Foundations and Trends® in
Human–Computer Interaction
Vol. 5, No. 2 (2011) 97–205

DOI: 10.1561/1100000012

now
the essence of knowledge

Foundational Issues in Touch-Surface Stroke Gesture Design — An Integrative Review

By Shumin Zhai, Per Ola Kristensson,
Caroline Appert, Tue Haste Anderson,
and Xiang Cao

Contents

Foundations and Trends® in
Human–Computer Interaction
Vol. 5, No. 2 (2011) 97–205

DOI: 10.1561/1100000012

now
the essence of knowledge

Foundational Issues in Touch-Surface Stroke Gesture Design — An Integrative Review

Shumin Zhai[1], Per Ola Kristensson[2], Caroline Appert[3], Tue Haste Anderson[4], and Xiang Cao[5]

[1] *Google, USA, zhai@acm.org*
[2] *University of St Andrews, UK, kristensson@acm.org*
[3] *University of Paris-Sud & CNRS, France, appert@lri.fr*
[4] *frog design, Milan, Italy, tuehaste@gmail.com*
[5] *Microsoft Research Asia, China, xiangc@microsoft.com*

Abstract

The potential for using stroke gestures to enter, retrieve and select commands and text has been recently unleashed by the popularity of touchscreen devices. This monograph provides a state-of-the-art integrative review of a body of human–computer interaction research on stroke gestures. It begins with an analysis of the design dimensions of stroke gestures as an interaction medium. The analysis classifies gestures into analogue versus abstract gestures, gestures for commands versus for symbols, gestures with different orders of complexity, visual-spatial dependent and independent gestures, and finger versus stylus drawn gestures. Gesture interfaces such as the iOS interface, the Graffiti text entry method for Palm devices, marking menus, and the

SHARK/ShapeWriter word-gesture keyboard, make different choices in this multi-dimensional design space.

The main body of this work consists of reviewing and synthesizing some of the foundational studies in the literature on stroke gesture interaction, particularly those done by the authors in the last decade. The human performance factors covered include motor control complexity, visual and auditory feedback, and human memory capabilities in dealing with gestures. Based on these foundational studies this review presents a set of design principles for creating stroke gesture interfaces. These include making gestures analogous to physical effects or cultural conventions, keeping gestures simple and distinct, defining stroke gestures systematically, making them self-revealing, supporting appropriate levels of chunking, and facilitating progress from visually guided performance to recall-driven performance. The overall theme is on making learning gestures easier while designing for long-term efficiency. Important system implementation issues of stroke gesture interfaces such as gesture recognition algorithms and gesture design toolkits are also covered in this review. The monograph ends with a few call-to-action research topics.

1

Introduction

The advent of a new generation of touchscreen smartphones and tablets is rapidly transforming the everyday computing experience of the masses. It is also shaping and changing research questions and priorities within the human–computer interaction research community. One research question is how to exploit the continuous stroke gesture capabilities that had not been previously available to most users on keyboard and mouse-based desktop and laptop computers.

In fact, stroke gesture research has a long history in the human–computer interaction (HCI) research field. Sketchpad, an early project commonly recognized as one of the beginnings of HCI research, was centered on graphical human–machine communication through stroke gestures [108]. About a decade later the influential textbook by Newman and Sproull [91] prominently featured stroke gestures as an input mechanism and described in detail how to implement a rudimentary stroke gestures recognizer. Buxton offered early insights into the cognitive functions, such as "chunking," that gestures may play in interaction [17]. Since the 1990s, stroke gestures as an interaction modality have been explored in a wide range of research prototypes for different application domains in the HCI research literature [25, 95, 128].

While stroke gestures have been continually explored in the HCI research literature, they only played a marginal role in HCI practice during the personal computing revolution in the 1980s and 1990s. The mouse and keyboard driven point-and-click style of Graphical User Interfaces (GUI) interfaces, also referred to as WIMP (Windows, Icons, Mouse, Pointer) interfaces, have been the dominant HCI paradigm in office and home computing for decades. The vision of using stroke gestures as an alternative way of interacting with computers, however, never ceased to exist in the research literature or in industrial and commercial efforts by companies and projects such as GO, the Apple Newton, the Palm Pilot, and the Windows Tablet. The spirit of a visionary entrepreneur in this space is well captured by Jerry Kaplan's tale of GO as a start-up venture [54]. Financially backed by Silicon Valley's best known venture capitalist John Doer and his firm (KPCB), GO offered a compelling vision of changing the fundamental user experience with pen-based computing, but nonetheless failed to gain market traction. In 1984, Casio released a wristwatch, the DB-1000, with a touchscreen that enabled the user to enter names and phone numbers by drawing them on the watch's screen. Later the Palm Pilot featuring Graffiti, a single stroke Roman letter-like gesture writing system, pioneered the mobile PDA (personal digital assistant) market and made the gesture user interface (UI) more mainstream. The success of the Blackberry smartphones turned the trend toward touchscreen and gesture UIs back to physical keyboards. Before Apple's iPhone, touchscreen and stroke gesture-based products remained on the fringe of personal computing. There could be many reasons for the difficult expansion of user computing from the keyboard and pointing device paradigm. One possible factor is that hardware limitations at the time prevented ultra-mobile devices from offering a truly good user experience. The other could be that the WIMP interfaces were good enough for most users' needs at the time. The tendency for a "good enough" but sub-optimal technology to persist in society has been theorized as path dependency by economists such as Paul David [31], but debated by others [79].

Today, stroke gestures are becoming increasingly more relevant to mainstream popular products such as the touchscreen-based smartphones and tablets. This is due to both technology-push, the

development of new technology with little regard for current market demands, and market-pull, the market need guiding new technology development. Advances in hardware, including processing power, memory capacity, bandwidth, touch sensors, and battery technology have enabled handheld devices to provide a level of computing power only possible in desktop computers a few years ago. High-quality touchscreens have also begun to turn walls and tables into interactive computer media. In the marketplace, the value of directly manipulating objects with finger gestures on mobile devices and touch sensitive surfaces is suddenly being realized by consumers and manufacturers alike.

The stroke gestures used in today's popular touchscreen products tend to be relatively simple: sliding a document for panning; sliding a virtual latch to unlock; swiping across an item to delete. These simple gestures may be a solid foundation and the beginning of a new gesture interaction era. Technologies and paradigms that start with simple and incomplete functions often are more likely to succeed than those that start with complex but powerful functions. As users gain more experience with gesture-based interaction, they may be prepared for more complex and more powerful gesture functions in order to gain interaction efficiency. On the other hand, simplicity may well be the very reason for the success of the iPhone and other recent products. Researchers and designers have to consider the cost of learning and either minimize such cost or embed it in gradual use. Indeed, understanding and facilitating gesture learning is a theme of the current review.

Along with other colleagues in the HCI field, we have investigated various aspects of stroke gestures as an interaction medium. This monograph's primary goal is to provide a synthesis, summary, and interpretation of some of our research work on stroke gestures in the past decade, in a more accessible form and from a broader perspective than the original papers. Secondarily, we also selectively review some of the most fundamental behavioral research on stroke gestures by other researchers in the field. This monograph is not meant to be a complete review of stroke gesture research. The vast body of literature existing today on stroke gestures means that a complete review can easily turn into a catalogue or annotations of papers with no coherence or synthesis. We have

limited the scope of this monograph to stroke gestures as commands and symbols. For gestures in sketching applications, Johnson and colleague's "Computational Support for Sketching in Design: A Review" in *Foundation and Trends in Human–Computer Interaction* [51] provides an overview of the area. For early work on gesture research and commercial efforts, the reader is referred to the book chapter on "Marking Interfaces" by Buxton [16]. The reader is also referred to Norman and Nielsen's critique of the "crisis" of gestural interfaces [92], in which they address the many challenges in the current generation of gesture interfaces, particularly the lack of consistency and discoverability.

The overall goal of the monograph is to provide a scientific foundation for future research and design of stroke gesture interfaces. We have aimed for a high enough level of abstraction so that researchers and designers who are not necessarily specialized in gesture research can understand the contents easily, while maintaining enough details so that the main logic behind the development of the research issues is supported.

The rest of this monograph starts with basic concepts, terminologies, and classifications of stroke gestures. We then briefly review a few sample stroke gesture user interfaces and systems, to ground the subsequent reviews of the more general and more abstract issues in stroke gesture research and design. After a brief survey of some of the early basic usability research on stroke gestures as an interaction medium, we focus the main body of the review on some of the foundational human-performance issues concerning stroke gestures in HCI including:

1. Measuring and modeling the motor control complexity of one or a set of stroke gestures;
2. Visual similarities of gestures;
3. The effects of visual and auditory feedback on gesture strokes;
4. The cognitive and memory characteristics of using stroke gestures.

After that we switch the orientation of the review from conceptual and empirical understandings of stroke gestures to the dimensions, rules, and guidelines in designing gesture systems. The focus of

these sections is on how learning new stroke gestures can be facilitated in system design. We then deal with implementation issues in stroke gesture interfaces, including the separation of commands from scope selections, separation of stroke gestures from inking, the pros and cons of different gesture recognition algorithms, and toolkits that help programmers to design and implement stroke gestures in software applications. Finally, we summarize this monograph and raise a few key open questions, unanswered or under-explored in the research literature.

2

Definition, Concepts, Characteristics, and the Design Space of Stroke Gestures

In this section, we provide a more formal definition of stroke gestures. Thereafter we discuss the stroke gesture design space along five dimensions: analogue–abstract, command–text, order of complexity, visual-spatial dependency, and implement and sensor types.

2.1 Definition

We broadly define stroke gestures as the movement trajectories of a user's finger or stylus contact points with a touch sensitive surface. A stroke gesture can be produced by either a bare finger or an implement (a pen, also known as a stylus), although there are important differences, both in user behavior and in sensing mechanisms, between the two. A stroke gesture is usually encoded as a time-ordered sequence of two-dimensional (x, y) points. Optionally, stroke gestures can also have time stamps as the third dimension so the sampling points are encoded as (x, y, t) if the temporal aspect of a gesture is to be preserved and used. Furthermore, stroke gestures can have pressure (or, more correctly, surface contact force) as yet another dimension of control.

