation.

General Terms
Algorithms, performance.

INTRODUCTION
An important topic in gesture-based interaction is recognizing gestures, i.e., 2D trajectories drawn by users with their finger on a touch screen or with a pen, so that a computer system can act based on recognition results. Although many sophisticated gesture recognition algorithms (e.g., [2]) have been developed, simple, template-based recognizers [4, 5] often show advantages in personalized, gesture-based interaction, e.g., end users defining their own gesture shortcuts for invoking commands. First I offer my insight into why template-based recognizers may be superior for this particular interaction context. I then focus on Protractor, a novel template-based gesture recognizer.

Template-based recognizers essentially use a nearest neighbor approach [3], in which training samples are stored as templates, and at runtime, an unknown gesture is compared against these templates. The gesture category (or the label) with the most similar template is used as the result of recognition, and the similarity implies how confident the prediction is. These template-based recognizers perform limited featurization, and a stored template often preserves the shape and sequence of a training gesture sample to a large degree. These recognizers are also purely data-driven, and they do not assume a distribution model that the target gestures have to fit. As a result, they can be easily customized for different domains or users, as long as training samples for the domain or user are provided.

In contrast, recognizers that employ a parametric approach [3] often operate on a highly featurized representation of gestures and assume a parametric model that the target gestures have to fit. For example, the Rubine recognizer [2] extracts a set of geometric features from a gesture such as the size of its bounding box. It uses a linear discriminate approach to classify gestures that assumes the featurized gestures to be linearly separable. These parametric recognizers can perform excellently when the target gestures truly fit the assumed model. However, if not, these recognizers may perform poorly.

For personalized, gesture-based interaction, it is hard to foresee what gestures an end user would specify and what the distribution of these gestures will look like. In addition, since an end user is often willing to provide only a small number of training samples, e.g., one sample per gesture category, it is hard to train a parametric recognizer that often has a high degree of freedom with such sparse training data. In contrast, template-based recognizers are well suited for this situation.

However, since a template-based recognizer needs to compare an unknown gesture with all of stored templates to make a prediction, it can be both time and space consuming, especially for mobile devices that have limited processing power and memory. In the remainder of this

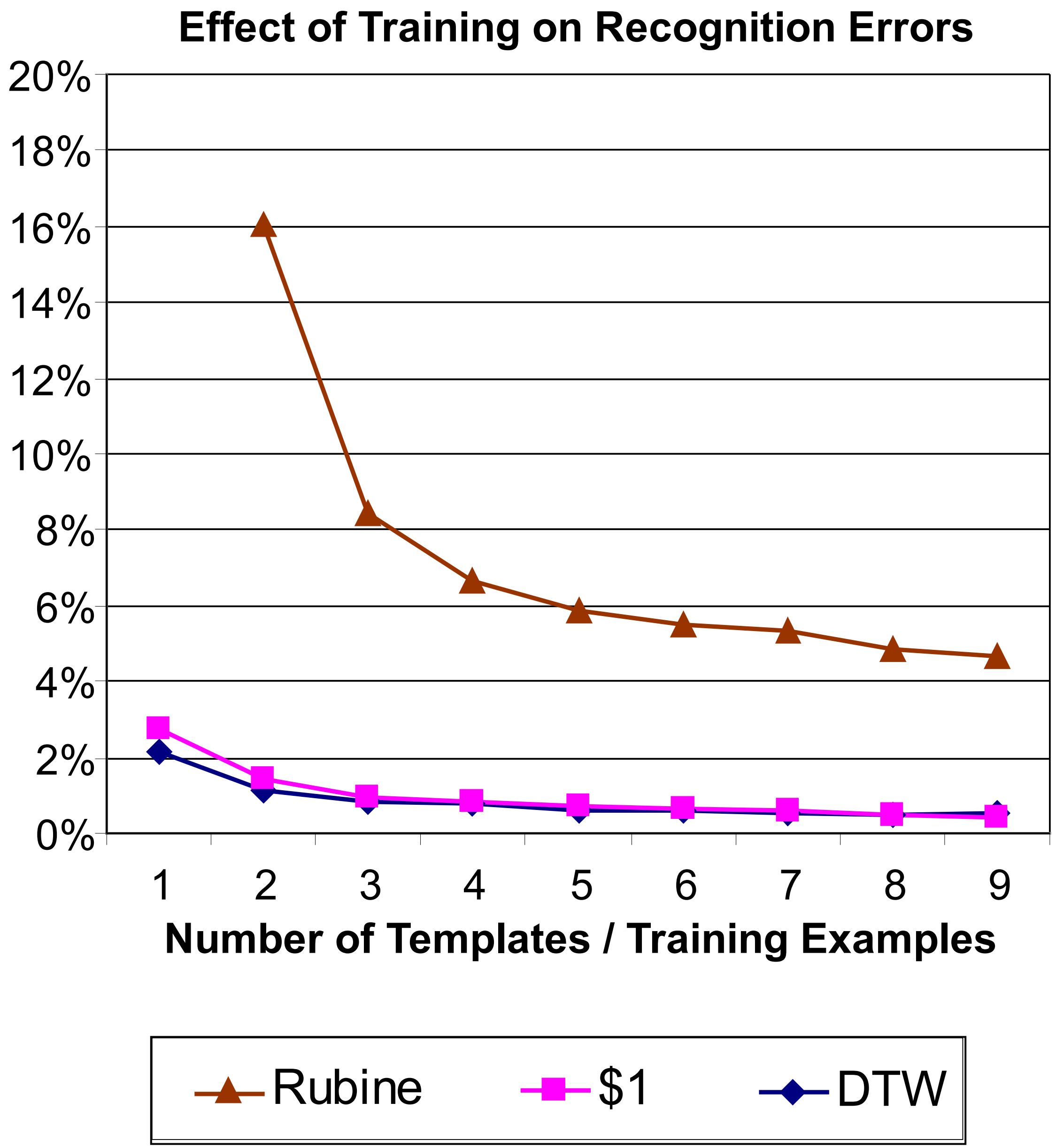
$$\theta_{optimal} = \arctan\frac{b}{a}$$