The pressure dimension can be used to control for example ink width, as a brush pen will do. There can be various creative ways of using this dimension although, in general, users' ability to control and maintain pen pressure on a surface tends to be quite limited in precision [99, 129]. On the other hand, many users have highly developed skills for manipulating the pressure of a stylus on a surface (such as in sketching) and digitizers with 512 or 1024 levels of pressure are often in demand by artists because of this preexisting skill.

The above definition of stroke gestures is limited to two-dimensional surface stroke gestures. Two-dimensional (2D) surface stroke gestures are a special, and especially powerful, case of hand gestures. Confining motions to 2D surfaces literally lowers the dimensionality of gestures. In comparison to gestures in three-dimensional space, stroke gestures on surfaces are easier to sense, record, process and recognize by both humans and machines, and yet still rich enough in the information they carry. In fact, all human symbols and writing systems, including the thousands of logographic Chinese characters, are products of stroke gestures. Stroke gestures are less ambiguous than free-hand gestures in three-dimensional (3D) space. An obvious but critical aspect of the ambiguity of hand gestures in 3D space lies in demarcation, also known as segmentation, which identifies the beginning and ending of a gesture expression. The beginning and ending of 2D stroke gestures are quite clear both to the machine sensor and to the human user's proprioception. Once the finger or stylus touches the surface, the user instantly feels that the stroke gesture has begun. This is a big advantage of stroke gestures. In contrast, it is more difficult to express a gesture in 3D space with a clearly defined beginning or ending, regardless if the user is using a bare hand or a wand. For such 3D gesture recognition, complex algorithms [87] are required to continuously track, then segment, usually imperfectly, user's hand or body movements in 3D space. In the case of games, however, much of the information on how to segment and interpret a 3D gesture can be derived from the script or dynamics of the game presented on the screen.

The nomenclature on touchscreen stroke gestures has not been consistent in the literature. In HCI they are often simply called gestures. However, this term is not ideal for two reasons. First, outside

HCI, gestures more commonly refer to whole hand and body gestures. Second, considering the clear difference between 2D stroke gestures and the more general hand gestures in 3D space, the word gesture is not the most appropriate for stroke-based interaction. However, since the term gesture has been the de facto standard choice in the research literature and in industrial practice, we continue to use it. Stroke gestures are sometimes also called pen gestures [82], hand drawn marks [128], hand drawn gestures [127] hand markings [39] or markings [63].

From an information theoretic point of view, stroke gestures are a form of two-dimensional geometric *signals* from the user's mind to the computer that *encode* text or commands (*messages*). They are transmitted through a *channel* with *noise* (due to inaccuracies in recall and production), received by the computer, and decoded by a recognizer into the messages intended by the user (Figure 2.1). The decoding process may take place at the end of the stroke or incrementally during the production of the stroke [8]. The capacity of this channel depends on how many messages can be accurately transmitted. This in turn depends on both the user's ability and the computer's algorithm to accurately classify different stroke gestures.

Even limiting "stroke gestures" to what is outlined above, there is still a vast conceptual space in which gestures may differ. We now focus on the major dimensions of gesture differences.

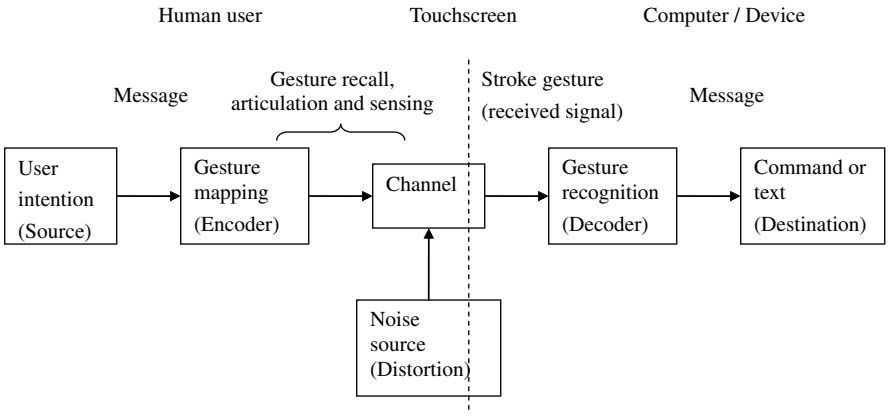


Fig. 2.1 Gesture input model from an information theoretic perspective.

2.2 Analogue versus Abstract Gestures

Analogue gestures are those that mimic the physical or conventional effects of the real world. An analogue gesture is analogous to what a stroke gesture would do in the physical world or according to a cultural convention. For example, a touch and slide gesture can cause a document to pan, as a physical paper document would. Immediate and animated feedback and "direct manipulation" (manipulation of an "object"), often with effects based on physical simulation, come into play when designing analogue gestures. On the other hand, abstract (or symbolic) gestures are fundamentally arbitrary gestures that do not resemble physical effects. For example, one may use an X sign to close a document.

The analogue–abstract classification is a spectrum, not a dichotomy. Gestures can be designed to resemble physical effects to a degree, often only partially. They can also be designed to resemble users' prior experiences or conventions in different ways. The degree to which a gesture is considered analogous can also be subjective and varies with an individual user's prior experience. For example, the question mark symbol must be quite abstract looking when it is first introduced to a writing system, but over time it looks more analogous to its semantics. For another example, a Chinese person often believes Chinese characters such as "laugh (笑)" versus "cry (哭)" and "walk (走)" versus "stand (站)" depict their meanings in a pictorial fashion, but when we tested those Chinese character pairs with non-Chinese speakers, their correct guess was no better than chance.

The more analogous gestures are to the user's prior experience, the easier they are to learn. On the other hand, the number of gestures that are highly analogous to physical effects or conventions in any given domain is limited. Analogous gestures may also be impractical if they are overly complicated or ambiguous when the same gesture has multiple possible effects and meanings. Abstract gestures free from mimicking real-world forms can be made simpler and more efficient to produce. For example, the Arabic numerals are abstract stroke gestures that represent quantities. The symbol "9" is faster to draw and easier to perceive than nine bars. Similarly, one may use a single finger hand

gesture for the number one, two fingers for the number two, and five fingers for the number five. Beyond that, it is difficult to make a gesture that is analogous to the actual quantity. In some cultures, more symbolic gestures are used to represent quantities greater than five (for example, a closed fist for the number ten).

The history of the Chinese writing system also reflects the tradeoff between analogous but complex gestures and abstract but simple gestures. Early forms of many Chinese characters were highly pictorial but over time they have been simplified and made more abstract to be more practical and more efficient.

One systematic middle design point on the analogue-symbolic spectrum is to leverage written language symbols. For example, the V-sign hand gesture with the palm outwards is often used for victory since Winston Churchill popularized it. For another example, drawing a letter *d* can be used as a gesture for a *delete* action. Such stroke gestures are fundamentally abstract yet familiar and conventional to the users. However this design approach is not without challenges since a single letter for an action can be too ambiguous. For example, the letter *p* can mean *print*, *paste*, *preview*, *page* or *picture*. To use a whole word, on the other hand, is inefficient. We will return to these issues later in the monograph.

In a somewhat different context, Wobbrock, Morris and Wilson [120] lay out a gesture taxonomy that has similarities and differences to the dimensions discussed in this monograph. In particular, its *nature* dimension describes gestures as abstract, symbolic, metaphorical, or physical, which captures similar characteristics as the current analogue–abstract dimension. We use the word abstract more broadly as a quality. It is opposite to "analogue" in our classification.

2.3 Command versus Symbol Gestures

Stroke gestures can be used to issue commands or to enter text, or more broadly, symbols and marks. Commands and text are two distinct types of information. Text is the written form of natural languages. Commands on the other hand are names of executable computing functions issued by the user. Commands tend to be more artificial (designed) and

less numerous than symbols and scripts (evolved, for the most part). A computing system may have dozens, hundreds, or perhaps thousands of commands. The number of words in a natural language, on the other hand, can be tens of thousands or more.

Stroke gesture interface research has dealt with both commands [25, 39, 63] and text [38, 96, 131]. "Natural" handwriting itself consists of gesture marks. Partly because most of the world's writing systems were not created for efficiency [28], inventing novel forms of text entry hence has been a very active topic of research [38, 59, 85, 96, 134].

Symbols and commands are also related in many ways. One can always use symbols to enter commands. Conversely, command systems can sometimes be extended for text entry. For example, marking menus, a gesture system primarily designed for commands [67], were later extended as a way of entering letters [115]. ShapeWriter (also known as SHARK) was initially designed as a gesture system for text entry [59, 134] but later extended to deal with commands [60].

2.4 Order of Complexity of Stroke Gestures

Gestures can also be classified according to their order of complexity. We may call touch points on a surface (without movement traces) zero-order stroke gestures. A simple soft button tap [72] is a zero-order gesture. Interestingly, landing points as zero-order gestures on a touch surface can still carry different information according to the orientation of the touching finger (Table 2.1) [46, 118].

We call gestures comprised of a single stroke first-order gestures. Unistroke gestures constitute the essential elements of all stroke gestures. They also have sufficiently broad applications by themselves. They are therefore the focus of the current review. Within the first-order stroke gesture class, the complexity of stroke gestures can still vary greatly. We will deal with the measurement and modeling of unistroke gesture complexity in the motor control complexity section later in this monograph.

We call gestures with multiple strokes, drawn either sequentially or in parallel (multi-touch), higher-order gestures. Parallel higher-order gestures drawn with multiple fingers or multiple hands may offer both

Table 2.1. The order of stroke gestures.

	Single contact point	Multiple contact points
Zero length stroke	Zero-order gestures	Zero-order multi-touch
Non-zero length stroke	First-order gestures (unistroke gestures)	Higher-order gestures

physical and cognitive advantages in certain applications. For example, the ability to grab two corners of a rectangle to rotate and move at the same time is both fast and compatible with how people naturally view this task [73]. However we limit the current review to first-order gestures because of their dominance in research and practice and because they form the building blocks of higher-order stroke gestures.

2.5 Visual-spatial Dependency

A critical design dimension of stroke gestures is whether and to what degree the effects of the gestures are dependent on the location and size at which they are drawn. When a gesture's meaning does not change with its location or scale, it is visual-spatially independent, or location-and-scale dependent.