$$b = \sum_{i=1}^{n}\left(x_{ti}y_{gi} - y_{ti}x_{gi}\right)$$

$$S(t,g) = \frac{1}{\arccos\dfrac{v_t \cdot v_g}{|v_t||v_g|}}$$

$$v_t \cdot v_g = \sum_{i=1}^{n}\left(x_{ti}x_{gi} + y_{ti}y_{gi}\right)$$

$$|v_t||v_g| = \sqrt{\sum_{i=1}^{n}\left(x_{ti}^2 + y_{ti}^2\right)}\sqrt{\sum_{i=1}^{n}\left(x_{gi}^2 + y_{gi}^2\right)}$$



$1 was **7.3 times** more accurate than Rubine, and ran **80.7 times** faster than DTW.

handwriting recognition: A comprehensive survey. *IEEE Trans. Pattern Analysis & Machine Int. 22* (1), 63-84.

22. Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P. (1992) *Numerical Recipes in C*. Cambridge Univ. Press.
23. Rubine, D. (1991) Specifying gestures by example. *Proc. SIGGRAPH '91*. New York: ACM Press, 329-337.
24. Salvador, S. and Chan, P. (2004) FastDTW: Toward accurate dynamic time warping in linear time and space. *3rd Wkshp. on Mining Temporal and Sequential Data, ACM KDD '04*. Seattle, Washington (August 22-25, 2004).
25. Sezgin, T.M. and Davis, R. (2005) HMM-based efficient sketch recognition. *Proc. IUI '05*. New York: ACM Press, 281-283.
26. Stojmenović, M., Nayak, A. and Zunic, J. (2006) Measuring linearity of a finite set of points. *Proc. CIS '06*. Los Alamitos, CA: IEEE Press, 1-6.
27. Swigart, S. (2005) Easily write custom gesture recognizers for your Tablet PC applications. *Tablet PC Technical Articles*.
28. Tappert, C.C. (1982) Cursive script recognition by elastic matching. *IBM J. of Research & Development 26* (6), 765-771.
29. Tappert, C.C., Suen, C.Y. and Wakahara, T. (1990) The state of the art in online handwriting recognition. *IEEE Trans. Pattern Analysis & Machine Int. 12* (8), 787-808.
30. Vermunt, J.K. (1997) *Log-linear Models for Event Histories*. Thousand Oaks, CA: Sage Publications.
31. Wilson, A.D. and Shafer, S. (2003) XWand: UI for intelligent spaces. *Proc. CHI '03*. New York: ACM Press, 545-552.
32. Zhai, S. and Kristensson, P. (2003) Shorthand writing on stylus keyboard. *Proc. CHI '03*. New York: ACM Press, 97-104.

APPENDIX A – $1 GESTURE RECOGNIZER

Step 1. Resample a *points* path into *n* evenly spaced points.

```
RESAMPLE(points, n)
  I ← PATH-LENGTH(points) / (n – 1)
  D ← 0
  newPoints ← points_0
  foreach point p_i for i ≥ 1 in points do
     d ← DISTANCE(p_{i-1}, p_i)
     if (D + d) ≥ I then
        q_x ← p_{i-1,x} + ((I – D) / d) × (p_{i,x} – p_{i-1,x})
        q_y ← p_{i-1,y} + ((I – D) / d) × (p_{i,y} – p_{i-1,y})
        APPEND(newPoints, q)
        INSERT(points, i, q)   // q will be the next p_i
        D ← 0
     else D ← D + d
  return newPoints
PATH-LENGTH(A)
  d ← 0
  for i from 1 to |A| step 1 do
     d ← d + DISTANCE(A_{i-1}, A_i)
  return d
```

Step 2. Rotate *points* so that their indicative angle is at 0°.

```
ROTATE-TO-ZERO(points)
  c ← CENTROID(points)   // computes (x̄, ȳ)
  θ ← ATAN (c_y – points_0,y , c_x – points_0,x)   // for -π ≤ θ ≤ π
  newPoints ← ROTATE-BY(points, -θ)
  return newPoints
ROTATE-BY(points, θ)
  c ← CENTROID(points)
  foreach point p in points do
     q_x ← (p_x – c_x) COS θ – (p_y – c_y) SIN θ + c_x
     q_y ← (p_x – c_x) SIN θ + (p_y – c_y) COS θ + c_y
     APPEND(newPoints, q)
  return newPoints
```

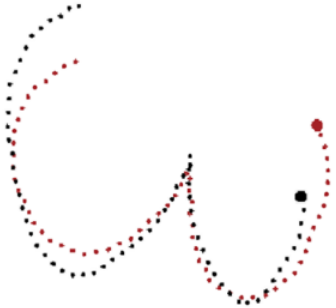
Step 3. Scale *points* so that the resulting bounding box will be of $size^2$ dimension; then translate *points* to the origin. BOUNDING-BOX returns a rectangle according to (min_x, min_y), (max_x, max_y). For gestures serving as templates, Steps 1-3 should be carried out once on the raw input points. For candidates, Steps 1-4 should be used just after the candidate is articulated.

```
SCALE-TO-SQUARE(points, size)
  B ← BOUNDING-BOX(points)
  foreach point p in points do
     q_x ← p_x × (size / B_width)
     q_y ← p_y × (size / B_height)
     APPEND(newPoints, q)
  return newPoints
TRANSLATE-TO-ORIGIN(points)
  c ← CENTROID(points)
  foreach point p in points do
     q_x ← p_x – c_x
     q_y ← p_y – c_y
     APPEND(newPoints, q)
  return newPoints
```

Step 4. Match *points* against a set of *templates*. The *size* variable on line 7 of RECOGNIZE refers to the *size* passed to SCALE-TO-SQUARE in Step 3. The symbol φ equals ½(-1 + √5). We use θ=±45° and θ_Δ=2° on line 3 of RECOGNIZE. Due to using RESAMPLE, we can assume that A and B in PATH-DISTANCE contain the same number of points, i.e., $|A|=|B|$.

```
RECOGNIZE(points, templates)
  1  b ← +∞
  2  foreach template T in templates do
  3     d ← DISTANCE-AT-BEST-ANGLE(points, T, -θ, θ, θ_Δ)
  4     if d < b then
  5        b ← d
  6        T' ← T
  7  score ← 1 – b / 0.5√(size² + size²)
  8  return ⟨T', score⟩
DISTANCE-AT-BEST-ANGLE(points, T, θ_a, θ_b, θ_Δ)
  x_1 ← φθ_a + (1 – φ)θ_b
  f_1 ← DISTANCE-AT-ANGLE(points, T, x_1)
  x_2 ← (1 – φ)θ_a + φθ_b
  f_2 ← DISTANCE-AT-ANGLE(points, T, x_2)
  while |θ_b – θ_a| > θ_Δ do
     if f_1 < f_2 then
        θ_b ← x_2
        x_2 ← x_1
        f_2 ← f_1
        x_1 ← φθ_a + (1 – φ)θ_b
        f_1 ← DISTANCE-AT-ANGLE(points, T, x_1)
     else
        θ_a ← x_1
        x_1 ← x_2
        f_1 ← f_2
        x_2 ← (1 – φ)θ_a + φθ_b
        f_2 ← DISTANCE-AT-ANGLE(points, T, x_2)
  return MIN(f_1, f_2)
DISTANCE-AT-ANGLE(points, T, θ)
  newPoints ← ROTATE-BY(points, θ)
  d ← PATH-DISTANCE(newPoints, T_points)
  return d
PATH-DISTANCE(A, B)
  d ← 0
  for i from 0 to |A| step 1 do
     d ← d + DISTANCE(A_i, B_i)
  return d / |A|
```

$1 Unistroke Recognizer



Jacob O. Wobbrock, University of Washington [contact]
Andrew D. Wilson, Microsoft Research
Yang Li, University of Washington†

†Currently at Google

Download

$1 source code: JavaScript, C#
Dynamic time warping: C#
Rubine classifier: C#
Pseudocode: $1, Protractor
Unistroke gesture logs: XML
Paper: PDF

About

The **$1 Unistroke Recognizer** is a 2-D single-stroke recognizer designed for rapid prototyping of gesture-based user interfaces. In machine learning terms, $1 is an instance-based nearest-neighbor classifier with a 2-D Euclidean distance function, i.e., a geometric template matcher. $1 is a significant extension of the proportional shape matching approach used in SHARK[2], which itself is an adaptation of Tappert's elastic matching approach with zero look-ahead. Despite its simplicity, $1 requires very few templates to perform well and is only about 100 lines of code, making it easy to deploy. An optional enhancement called Protractor improves $1's speed.

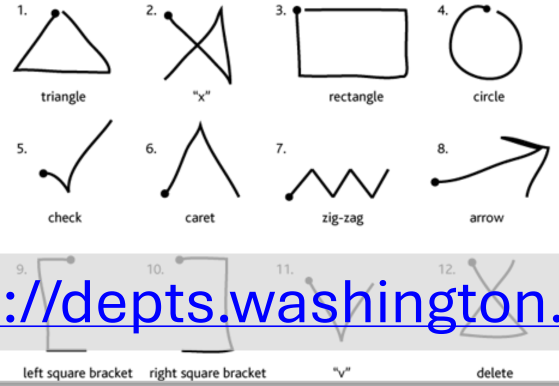
Upon its publication at UIST 2007, $1 was invited for a special reprise presentation at SIGGRAPH 2008.

The $N Multistroke Recognizer extends $1 to gestures with multiple strokes. The $P Point-Cloud Recognizer performs unistroke and multistroke recognition without the combinatoric overhead of $N, as it ignores stroke number, order, and direction. The $Q Super-Quick Recognizer extends $P for use on low-powered mobiles and wearables, as it is a whopping 142× faster and slightly more accurate.

The $-family recognizers have been built into numerous projects and even industry prototypes, and have had many follow-ons by others. Read about the $-family's impact.