There are three types of advantages to visual-spatial independency. First, it saves screen space. Consider the traditional point-and-click style of graphical interfaces where icons or menu items are laid out spatially, and each item takes up a certain amount of space. On the other hand, different visual-spatially independent gestures can be drawn in the same space. Second, with visual-spatially dependent user interfaces, the user first needs to navigate, by visual search, scrolling, panning, flipping (to a different screen), zooming or clicking (in the case of hierarchical menus), to a particular object before activating it. In contrast, visual-spatial independent gestures can be drawn anywhere to access different functions directly without navigation. To borrow from computer memory jargon, they become a "random access medium (RAM)."

Another advantage of visual-spatially independent gestures lies in their reduced visual attention demand. Since they are location and scale independent and the information is all expressed in the gesture's relative shape itself, they help support head-up use, which is very important in ultra-mobile situations.

On the other hand, visual-spatially dependent gestures have many of their own advantages. First, they enable action-object coupling. The action can be expressed by the gesture shape, but the object the action applies to is specified by where the gesture is drawn. For example, an X-shaped gesture on an email might delete that specific email. So the X is serving double duty: it selects the email and specifies what to do with it in a single action. Such action-object coupling increases user efficiency, although at the cost of visual attention demand.

Second, visual-spatial dependency can be used for disambiguation. For example, in the early Palm Pilot Personal Digital Assistants (PDAs), text and digits are written into different boxes to avoid ambiguity. For another example, the ShapeWriter gesture keyboard uses gestures both for words and for commands, but "gesture commands" in ShapeWriter have to start from a special key (e.g. a Cmd key) in order to be separated from word gestures.

Another form of visual-spatial dependency is object-context dependency. If designed to be object-context dependent, the same gestures can have a different meaning in different applications and on different objects. For example, a straight-line movement can mean dragging an icon, flipping a page, or deleting text, depending on which application and what function the user *navigates* to and onto which object the user applies the gesture.

Crossing-based interfaces are a form of a visual-spatial dependent gesture interface. Gestures such as a straight line, a zigzag line, or a circle do not carry much information. When used to cross different graphical objects, however, they can in principle perform all actions needed in interactive software design [4, 7].

In general, if gestures are designed to be visual-spatial dependent, much can be gained in information capacity and understandability, since part of the information transmitted when using these gestures can come from the graphical objects on the screen, rather than the shape of the gesture alone. Of course, visual-spatially dependent gestures take valuable screen space, necessitate navigation, and require user's visual attention.

Visual-spatial dependency is also a continuum. Most visual-spatially independent gestures are still restricted to a certain area of the

screen and most visual-spatially dependent gestures, potentially even crossing-based interfaces, can be relaxed according to the amount of "white space" available in the task. As a simple example, a scroll gesture must start on a scrollbar to distinguish itself from other actions achieved via dragging but it can continue outside the bar. More importantly visual-spatial dependency can also be changed dynamically according to the speed of the gesture and the level of user experience, as we will see when we review marking menus and the ShapeWriter gesture keyboard later in this monograph.

Visual-spatial independency and gesture complexity are closely related. Simple gestures needs to be visual-spatially dependent to carry rich meanings. At the extreme, tapping (a zero-order gesture) has to be completely visual-spatially dependent to carry more than one bit (on–off) of information. It can activate any number of applications on a smartphone because the activation of these applications depends on where (or which icon) the user taps. In contrast, more complex gestures can carry different meanings independent of where the gestures are drawn.

2.6 Implement and Sensor Type

Early gesture systems have focused on the digital stylus as the implement of articulation, to the extent that gesture interfaces were synonymous with pen-based interfaces. Stylus drawing involves more joints of the hand and offers higher precision and dexterity than single finger drawing. However, since the iPhone, more recent developments in commercial products have shifted stroke gestures away from stylus drawing to finger drawing. With a finger-based gesture system there is no device acquisition time (the time to pull out and grasp the pen) so it is more direct and more convenient to use. One certainly does not have to deal with the problem of losing the stylus. On the other hand, the finger obscures more screen surface and is less dexterous and less precise than a stylus. See Hinckley et al. [44] for an example of how one system can combine both.

Tu, Ren and Zhai [111] studied the differences and similarities between finger and pen drawn gestures. They found that finger drawn

gestures tended to be larger and faster than pen drawn gestures. Finger and pen drawn gestures also differed in shape geometry as measured by aperture of closed gestures and corner shape. Pen drawn gestures and finger drawn gestures were similar in articulation time, indicative angle, axial symmetry and proportional shape distance. Their findings suggest that "finger friendly" systems should exploit global features such as proportional shape distance when designing finger interfaces and avoid local and detailed features.

It is possible to enable both finger and stylus operation on the same device so that, for a small amount of imprecise gesturing, the hurried user should be able to gesture with a bare finger. When more prolonged and more precise gestures are needed in applications such as note taking or mark-up of digital documents, one should be able to switch to a stylus. Whether a touchscreen will react to both finger and stylus input depends on the type of touch sensor embedded. Capacitive sensors, such as those in today's iOS (iPhone/iPod/iPad) and most Android phones, work only with fingers since they measure the capacitance change on the surface. Special styli with a large enough surface area, rather than a pointing tip, have to be used in order to work with capacitive sensors. However, a stylus without a sharp pointing tip loses much of its advantage.

Inductive sensors, such as the Wacom sensors embedded in most Windows Tablet PCs, work with digital styli in order to measure inductance change. One advantage of inductive sensors is that they can sense the stylus when it is near the sensing surface. This is sometimes referred to as the hover state. See Buxton [18] for a more complete description of this three-input state model. Analogous to moving a mouse before clicking a button to trigger a command, the hover state enables systems to separate the action of moving a pointer on the screen from the action of triggering a command.

Resistive sensors, such as those found in Palm Pilot PDAs and old Windows Pocket PCs, measure change of electrical resistance at the contact point of the touch-sensing surface where two layers of membrane are pressed together. This works better with a stylus since the stylus tip on the surface forms a well-defined pressure point. It works less well with fingers, particularly for stroke gestures. This is because

when the user articulates a stroke gesture over a resistive sensor the user usually has to maintain a certain force-level to keep the sensor activated. This task is particularly hard when the user is using a finger's soft tissue.

Touchscreen devices may also use optical or other types of sensors that react to both stylus and finger touch, as used for example in Microsoft Surface and the HP TouchSmart desktop computer. Some recent PC products, such as the Lenovo X220 Tablet, embed both pen and finger touch-sensitive layers in their screen so the user can switch between pen and finger gestures. An interesting design space to enable is the simultaneous use of the finger and the pen, with the pen in the dominant hand for writing and drawing, and fingers for object manipulation [44].

In summary, there are at least five dimensions of stroke gesture differentiation: analogue–abstract, order of complexity, commands versus symbols, degree of visual-spatial dependency, and implement and sensor type. These are just a few of the most important dimensions relevant to this review. For the benefit of conceptual clarity and experimental rigor, research tends to focus on one or two dimensions at a time. However, when designing a practical system, it is important to realize that all these dimensions need be considered in the context of the total system design. Often these dimensions trade off against one another and designers are faced with the difficult task of comprising and prioritizing certain dimensions for a particular application domain.

3

Sample Gesture Systems

In order to better ground the conceptual issues to be reviewed later in the monograph, this section briefly reviews a few concrete examples of stroke gesture user interfaces.

3.1 The iPhone Gesture Interface

Most of the iPhone system-wide user interaction is through soft button tapping, or zero-order gestures. There are only a few first-order stroke gestures: slide to unlock (Figure 3.1), a straight stroke to scroll and pan a document or a map, flip to the next photo, or invoke a soft button (such as "delete" in case of a list of items). There is also one second-order gesture, "pinch," for zooming. Yet another type of zero-order but more complex gesture is the "long press." Press-and-hold for a set period of time often is used as a way of invoking a menu for more options.

Although small in number, these iPhone gestures, particularly scrolling and panning, are used frequently, and hence are critical for the iPhone's overall user experience. The scrolling and panning gestures are analogue gestures mimicking physical effects, therefore quite easy to discover.

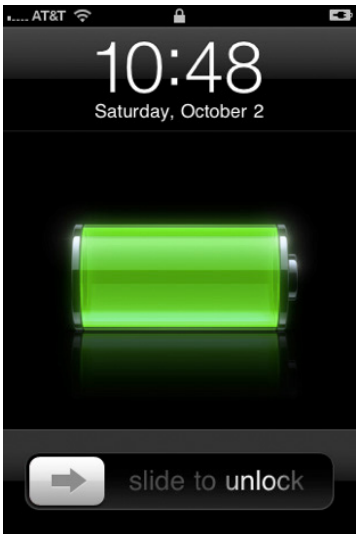


Fig. 3.1 A slide gesture unlocks iPhone's cover screen.

The small set of simple gestures does limit the number of functions enabled and often precludes other stroke gestures. For example, a simple swiping gesture is typically used for scrolling/panning/flipping, but it precludes the ability to for example drag an object and drop it into a folder or a document. The swiping gesture for the unlock action would not have been as obvious had it not been aided by an arrowhead and animation as a "preview" of the gesture effect. On the iPhone, the user can swipe over an item in a list, such as a contact or an email, among a list of many, to invoke a menu option (e.g., delete). This is a more abstract and less discoverable gesture.

Another interesting gesture design choice on the iPhone is the use of temporally and spatially dependent stroke gestures for the on-screen keyboard. If the user presses a letter key and pauses for one second, an additional set of letters pops up and enables the user to slide to one of the alternative letters (e.g., if the user taps-and-holds the letter key *s* then the system proposes ß). This scheme allows a greater number of characters with different diacritical marks. The downside is that this feature is not easily discoverable by the casual user.

3.2 Graffiti and Unistrokes

As simple as they may seem, Roman characters are not really easy for humans to write nor are they easy for computers to accurately recognize, for a number of reasons. Because each person has his own way of shaping and joining letters, there is a high degree of variability in natural handwriting. Furthermore, the letters f, i, j, t, x, consist of two strokes. As any second-order gestures, these letters create two major disadvantages. First, they complicate the demarcation (or segmentation) of the beginning and ending of a character. The system would have to use other spatial and temporal measures to determine whether the production of a letter is complete. Second, it takes more time to produce two strokes than one, and more importantly, since the second stroke is referential to the first stroke ("dotting the i's and crossing the t's"), it demands more visual attention.