Demo

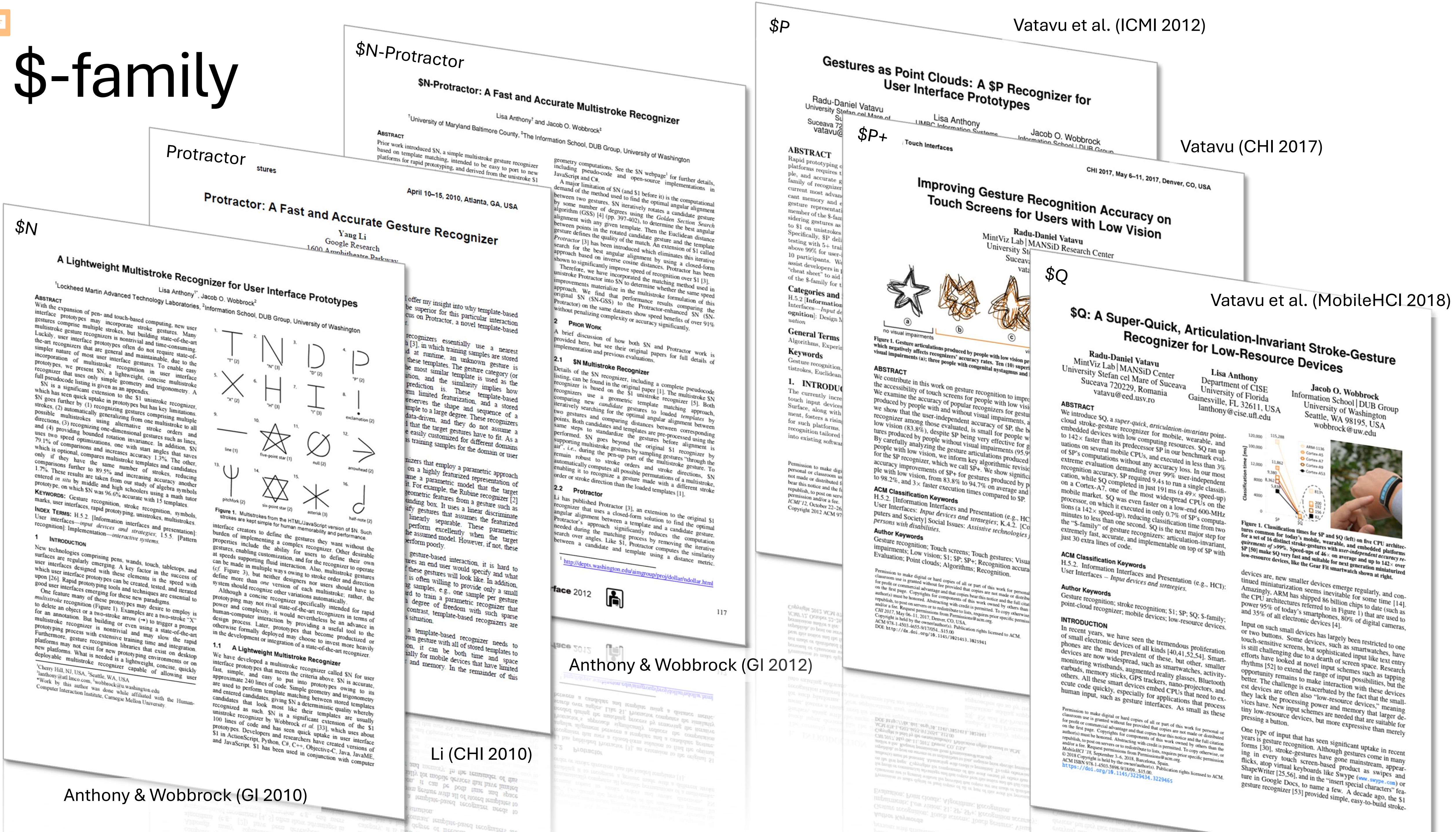
In the demo below, only one unistroke template is loaded for each of the 16 gesture types. You can add additional unistrokes as you wish, and even define your own custom unistrokes.



Make strokes on this canvas. If a misrecognition occurs, add the misrecognized unistroke as an example of the intended gesture.

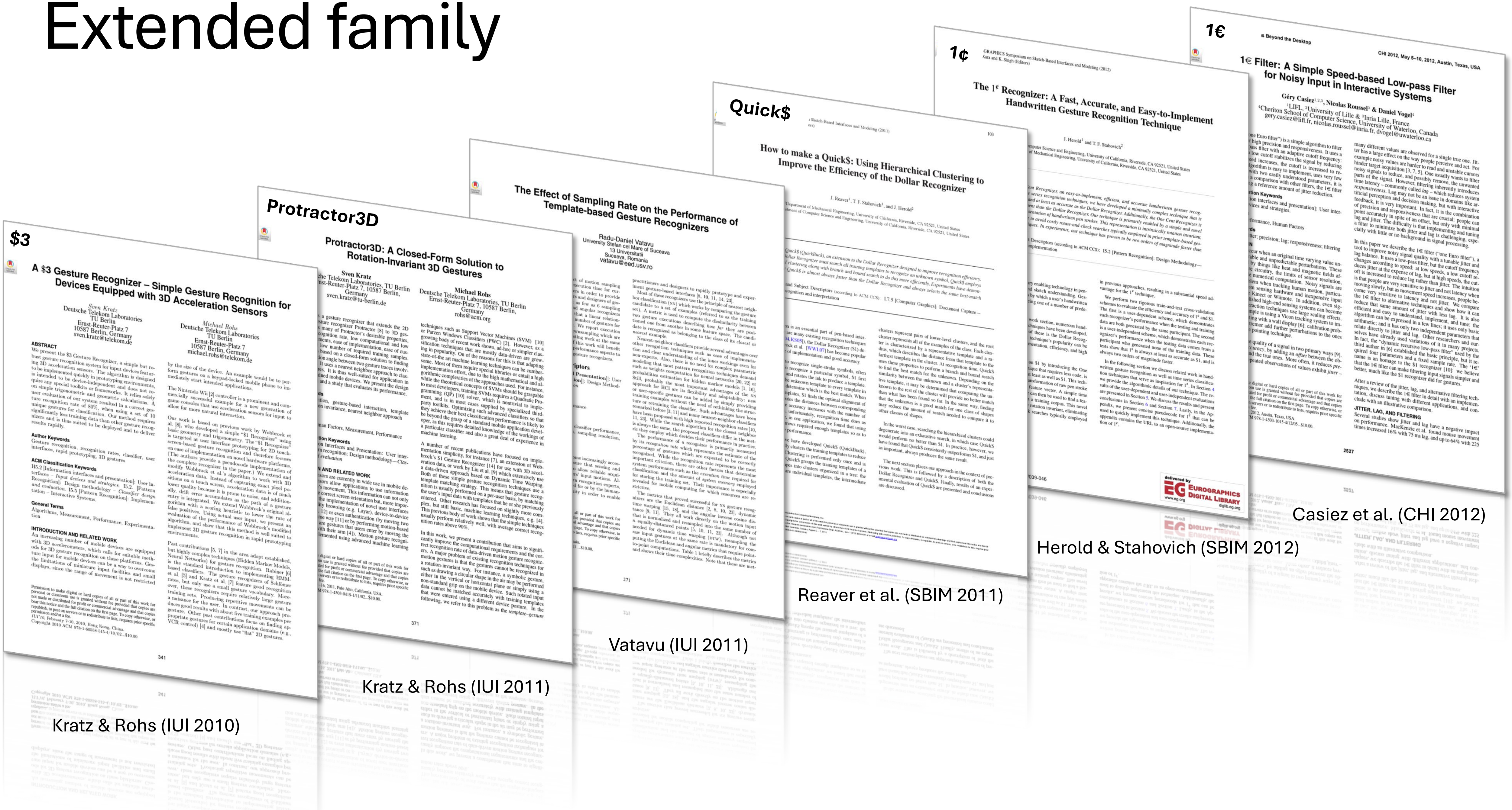
https://depts.washington.edu/acelab/proj/dollar/

$-family

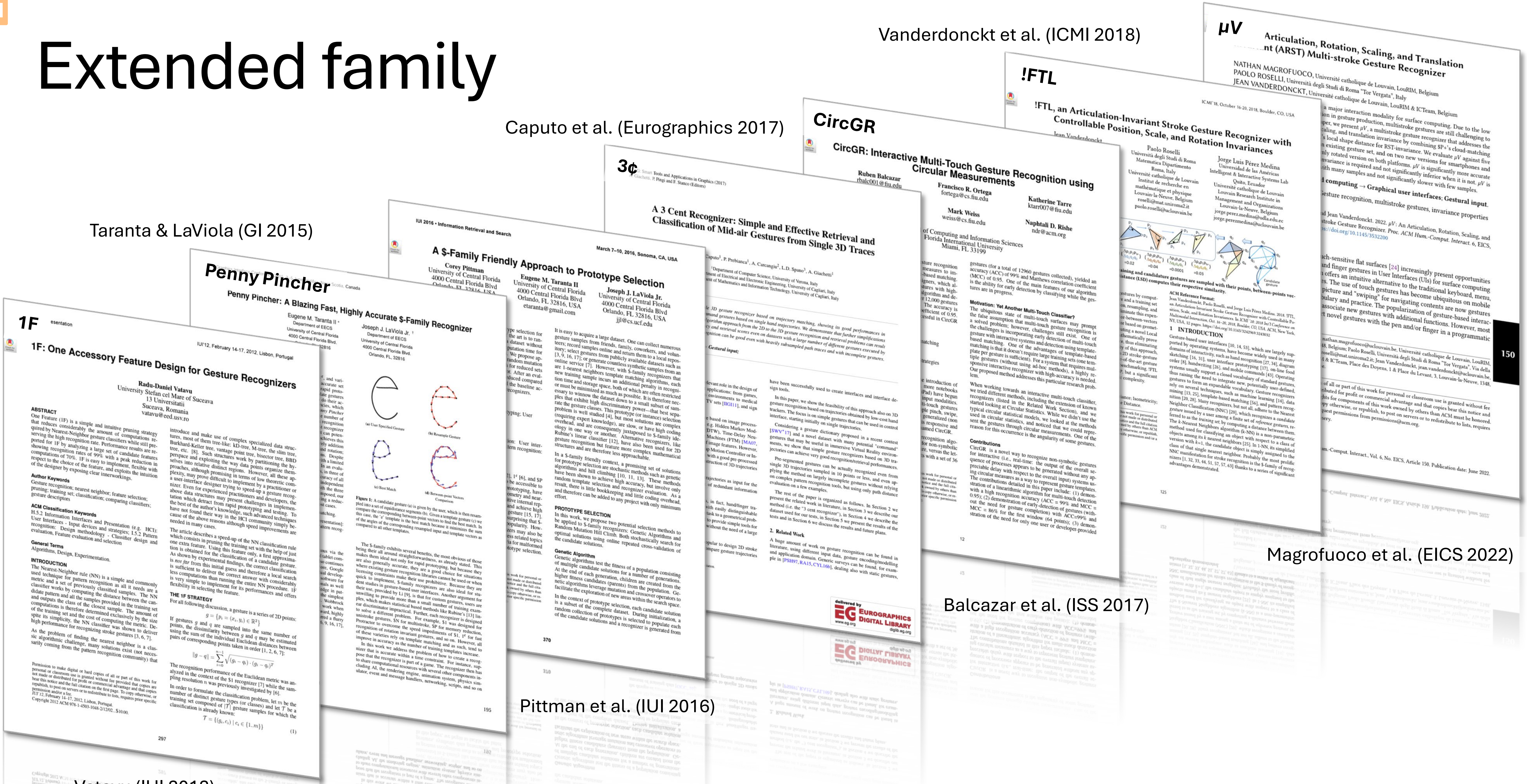


$N
A Lightweight Multistroke Recognizer for User Interface Prototypes
Anthony & Wobbrock (GI 2010)
Protractor
Protractor: A Fast and Accurate Gesture Recognizer
Li (CHI 2010)
$N-Protractor
$N-Protractor: A Fast and Accurate Multistroke Recognizer
Anthony & Wobbrock (GI 2012)
$P
Gestures as Point Clouds: A $P Recognizer for User Interface Prototypes
Vatavu et al. (ICMI 2012)
$P+
Improving Gesture Recognition Accuracy on Touch Screens for Users with Low Vision
Vatavu (CHI 2017)
$Q
$Q: A Super-Quick, Articulation-Invariant Stroke-Gesture Recognizer for Low-Resource Devices
Vatavu et al. (MobileHCI 2018)