Given this background, it is not surprising that attempts have been made to design new ways of writing the Roman alphabets. In fact alternative forms of the alphabets have been repeatedly created in history dating back to Roman times. A relatively recent attempt at inventing an English alphabet is the Shavian script (named after George Bernard Shaw). The challenge to novel alphabets, however, has always been the difficulty of switching away from what millions of people have already learned and practiced.

Motivated by the text entry challenge in pen-based computing, Graffiti and Unistrokes are two attempts at alternative forms of the Roman alphabets. Graffiti was a central feature in the early versions of the Palm Pilot, a device that played a leading role in launching PDAs, which have evolved into smartphones today. Unistrokes was the result of a research project at Xerox PARC. Its creators, Goldberg and Richardson, not only made an invention but also articulated a set of compelling motivations and principles for single-stroke gesture design [38]. Graffiti and Unistrokes made different choices in the degree of novelty and optimization. Graffiti stayed quite close to the standard form of the Roman alphabet (Figure 3.2). It is essentially a simplification of the Roman alphabet. Unistrokes on the other hand offered much

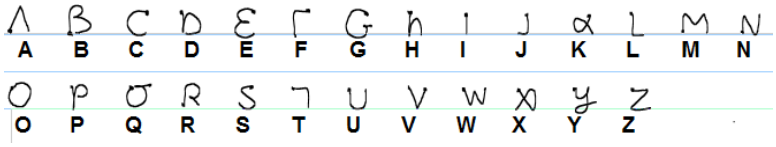


Fig. 3.2 Graffiti symbols resemble the standard Roman letters.

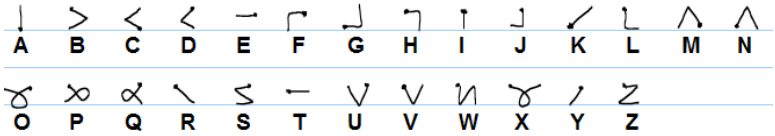


Fig. 3.3 Unistrokes symbols are more optimized.

greater potential in efficiency by designing completely new symbols that had little resemblance to their standard Roman letter counterparts (Figure 3.3). Users of Unistrokes would have to make a greater effort in learning and memorizing these new symbols than Graffiti users. From the two designs, we see the tradeoff between a set of gestures more analogous to people's existing practice and a more abstract set of gestures. We will review the efficiency difference between the two systems in more depth later in the gesture complexity section of this monograph.

3.3 Crossing UI

As mentioned earlier, object crossing is a strongly location dependent form of stroke gestures. The gestures used for crossing can be as simple as unidirectional swiping strokes. The effect of the crossing gesture depends on the object crossed. Potentially all buttons that are traditionally activated by a clicking action can be replaced by objects that accept crossing as an activation mechanism. The popular notification bar on Android phones is activated by a downward crossing gesture.

Crossing tasks can be well modeled just as pointing tasks can be well modeled [4]. According to the models established by Accot and Zhai [4], the time to cross a goal can be reliably predicted based on the goal width and the distance to reach the goal. The gestures used to cross an object can also be more complex and richer in semantics than a simple unidirectional swiping gesture. For example, a system

may assign a forward and backward swiping gesture to have different effects (e.g., open and close).

Crossing-based gesture interfaces can be built to implement various sophisticated functions. For example, Apitz and Guimbretière built an entire drawing prototype based on crossing gestures [7]. The EdgeWrite text input method [124] can also be viewed as a crossing interface, as crossing in and out of each of its four corners determines the characters entered. Crossing provided the theoretical basis for modeling the trackball version of EdgeWrite, which has no physical edges [122]. Interestingly, the version of EdgeWrite with physical edges can reduce or remove the need for visual dependency that is typically required of crossing interfaces.

3.4 Marking Menus

Although still not widely practiced in mainstream products nearly 20 years after their initial publications, marking menus or self-revealing pie menus [20, 48, 63, 64] were probably the first gesture UI designs that embodied the important goal and principle of facilitating learning, or novice to expert transition, that was well articulated by Kurtenbach and Buxton [64].

A marking menu is a radial menu with two explicit modes based on the user's action speed (Figure 3.4). The menus are only displayed if the user pauses for a moment. Novice users wait for the menu display to pop up then make their gesture motion to select the target section. Expert users who are more familiar with the menu layout, however, do not have to wait for the menu display guidance to stroke ahead and make the same gesture mark. Importantly the two modes share the same motion for the same menu action. The delay in the menu pop-up may encourage the user to move to the expert mode. Marking menus work on radial menu layouts and use angle detection to determine what commands to execute. No statistical pattern recognition is involved.

Marking menus are especially robust when they contain up to eight items at each menu level. They can be nested with multiple levels, so the gestures are compound marks with multiple inflection points. Since the items selected in marking menus are determined by angular direction,

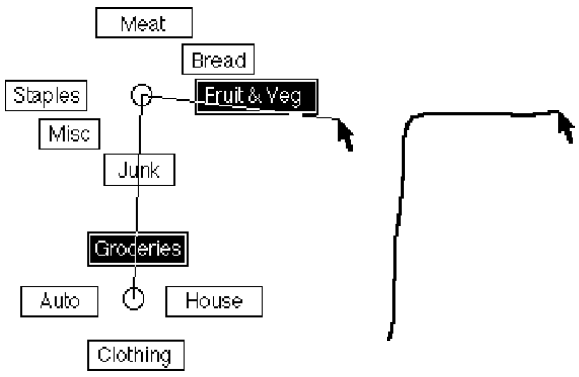


Fig. 3.4 Marking menus. Courtesy of Bill Buxton.

there can be ambiguities when multiple levels of selections happen to be aligned in the same direction (e.g., up and up). Simple hierarchical marking menus [139] address such a problem by selecting one level at a time with a straight stroke, so a multi-level selection consists of a series of straight strokes in different directions rather than a single stroke with multiple inflection points. Hotbox [65] increases the breadth of marking menus by either combining marking menus with other interaction techniques (e.g., a menu bar) or by differentiating items according to the absolute or relative location of the stroke's starting point. T-cube extended marking menus from commands to text entry [115].

3.5 Word-gesture Keyboard

A word-gesture keyboard uses gesture strokes on a touch sensitive keyboard, rather than tapping points, to enter text or commands. For each word in the lexicon, a stroke on the gesture keyboard that approximately connects all the letters in the word enters the word. For example, a stroke connecting *w-o-r-d* returns the word *word* (See Figures 3.5–3.7 for a few examples).

A word-gesture keyboard, such as SHARK [59, 134], has the following three essential components. The first is a keyboard capable of sensing gesture strokes. The keyboard also serves as a visual guide since each word gesture is defined by connecting the word letters on

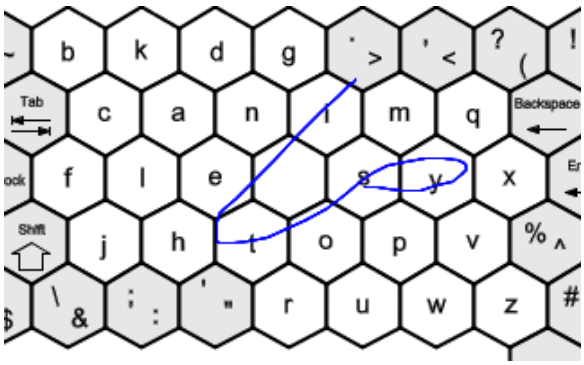


Fig. 3.5 A July 2003 screenshot of the SHARK word-gesture keyboard: the gesture represents the word "system" based on an ATOMIK layout.

Fig. 3.6 A screenshot of the SHARK word-gesture keyboard in 2005: the gesture represents the word *method* based on a QWERTY layout.

the keyboard. The second component is a language model that represents the regularities and constraints of the language that the keyboard is capable of expressing. Such a model can be a standard large list of N-grams or at the least a lexicon that constrains a finite number of permissible words. The third component is a pattern recognition system that classifies each user-entered stroke gesture into a ranked list of words according to the gesture's matching distance to these words and outputs the top-ranked candidate to a text field.

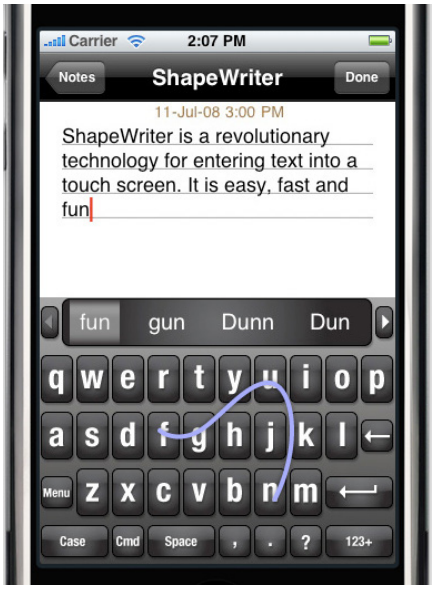


Fig. 3.7 A gesture keyboard application in the iPhone Store released by ShapeWriter Inc. in 2008. The gesture illustrated enters the word *fun*.

A gesture keyboard also supports the user's transition from performing visually guided actions to performing the same movement patterns by recall from memory. A beginner traces every word on the gesture keyboard by sliding from letter to letter by visually locating each letter key. Each time the same word trace is drawn, it is an opportunity to encode that trace in the user's procedural memory. For well-practiced words, or common fragments of words, the user will be able to recall their traces on the keyboard and hence rely more on memory recall and less on visual guidance, to more rapidly produce the gesture strokes. An experiment has shown that after on average 15 repetitions per word, users could reach completely recall-based gesturing [131]. We will more deeply review this experiment and other human-memory and skill acquisition aspects of gesture keyboards later.

Importantly, the recognition algorithm design of gesture keyboards should support both visually guided tracing and recall-based gesturing. In general, this means the recognizer should be highly error tolerant to imprecise productions of gesture details.

Unlike marking menus, a gesture keyboard does not have two distinct novice and expert modes. The transition from visually guided letter tracing to recall-based stroke gesturing is continuous. The actual user behavior is almost always a combination of the two components, shifting from the former to the latter as the user gains more experience. Also different from marking menus, gesture keyboards employ pattern recognition based on language models and handle a very large number (tens of thousands) of gesture entries.

SHARK (shorthand aided rapid keyboarding) was the first experimental word-gesture keyboard published in the literature [134]. Later the technology and paradigm were matured to a practical stage [59] and released to the public by IBM through the IBM AlphaWorks web site in November 2004. In 2007 it was extended to also handle commands [60]. SHARK was renamed ShapeWriter and commercialized and released on iPhone, Android and other platforms by ShapeWriter Inc [137] for English, French, German, Spanish, and Swedish. Since 2007, multiple companies have adopted the gesture keyboard paradigm with similar features.

There are many existing and potential algorithms to perform gesture keyboard recognition. SHARK/ShapeWriter primarily relies on two channels of gesture pattern recognition, one focusing on the location of the user drawn gesture, from beginning to end, in relation to the candidate word letters on the keyboard, and the other on the overall shape of the gesture, independent of the gesture's location and scale. The details of this recognition algorithm have been presented in [59]. We will review some of its key components later. One can also find additional information in Zhai and Kristensson [135] regarding ShapeWriter's design principles. Zhai and Kristensson [136] provides a retrospective synthesis of the SHARK/ShapeWriter project.

It is quite possible to develop much more flexible and more error-tolerant gesture keyboards in the future. A key design question still unresolved is the gesture keyboard's layout, be it the familiar QWERTY layout, or the more efficient ATOMIK [133], or some other layout specifically optimized for gesture keyboards. This choice again poses the difficult trade off between short-term ease of learning and long-term performance.

4

The Usability of Stroke Gesture as an Interaction Medium: Early Research Issues

This section reviews some of the early, basic and empirical studies on why stroke gestures are a usable interaction medium. Serious attempts at understanding stroke gestures as an interaction medium began shortly after the emergence of graphical user interfaces (GUI). Gesture interfaces were expected to be even more direct and more natural than the point-and-click style of direct manipulation [127]. The most comprehensive empirical work was probably done by Gould and Salaun [39] and Wolf and Morrel-Samuels [126, 127], both from the IBM T. J. Watson Research Center. These studies attempted to address a set of basic questions concerning stroke gestures as a control medium for common user computing tasks, particularly text and graphical editing.

One of their central questions was: How consistent are different users in creating or recalling stroke gestures for the same commands such as move, delete, and insert? Both the Gould and Salaun study and the Wolf and Morrel-Samuels study used paper and pencil tasks and asked participants to draw marks on paper to express commands such as delete, move, and replace. Both found a high degree of consistency in users' choice of text or graphics editing marks. People tend

to use a circle for "scoping," an arrow for moving, an arrowhead for destination and an X mark for deleting. The high degree of consistency was considered strong support that gesture marks were a "natural" way of interaction. With little training, users were expected be able to use these gestures in a consistent manner.

Recently, Wobbrock et al. [120] studied user agreement in their creation of gestures for tabletop computers. They asked 20 participants who did not have prior experience with gesture interfaces to intuitively come up with gestures for 27 functions (*referents*). They used a computational score of agreement to measure cross-user consistency in gesture expectation. Fewer unique gestures used for the same function by a greater number of participants yielded a higher agreement score for the function. If all users used the same gesture for the function, it received a score of 1. They found 7 functions had higher than 40% agreement scores, the rest less than 40%. Functions *move a little* and *move a lot* received a score of 1; *duplicate, rotate, selection group*, *insert*, and *select single* received a score between 0.6 and 0.4 for single hand gestures. Functions such as *undo, accept, help, reject, next*, and *cut* received less than 0.2.

There can be several counter-arguments and caveats against the natural consistency principle in gesture user interfaces. First, the types of tasks and functions that received high agreement and consistency in these studies were spatial, and the actions commanded were very similar to their physical effects. In other words, these were analogue gestures. It could be the affordances (in the original sense, as by Gibson [37]) in the experimental tasks, such as moving an element from one location to another, that required the participants to draw highly consistent gesture marks. In fact, Experiment 5 in Gould and Salaun [39] used various physical toys or toy tasks that required versions of select, delete, copy, and replace actions. As a side note, this experiment in some sense pointed to the advantages of two-handed interaction and tangible user interfaces. Both later became major interests of research in HCI.

Second, the number of analogue gesture commands of this nature is limited, which means the capacity of such gesture marks is small. They

are also limited to relatively simple analogue actions. This is evident in the study of Wobbrock, Morris and Wilson [120]. When the command becomes more complex and more abstract, such as "*replace* all instances of *colour* with *color*," or "print this document," the "natural" consistency of user's choice of gestures is likely to decrease. This reinforces the fact that UI naturalness is a relative concept. It has to do with the amount of prior experience, including experience in the physical world, to which the user can relate the current task.

Third, although the gesture marks chosen by the participants in the early studies were consistent to a high degree, they were not necessarily unique. For example in Experiment 1 of the Gould and Salaun study, there were always two or three gesture marks used for the same command. Some used a circle for scoping, others did not. Some used an X, and others used a pigtail proofreader's mark, for deletion. Some used an arrowhead to indicate the destination, but others expected the motion itself to indicate where the gesture started and where it ended. Interestingly, in Experiment 2 of Gould and Salaun, participants were asked and able to find another alternative gesture mark for the same command.

Despite these caveats, the point that users tend to be quite consistent in their ways of using gestures analogous to physical actions was convincingly demonstrated in the early empirical studies of Gould et al. The number of gestures of this type might be small, but they may cover a disproportional number of instances of action, hence they should always be taken advantage of as much as possible in UI design.

These early studies attempted to prove that the stroke-based gestural interface enabled users to enter commands faster than their typing-based lexical counterparts. Based on data reported in the literature at the time, the average time to execute a text editing operation was 20 to 35 seconds. In Experiment 1 of Gould and Salaun of drawing editing marks on paper, the average time was 15.9 seconds. This was increased by 25% to 19.8 seconds when the task was carried out with a stylus on a tablet that was separated from the display. When the same stroke-drawing task was done in a "clumsy" mouse-and-display

configuration (and with tasks actually executed), the average time was further increased to 21.1 seconds. Gould and Salaun [39] concluded that "It remains to be seen whether, with the eventual implementation of a hand marking command language, hand marking will maintain this advantage."

5

The Motor Control Complexity of Stroke Gestures

Laws and models that capture robust phenomenal or behavioral regularities play a fundamental role in the physical sciences. In the user interface field, "laws of action" that model user motor control behavior on the computer screen have also achieved considerable success (See Zhai, Accot and Woltjer [132] for a brief review). These empirical laws capture the regularities in pointing [35], crossing [4] and path steering [1, 2, 3]. A logical follow-up to this line of work is to extend this family of laws of action to stroke gestures. Just as Fitts' law could relate pointing time to a simple "index of difficulty" based on the geometry of the pointing task, ideally a quantitative relationship can be established between gesturing time and simple characteristics of a gesture stroke. Such a law would help the design and evaluation of existing or future gesture interfaces by quantitatively predicting their time efficiency without running extensive user studies.

However, in comparison to visual feedback-based control tasks, to which the existing laws of actions apply well, stroke gesture production is an inherently more complex behavior that involves planning, memory, and more importantly a certain degree of open-loop execution.

As a theoretical conjecture, a "law of gesturing" should reveal that the production time of a stroke gesture should be proportional to the

entropy in the stroke produced. The more varied, less predictable, more complex (hence higher in entropy) a stroke is, the more time it should take to produce it. The challenge is to reduce such a conjecture to simple computational forms relating to the stroke's geometry. To date no explicit model that can produce a law of gesturing exists.

However, elemental laws and "computational" models concerning the production of stroke gestures do exist. This section reviews three of them.

5.1 Viviani's Power Law of Curvature

Viviani and colleagues [116, 117] investigated human handwriting and drawing behavior in terms of instant movement velocity as a function of curvature, and proposed a power-formed model. A simple version of their formula is that at a given point on the written/drawn trajectory, the velocity is

$$V = KR^{\beta}, \tag{5.1}$$

where V is the instant (tangential) velocity of movement; R is the radius of curvature ($R = 1/C$, where C is the curvature), and K and β are constants of the model.

The model, known as the power law of curvature, indicates that the larger curvature the trajectory has at a given point, the slower the pen motion will be at that point. This model has been tested in different settings, including drawing trajectories with or without visual guidance [117].

Motivated by the need to model "sloppy" selection gestures, Lank and Saund [71] combine the power law of curvature, which models the structure of a gesture, and the steering law [1, 2, 3], which models trajectory constraint, to devise a compound model that describes both the structural and constraint properties of a gesture.

5.2 Isokoski's Line Segments Model

Isokoski proposed a model for stroke gestures that used the approximate number of straight line segments in a gesture as a predictor of complexity, correlating to production time [50]. An underlying

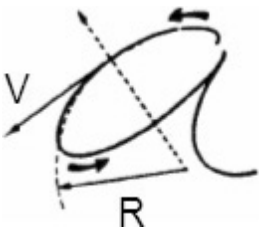


Fig. 5.1 Viviani's power law of curvature.

assumption is that drawing a straight-line segment takes a constant amount of time, regardless of the length of the segment. The model's best correlation result was $R^2 = 0.85$ on Unistroke characters [38], and it achieved R^2 between 0.5 and 0.8 for other gesture sets.

This model provides useful quick predictions, with the attractive merits of simplicity and ease of application. However, defining the number of straight-line segments needed to approximate a curved gesture is ambiguous. Furthermore, it does not provide an estimation of the magnitude of the actual production time.

5.3 Cao and Zhai's CLC Model

The CLC model by Cao and Zhai [22] decomposes any stroke gesture into three classes of gesture elements: curves, line segments, and corners (CLC), as is illustrated in Figure 5.2. In the CLC model the total production time of a gesture is computed as the sum of all time durations necessary to produce all gesture segments:

$$T = \sum T(\text{line}) + \sum T(\text{corner}) + \sum T(\text{curve}). \tag{5.2}$$

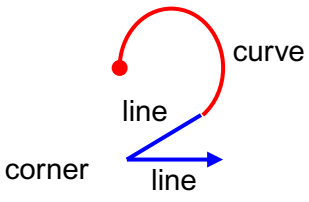


Fig. 5.2 Decomposition of a gesture.

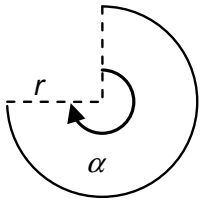


Fig. 5.3 Circular arc.

Each element of the CLC model is computed by a lower-level model as discussed in the following sections.