Extended family



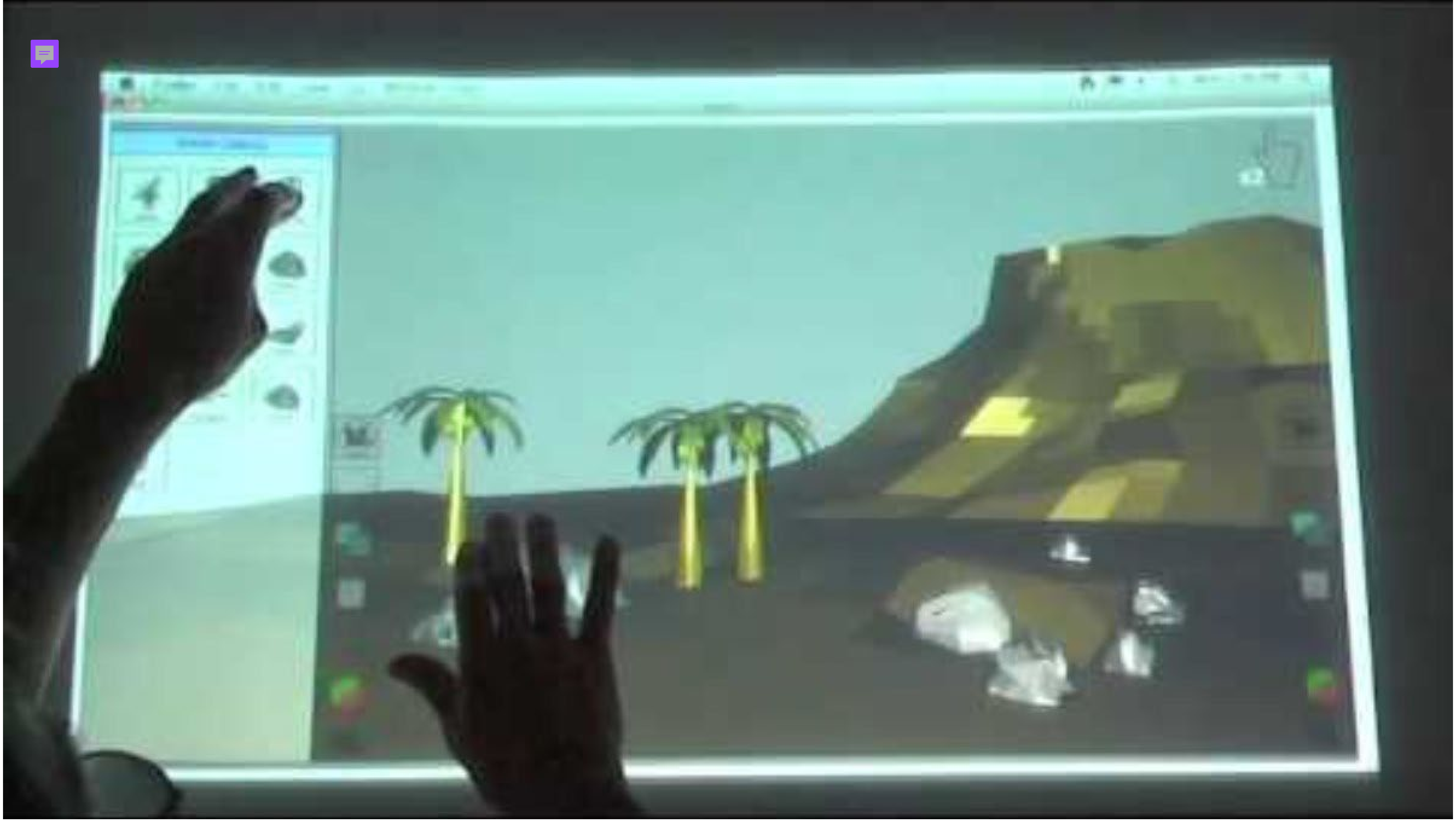
Extended family



Other people's projects…

Mr. Spiff's Revenge

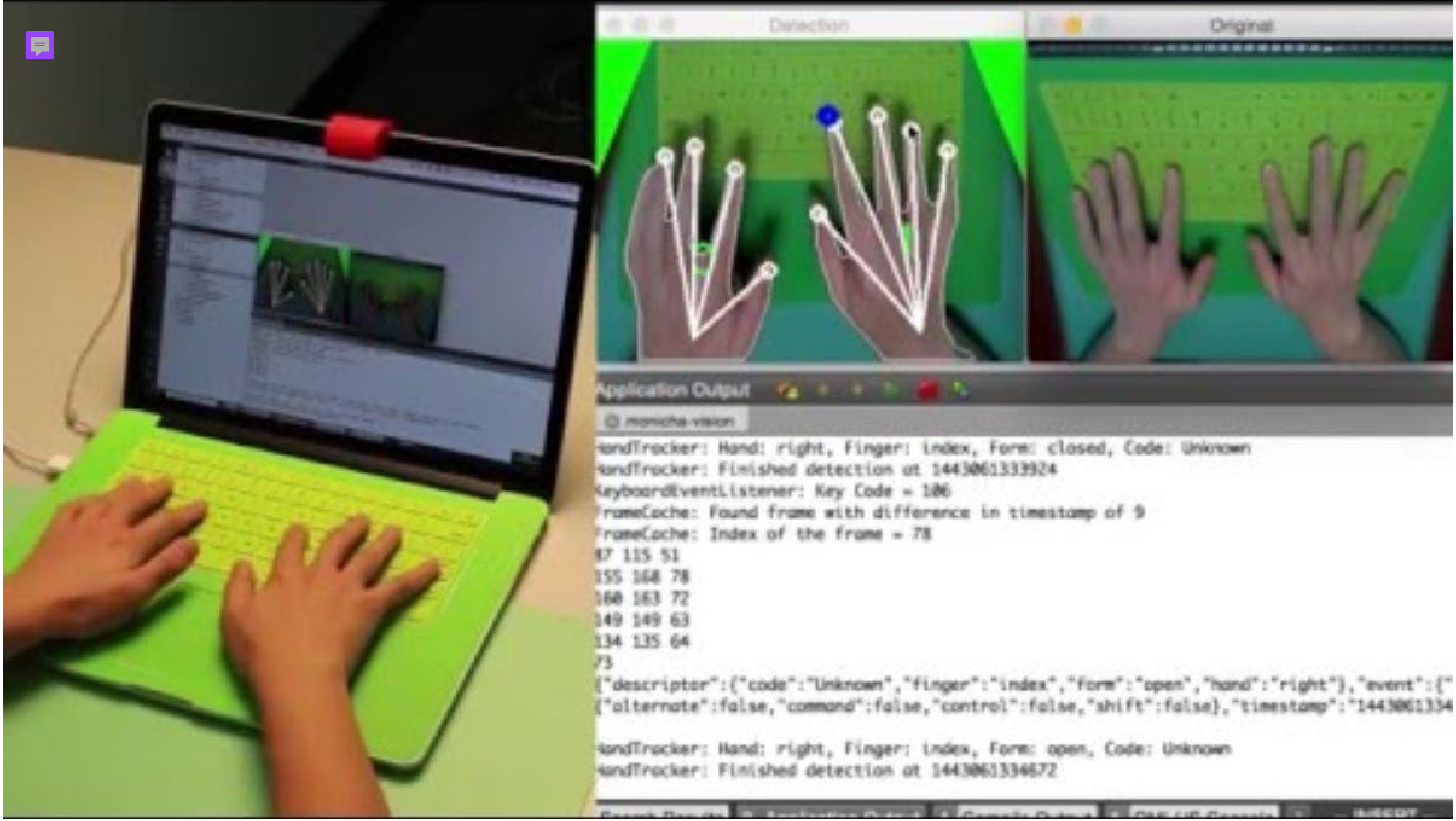
(POW Studios 2008)