5.3.1 Smooth Curve

The production time of smooth curves can be derived from Viviani's power law of curvature. Integrating the instant velocity variable in Viviani's power law of curvature results in:

$$T(\text{curve}) = \frac{1}{K}\int_0^S R(s)^{-\beta} ds, \tag{5.3}$$

where

s is the curve length between the starting point and the current point;
$R(s)$ is the radius of curvature at s;
S is the total curve length ($0 \le s \le S$);
$T(\text{curve})$ is the total production time.

K and β are constants reflecting the performance of the individual person and device used. In Cao and Zhai's experiment, they were measured at $K = 0.0153$, $\beta = 0.586$, where s was represented in mm and T represented in ms.

A special case of smooth curves is an arc with a constant radius r and a sweep angle α (Figure 5.3). The total length of the arc is $S = \alpha r$. The total production time is then:

$$T = \frac{1}{K}\int_0^S R(s)^{-\beta} ds = \frac{1}{K}\int_0^{\alpha r} r^{-\beta} ds = \frac{1}{K}(\alpha r) r^{-\beta} = \frac{\alpha}{K} r^{1-\beta}. \tag{5.4}$$

5.3.2 Straight Line

The production time T as a function of the length L of a straight-line segment could potentially take various forms: a c*onstant, a linear relationship* $T(\text{line}) = aL + b$, or a *non-linear relationship* $T(\text{line}) = mL^n$. Empirical data collected from producing straight lines of various lengths in various orientations within specified precision best matched the following power-form nonlinear model

$$T(\text{line}) = mL^n, \tag{5.5}$$

where m and n are constants reflecting the performance of the individual and the device. For the group of individuals and device tested in Cao and Zhai [22] m and n were measured at 68.8, and 0.469, where L was represented in mm and T represented in ms.

The following linear model, with a nonzero intercept, also matches empirical data well, although not as well as the power model (Figure 5.4):

$$T(\text{line}) = a + b\ L, \tag{5.6}$$

where a was measured at 203 ms and b at 4.24 ms/mm .

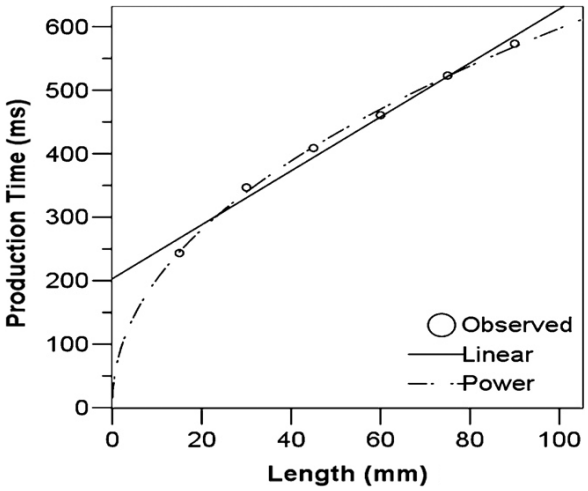


Fig. 5.4 Linear and power model of line segment production.

5.3.3 Corner

A gesture corner is an abrupt change in direction (with a zero radius). Since, by this definition, a corner is a single point, producing a corner per se takes zero or a negligible amount of time. The cost of a corner production lies in stopping one segment of the gesture and starting another. As is demonstrated in the empirical work of Cao and Zhai [22], the presence of a corner implicitly affects the production time by breaking one stroke segment into two, causing acceleration and deceleration in movement. For example, drawing two straight lines with lengths L_1 and L_2 connected by a corner takes a longer time than drawing one straight line with length $L_1 + L_2$.

5.3.4 The Total Model

Since the cost of a corner is only implicit, the CLC model therefore simplifies to summing the production durations of all curve segments and all line segments:

$$T = \sum T(\text{line}) + \sum T(\text{curve}). \tag{5.7}$$

Or

$$T = \sum_i m L_i^n + \sum_j \frac{1}{K} \int_0^s R_j(s)^{-\beta} ds. \tag{5.8}$$

The four constants m, n, K, and β should depend on the individuals and devices used. They are expected to be in the neighborhood of the values reported earlier ($m = 68.8$, $n = 0.469$, $K = 0.0153$, $\beta = 0.586$).

5.3.5 Verifications, Validity, and Applications of the CLC Model

Cao and Zhai empirically tested their models. Figures 5.5 and 5.6 show samples of component gestures and total gestures used in their experiment, respectively.

Cao and Zhai's study shows high correlations between the model predictions and empirical measurements, with greater than 0.9 R^2 value in all cases. Typically in Fitts' law-type models, the correlation coefficient is the measure of validity and the best-fit parameter values, in

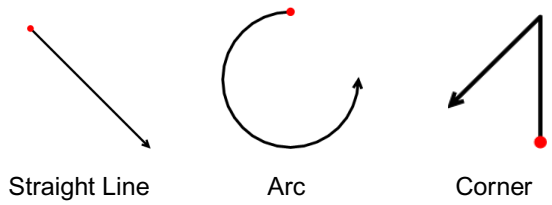


Fig. 5.5 Component gestures tested.

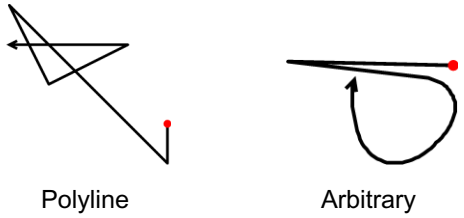


Fig. 5.6 Total gestures tested in Cao and Zhai [22].

this case m, n, K, and β, can be used to characterize the individuals or the devices tested. The CLC model can be used as an evaluation tool in the same fashion. However we should note that while R^2 values reflect the goodness of fit in variance between an empirical measured random variable with a prediction (such as time predicted by a model), it is not a be-all-and-end-all metric. It does not predict the magnitude match between two variables.

Another type of CLC application is to predict and compare the performance of one set of gestures against another in magnitude, under fixed m, n, K, and β parameters. This is a more stringent application of the model than purely correlation-based applications with free parameters. For example we might be interested in knowing the production time difference between one set of symbols against another. Or we might be interested in one set of alphabet letters such as Roman letters against another, such as Graffiti. In those cases the m, n, K, and β parameters have to be set at values as close to the target users and device as possible. If these parameters do not match the target users and device, there will be errors in the absolute values of prediction, although the relative order of the comparison may still be correct.

Cao and Zhai found two causes of errors in absolute value prediction. One was the underestimation of time with unfamiliar and little practiced gestures. CLC only models the motor control execution time. It does not include online visual perception, planning and decision making when producing unfamiliar gestures. The actual production time of these unfamiliar gestures could therefore be longer than the model prediction. The second type of magnitude error was the overestimation of familiar or well-practiced gestures. With well-practiced gestures not only could the visual perception, planning and decision making components be reduced to be negligible, but even the motor control aspect of two or multiple elements of a gesture also tended to be linked into one action, that is, a greater amount of "chunking" behavior tended to occur. For example although the canonical shape of the Arabic number 2 has a sharp corner at its bottom-left portion, a quickly written copy of it may round off the corner and hence gain higher speed. Chunking behavior in producing familiar gestures means that the actual production of these gestures can be more rapid than the model prediction based on the canonical shape. This overestimation can be corrected if desired, by modeling the actual shape with rounded corners that are in fact curves.

5.4 Applications of the CLC Model

Recall that we earlier reviewed Unistrokes [38] and Graffiti as two novel symbol sets, and alluded to the issue that although Unistrokes may require more learning than Graffiti, it may be more efficient than Graffiti once learned. With the CLC model, we now can quantitatively estimate the motor production difference between the two sets of gestures. To that end Cao and Zhai [22] first calibrated the CLC model to the five participants they tested and fixed the parameters of the model as follows (in ms):

$$T = \sum_i 88 L_i^{0.394} + \sum_j \frac{1}{0.0193} \int_0^s R_j(s)^{-0.566} ds. \tag{5.9}$$

They then computed time predictions T_p for each gesture sample from Unistrokes and Graffiti. Weighted by the letter frequencies reported

in Mayzner and Tresselt [86], they then calculated the mean gesture production time for Unistrokes at 622 ms and Graffiti at 1125 ms. The ratio between Unistrokes and Graffiti was 0.553:1.

They further collected experimental data from the five participants repeatedly writing Unistrokes and Graffiti symbols. Due to the possibility of chunking (see the second type of error discussed earlier) the experimental mean times per gesture, weighted by letter frequency, were 365 ms and 591 ms for Unistrokes and Graffiti respectively, both significantly lower than the model predictions. However the ratio between the two gesture sets, 0.618:1, was close to that of the model's 0.553:1.

Later Castellucci and MacKenzie [23] conducted a separate empirical study of Unistrokes and Graffiti involving 20 participants and 10 sessions of tests, each session involved 15 phrases of text entry. Probably due to the much longer learning time, the average gesture durations were 284 ms and 459 ms for Unistrokes and Graffiti respectively, even lower than those values measured in Cao and Zhai [22], but the ratio between the two gesture sets, 0.620:1, was still very close to the model prediction.

To review another sample application of the CLC model, recall that an unresolved question concerning the performance of gesture keyboards is the keyboard layout. The QWERTY layout is very familiar to computer users, but word gestures defined on them tend to zigzag back and forth. In contrast, gestures defined on an optimized layout, such as the ATOMIK, tend to be isotropic — moving in all directions with equal probability. Leaving the potential memory and accuracy advantages aside, are these more isotropic word gestures faster to produce than the zigzag QWERTY gestures? Cao and Zhai [22] tested the CLC model on ShapeWriter gestures defined on both the traditional QWERTY layout and the optimized ATOMIK layout [133]. They selected gestures for the 24 most frequently used words in spoken English and computed time predictions T_p for each gesture sample on both keyboard layouts. These predicted values were then compared with empirical data collected from users repeatedly drawing these gestures. The model predictions and empirical data correlated well, with an R^2 value of 0.96.

Both the model and the empirical data indicated that ATOMIK gestures were more efficient than the QWERTY gestures.

In summary, the CLC model provides an approximation and simplification of the actual user performance. The straightforward application of the CLC model may give predictions in the right direction between two designs, but it could either over or underestimate the absolute values of gesturing time. This in part depends on the amount of practice the user has. For well-practiced gestures, the user inevitably cuts corners at the joints between elements of the normative gesture templates and "chunks" them into one continuous curve stroke, resulting in a faster production than the model predicts. One could either view this as a weakness of the model, or apply the model with more sophistication than simply summing up the elements. One could use smooth curves to replace the sharp corners in the templates, as proposed in Cao and Zhai's discussion section of their report [21]. To what degree the corrections should be applied depends on the level of practice and chunking of the individual user and needs to be validated in further investigations [21].

Overall, motor control modeling of stroke gestures is still at a relatively primitive stage but the existing models are already useful as a research and design tool.

6

Characterizing Visual Similarities of Stroke Gestures

The last section reviewed motor control complexity as a foundation for gesture design. With the guidance of models presented there, one may be able to design the most efficient set of gestures for a given application from a motor execution point of view. However, a gesture set solely designed for motor efficiency could be very unusable if the gestures are all similar to one another. It is rather logical to assume (although no systematic empirical proof can be found in the literature) that gestures in a given system or alphabet that are more distinct from each other are less prone to confusion. This raises the question of how the human visual system judges gesture similarities and differences. Long et al. [82] did a comprehensive study to investigate such a question. They studied all gesture features drawn from Rubine's recognizer [100] and features identified from the authors' own observations. These features include such properties as initial angle, angle of the bounding box, distance between first and last point, total angle, etc.

Importantly, they found the following features, mostly related to size or scale, NOT important:

- Size of bounding box
- Total length

- Total angle traversed/total length
- Area of bounding box
- Log (area)
- Total angle/total absolute angle.

Through multidimensional scaling (MDS), they found the following five compressed dimensions and their correlating features (in decreasing importance) in gesture similarity visual perception.

- Dimension 1: Curviness, Angle/distance
- Dimension 2: Total absolute angle, Log(aspect)
- Dimension 3: Total length/Size of bounding box, Cosine of initial angle
- Dimension 4: Cosine of angle between first and last points, Cosine of initial angle, Sine of initial angle, Distance between first and last points, Angle of bounding box
- Dimension 5: Aspect, Sharpness, Cosine of initial angle, Total Angle

Through a regression analysis, Long et al. derived a model that correlated with experimental data at 0.74 R^2 value.

Understanding visual similarities is critically important to a gesture recognizer and gesture set design. However, no stronger models on visual similarity have been developed since Long et al.'s work.

7

The Role of Feedback in Gesture Interfaces

Another fundamental issue in gesture interface design is if and how feedback should be provided. Basic gesture visual feedback in the form of digital ink is rather straightforward to display, yet its impact is not obvious. On one hand one can argue that such feedback assures the user that the gesture produced meets the user's expectation and helps the user to produce stroke gestures more accurately. On the other hand, one can also argue that visual feedback draws the user's attention but is too late to help the user correct the gesture.

7.1 Visual Gesture Feedback

Andersen and Zhai conducted a systematic and baseline experimental study evaluating the impact of visual feedback on gesture production [5]. Their entire experiment included a factorial combination of visual and auditory feedback conditions. For ease of comprehension we first review the impact of visual feedback here.

The experimental task was to reproduce a set of 20 (classes of) gestures of varying complexity according to their respective templates, the canonical forms of the gesture classes. The participants were asked to

write with a digital stylus on a digitizer tablet as accurately as possible in a normal writing speed. The target template gestures were first presented on a screen and the reproduced gestures were recorded. Using the recordings, the stroke gestures were measured and analyzed in multiple ways according to gesture size, gesture speed, aperture, shape distance, and directional difference. In the visual feedback condition, the system displayed the ink traces as the participants produced the stroke gestures. The impact of such feedback, in comparison to no feedback, is as follows:

Size, as defined by the area of the bounding box of the gesture, was significantly influenced by visual feedback. With visual feedback, the average size of a gesture was about $4.5\,\mathrm{cm}^2$. Without visual feedback, the average size of a gesture was about $7.5\,\mathrm{cm}^2$. Figure 7.1 shows examples of gestures produced with and without visual feedback in the experiment. Such an effect has been previously observed in the handwriting literature. For example van Doorn and Keuss demonstrated that handwriting without visual feedback was larger than with visual feedback [113].

Speed. The average movement speed (movement length per unit time) also changed significantly with visual feedback conditions. With visual feedback, the average speed was 6.2 cm/sec, lower than the average speed of 7.8 cm/sec without visual feedback.

Completion time. While visual feedback tended to have a large effect on gesture size (smaller with visual feedback) and drawing speed (slower with visual feedback), it had a statistically significant but

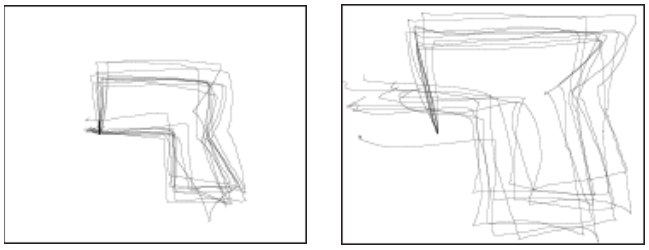


Fig. 7.1 Multiple trials of gestures produced with (left) and without (right) visual feedback.

magnitude-wise much smaller effect on gesture completion time. The average completion time used to produce a gesture was 1.23 sec with visual feedback and 1.32 sec without visual feedback. With visual feedback the slower speed and smaller gesture tended to compensate for each other, resulting in a completion time similar to (although still shorter than) the completion time without visual feedback.

Aperture. In the experiment, half of the gestures were closed between the beginning and ending of the gestures (e.g., a square). In actual production the distance between the beginning and the ending points, called aperture, may not be zero. Without visual feedback, returning to the beginning point is, not surprisingly, much harder. It requires accurate reproduction of the order, direction, as well as the relative proportions of the individual segments in a gesture (see Figure 7.1 for examples). Indeed, in the experiment, visual feedback helped to reduce the average gap between the beginning and the ending points to about 60% of the average gap in the no-feedback condition. Aperture reflects the same type of referential ability to cross the *t*'s and dot the *i*'s, as in handwriting research [107]. Being able to accurately cross a referential point is perhaps the most important positive role of visual feedback.

Shape distance. In addition to size, speed, and referential features such as aperture, it is also informative to know how visual feedback impacts the overall shape of the gesture produced relative to the target template gesture. One measure of shape difference is *proportional shape distance* (PSD) d. To calculate d, both the template and the drawn gesture are normalized to the same size and each of them is sampled into a fixed number of equidistant points. d is then calculated as the mean distance between the corresponding sampling points in the drawn gesture and in the template. Such a measure is sensitive to the overall shape proportion, but not to local features. We will review PSD as a gesture recognition feature later in the section "Gesture Recognition Algorithms and Design Toolkits."

As measured by PSD, visual feedback made no statistically significant difference to shape distance. An important implication here is that if the gesture recognizer relies on a gesture's overall shape and ignores

referential features, then visual feedback would have little impact on recognition accuracy.

Directional difference. Another measure calculated in Andersen and Zhai's experiment was *directional difference*. This measure compares only directional differences in all corresponding segments of the two stroke gestures (the template and the drawn gesture) and discards the extent (length) of each segment. To compute such a measure, the gestures first needed to be segmented based on inflection points (see [5] for details). The experimental results showed no statistical difference with and without visual feedback in this measure either.

In summary, visual feedback has a large impact on gesture size and on referential features such as the aperture of the closed gestures. It has a small impact on completion time, but no significant impact on the overall shape or segment direction accuracy. These findings point to different aspects of stroke gesture production. Fundamentally, the spatial-temporal patterns of a gesture recalled and reproduced from memory (inside-out) are largely open loop and articulated without much feedback modulation. It is important to take advantage of such aspect of gesture production so that information input based on gestures can be achieved eyes-free, a key point emphasized in the design of Unistrokes [38]. On the other hand, if referential features that require more of a closed-loop control behavior ("crossing the *t*'s") and if the size of the stroke gestures is important, visual feedback can be very useful.

7.2 Audio and Other Forms of Gesture Feedback

7.2.1 Auditory Gesture Feedback

There are compelling reasons to believe auditory feedback can play a potentially interesting role in the use of stroke gestures. In comparison to absolute judgment of sound quantities such as pitch or loudness, the human auditory modality is more sensitive to relative rhythm and patterns [26]. For example, although it was difficult to tell which tones corresponded to which digits in the old modem dialing experience, one could often tell if the entire sequence was correct based on the overall melody. Considering that stroke gestures are spatial or movement

patterns, it might be beneficial to couple these movement patterns with sound feedback. A related experience is the toy, "Simon," in which the sound pattern may help the player memorize the previous sequence of keys. Tight coupling between sound and kinesthetic patterns is most evident in musical activities such as dancing and instrument playing. It is difficult to imagine dancing without sound patterns (music). In the case of many musical instruments, the kinesthetic spatial movements of the musicians and the temporal sound patterns produced are inseparable. Wouldn't it be interesting, at least as an option, to turn a stroke gesture interface into some form of musical instrument that is similar but hopefully less eerie than the Theremin? However, one should be mindful that musical skills are acquired over intense and longitudinal practice. That great amount of practice is usually unacceptable to user interfaces.

There is a vast design space of auditory feedback and there are many ways of mapping a gesture to any particular design. A basic goal should be to make the auditory feedback both informative and pleasant. To be informative means the feedback should indicate the shape of the gesture in real time. As the gesture changes direction or speed, the user should be able to tell the difference from the auditory feedback. Often a discriminative signal may not be pleasant to listen to. A simple design, for example, is to map the x and y coordinates of the gesture movement to the amplitude and frequency of a continuous complex tone. While such a design produces rather informative signals, it is unpleasant to listen to. To make it more pleasant, more complex sound, such as musical tones or the sound of different musical instruments could be used, although such a design may not be as responsive to instantaneous changes of gesture and hence less informative.

One lesson learned in Andersen and Zhai's exploratory study [5] was that auditory feedback of stroke gestures tended to be overshadowed by visual-spatial memory. To obtain a baseline performance impact, we designed an experiment in which 20 gestures were reproduced with visual or auditory feedback. The auditory feedback was made as discriminatory as possible through a complex tone. The vertical position of the pen tip was mapped to the fundamental frequency of the tone: a higher tone was played if the gesture moved to a higher y position. The

horizontal position of the pen tip was mapped to the number of overtones, creating more pure tones on the left and richer tones on the right. In addition, jitter, that is, random frequency variations of each partial, and inharmonicity were used to further increase perceptual difference in the horizontal x dimension.