Eden by Kin, Miller, Bollensdorff, DeRose, Hartmann & Agrawala (CHI 2011)

Air+Touch by Chen, Schwarz, Harrison, Mankoff & Hudson (UIST 2014)

Expanding the input expressivity of smartwatches… by Xiao, Laput & Harrison (CHI 2014)

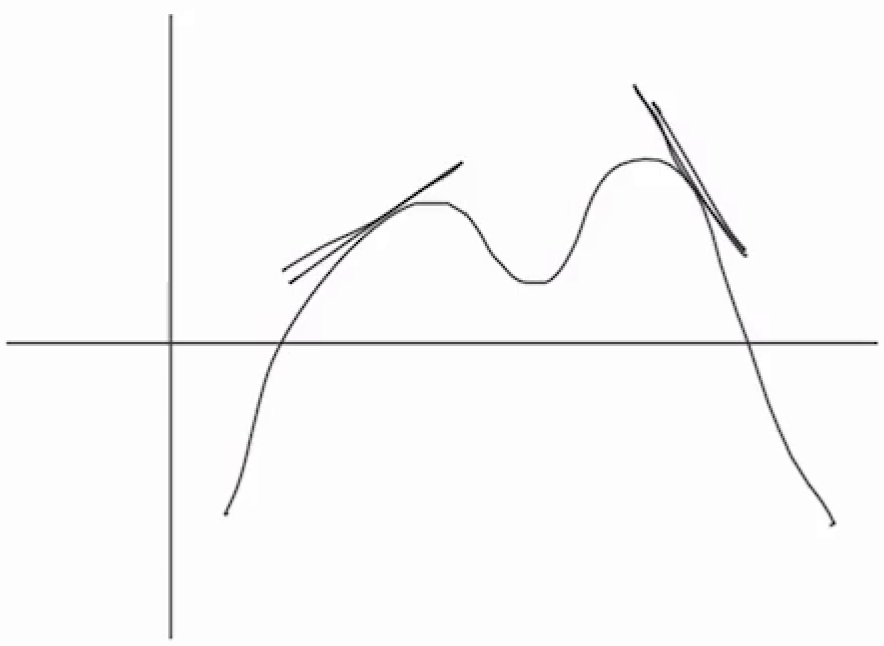


Finger-aware shortcuts by Zheng & Vogel (CHI 2016)

Inkbase

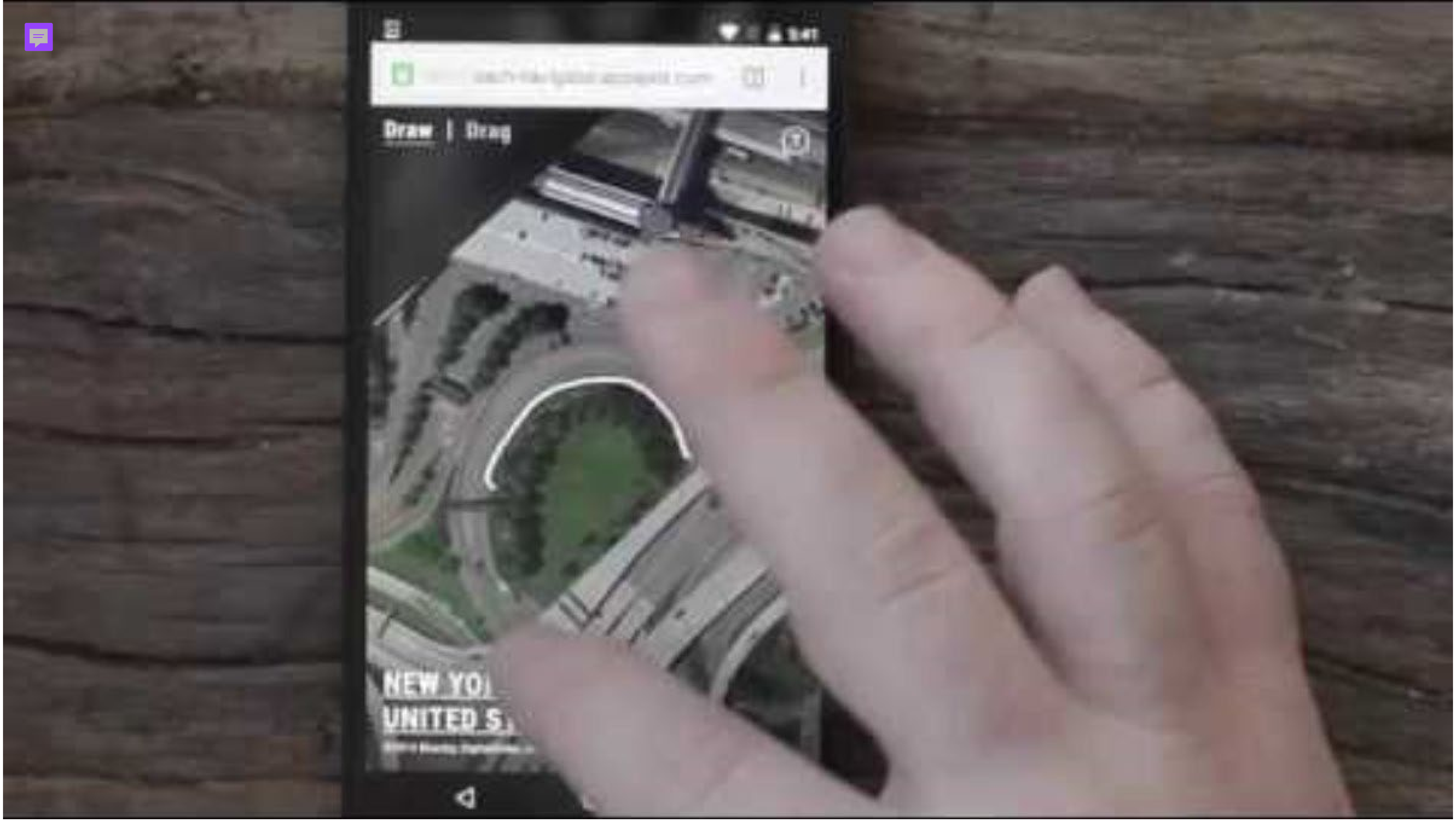
Programmable Ink

James Lindenbaum
Szymon Kaliski
Joshua Horowitz
November 2022



Sketchy math example using the $1 Unistroke Recognizer to identify a triangle.

https://www.inkandswitch.com/inkbase/



Land lines by Lieberman & Felsen (2016) https://lines.chromeexperiments.com/

Takeaways

- UIST is definitely <u>the</u> place for **novel** UI inventions!
- But it also has a long history of contributions that make known things **easier** to do (e.g., UI toolkits)
- For maximum impact, we must continue to prioritize **both!**
- This means:
  - Make your contributions easy to take up and use by others
  - Enable others to replicate and test your innovations
  - “White box” your prototypes and algorithms
  - Think beyond your UIST publication!

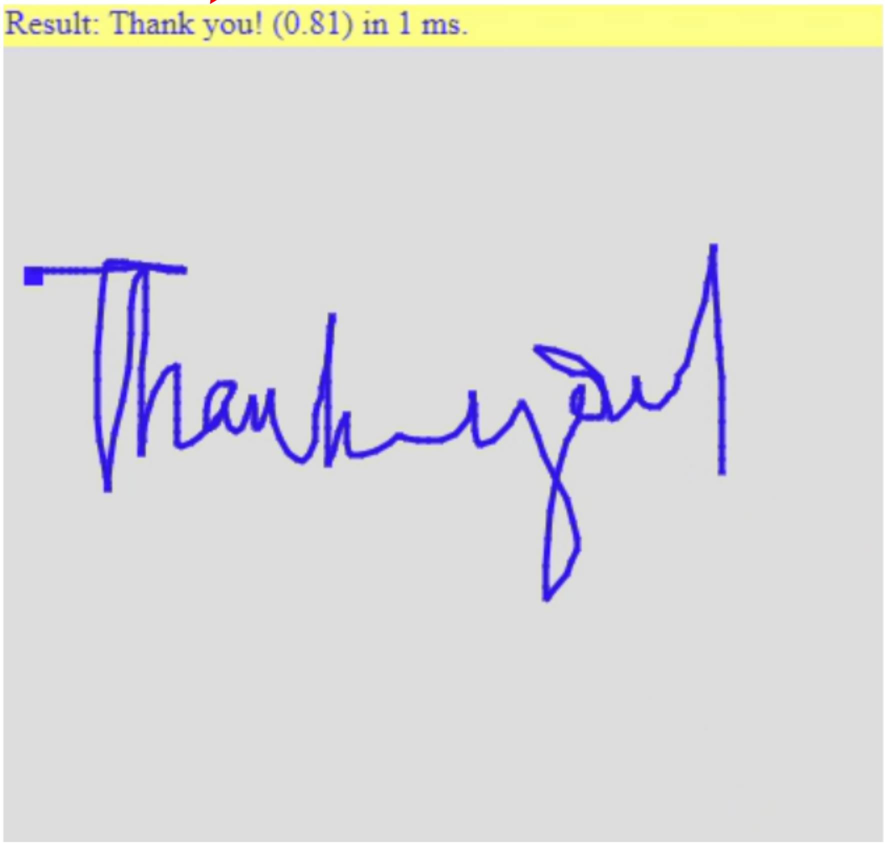
(Yes, some of these are in tension with commercial priorities)

Thank you!

Jacob O. Wobbrock
University of Washington
wobbrock@uw.edu

Andrew D. Wilson
Microsoft Research
awilson@microsoft.com

Yang Li
University of Washington
(Now at Google DeepMind)
liyang@google.com



https://depts.washington.edu/acelab/proj/dollar/