In Andersen and Zhai's experiment, amplitude was mapped to the speed of the pen, so that a fast pen speed created a loud tone. This also had the effect of making the sound less intrusive when the pen was at a stop, which can be important for a practical realization of auditory gesture feedback.

Another perceptual parameter used to encode the position of the pen was the perceived direction of the stereo sound source. The direction of the pen position relative to the starting point would determine the direction of the sound source in the horizontal plane. If the pen was moved to the left of the starting point, the complex tone was perceived as coming from the left, whereas when it was moved above the starting point, the tone was perceived as coming from the front.

With such an auditory feedback signal carefully designed to make different spatial patterns discriminatory in the auditory space, Andersen and Zhai measured and analyzed the same aspects of gestures as in the visual feedback conditions reviewed in the last subsection. The main effects of audio feedback, in comparison to visual feedback, are summarized as follows.

Size. While gestures produced with visual feedback were much smaller than without, gestures produced with audio feedback were only slightly smaller than without.

Speed. While the average movement speed (movement length per unit time) with visual feedback was significantly slower than without, auditory feedback did not have a statistically significant effect on speed although the mean was slightly lower with auditory feedback as well.

Completion time. Recall with visual feedback gestures tended to be smaller in size and slower in speed. These two effects compensated for each other, resulting in a gesture completion time with visual feedback similar to but still shorter than without visual feedback. In the case of audio feedback, the completion time was not significantly influenced.

Aperture. One of the strong effects of visual feedback was on the accuracy of closing the aperture of the closed gestures. Audio also had a similar effect, although to a much lesser extent.

Shape distance. Like visual feedback, audio feedback had no measurable effect on the accuracy of the overall shape produced.

Directional difference. Like visual feedback, audio feedback had no impact on this measure.

In summary, audio feedback had effects to a much smaller extent on those aspects of gestures where visual feedback had large effects. These include gesture size and aperture. Where visual feedback failed to have any impact, such as a gesture's overall shape, auditory feedback did not have any impact either.

Andersen and Zhai's study did not lend support to auditory feedback's contribution to learnability, retention, or in-process correction, which they set out to investigate [5]. It was realized that the type of learning involving in playing music instruments and dancing would take much longer time than acceptable UI learning time. Their investigation also showed a strong spatial pattern dominance. People tended to remember the spatial patterns rather than the auditory patterns. Overall their investigation on audio feedback reinforced the understanding that gesture production is largely "open-loop," recalled and reproduced from memory (inside-out), and feedback would be too slow to modulate the production except in cases where there are reference points "crossing t's") that need to be addressed.

However, guiding the gesture production *process* is not the only possible use of auditory feedback. It is much easier to use auditory feedback to inform the user about the *product*, or result, of the gesture articulated. Similar to touch-tone phone sounds, auditory feedback of gestures could potentially help the user recall gestures or at least let the user notice right away if a gesture is ill-articulated. An even more obvious type of product feedback of gestures, with which we experimented, is to pronounce (in appropriate situations) the ShapeWriter gesture keyboard's output word so that the user does not have to visually check if the correct word is entered. Similarly, auditory feedback can also be used to pronounce the result of a pie menu selection [140].

7.2.2 Haptic Feedback

In addition to visual and audio feedback, it is also possible to provide haptic feedback to stroke gesture production. One can feel the difference between writing with a fountain pen on paper and writing with a ballpoint pen on a hard surface. When we discussed such issues with designers at touchscreen device makers, we learned that device makers pay attention to the material used for the pen tip for better feel of the digital stylus. It is conceivable that various vibrato-tactile feedbacks can be further provided via relatively simple technology to create virtual frictions on the touchscreen. The impact of haptic feedback on gesture, however, is yet to be systematically studied in the literature.

When the touchscreen is capacitive, the haptic experience of finger gesturing on the smooth screen tends to vary from one individual to another. Although we know of no formal studies, our own observation is that dry finger skin tends to slide on smooth surfaces more easily, but moist skin tends to be sticky on smooth screens. This is rather evident when operating a touchscreen device immediately after washing one's hands.

Conceivably, haptic feedback can at least provide anchor points on the touchscreen. For example a touchscreen may give haptic feedback if a few reference points are crossed, very much like the role of raised dots on selected keys on physical keyboards. For special applications of touchscreens, it is also possible to superimpose physical templates over the touch surface to assist the user in eyes-free operation [19].

The original version of EdgeWrite [124] is an interesting example of exploiting haptic assistance in gesture production. It is a text entry method for handheld devices designed to provide stability of motion for people with motor impairments. Its special alphabet leverages the physical edges for greater stability in text entry. A user enters a modified version of Roman letters by traversing the edges and diagonals of the square hole of a plastic fixture mounted on a PDA device. Although the EdgeWrite letters differ in visual appearance from the standard Roman letters, the two sets have quite similar kinetic movement patterns, which makes EdgeWrite easier to learn [124].

Electrovibration is an intriguing mechanism to implement haptics without mechanical actuators. Like stereoscopic displays that can create synthetic 3D imagery to human eyes, electrovibration can create a synthetic texture perception to the human finger [14].

7.3 Feedback and the Esthetic, Subjective, and Emotional Experience of Gesture Interfaces

Drawing stroke gestures is a fun experience for most people. This advantage should be exploited in product design. Most children enjoy doodling and drawing from a very young age. The continuous motion of drawing gestures is arguably more enjoyable than the discrete motion of typing.

In addition, designers of gesture interfaces can do more to make the esthetic, subjective, and emotional experience of a gesture interface more pleasing, engaging, and fun. The importance of designing for emotional appeal was well articulated in Norman's book on the subject [93], which notably was published well before the wild market success of the Apple iPhone, which has a strong emotional appeal.

However not much is known about how to make an interface more fun to use. Design of feedback, either visual or auditory, might be one important contributing factor. Many researchers working on sketching UI have developed various graphics techniques to "beautify" hand-drawn digital ink after it is drawn into known geometric shapes such as squares and circles [11], and 3D objects [49, 130].

There are documented techniques in cartoon animation that have been explored in making fun user interface applications [24]. For example an object that prefaces forward movement with a small and quick contrary movement, or an object that comes to a stop and vibrates into place, would appear more fun, more engaging and more lively than otherwise.

However, making an interface more fun can conflict with the goal of fast performance. For example in the Tablet PC version of ShapeWriter, we implemented stroke animation that morphs the approximate gesture drawn by the user into the template as recognized by the system [59] (Figure 7.2), similar to the effect of making hand drawn CAD lines into

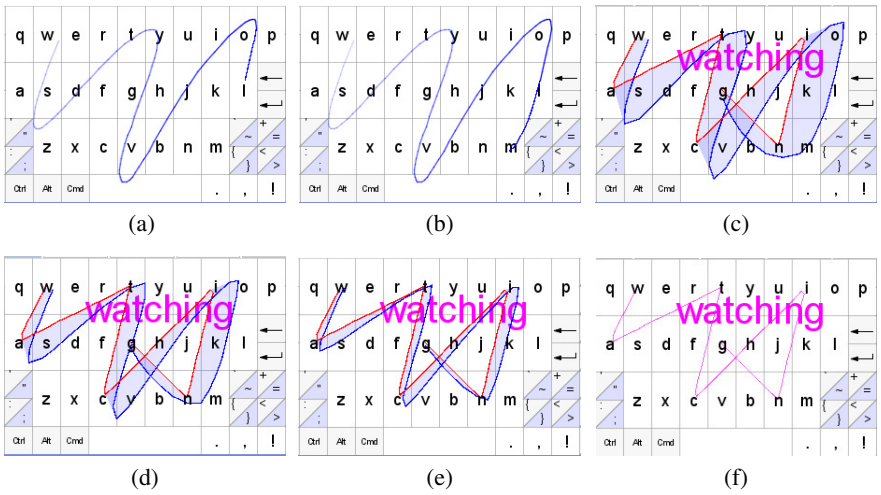


Fig. 7.2 The gesture animation in ShapeWriter: (a) the approximate gesture w-a-t-c-h-i-n-g is being drawn; (b) gesture is near completion; (c) immediately after finger lift, the ideal trace (straight lines), the actual trace drawn, and the word as recognized are displayed; (d, e) the actual trace is morphed toward the ideal trace; (f) the animation completes.

perfect shapes in SketchPad according to its built-in rules and models [108]. The stroke animation in ShapeWriter not only informs the user as to how much he or she has deviated from the ideal trace of the recognized word, but also makes the interface more lively [59]. On the other hand, the animation process may slow down the user and take the user's attention away from entering the next word, which is often what the user wants to focus on. Although the user can always choose not to look at the animated feedback and forge ahead, there was still a question whether the animation gives the user a more complex impression than necessary. No systematic empirical study has been reported on the effects of animated gesture strokes.

The Android version of ShapeWriter made a compromise between the aim of teaching the user via animated feedback and fast performance. As soon as the user lifts his or her finger, the ideal trace of the recognized word was displayed immediately, without intermediate animation steps. This still gave an apparent snapping motion from the actual trace to the ideal trace, which may potentially teach the user the ideal shape and make the UI livelier.

Auditory feedback also has the potential to make stroke gestures more fun and more engaging. Andersen and Zhai [5] presented an experiment on "writing with music" in which they coupled stroke gestures with various audio feedback such as music and continuous tones. They evaluated a number of subjective dimensions such as "terrible-wonderful" and "dull-stimulating" under different auditory feedback conditions. On the dimension of "dull-stimulating," musical feedback was rated significantly better than both silence and a continuous tone. On the "terrible-wonderful" dimension, however, the musical and silent conditions were rated equally high, both significantly better than the continuous tone. Although the results show significant statistical differences between the tested conditions, it was also clear that there were large individual differences between the tested users. This indicates that auditory feedback should be made optional in practical systems, allowing the user to turn on and off or adjust the feedback provided.

8

Memory and Cognitive Aspects of Stroke Gestures

In addition to motor control execution and perceptual feedback of stroke gestures, cognitive functions related to stroke gestures, particularly learning and memory, are also very important to user interaction. We can find much anecdotal evidence that humans are good at remembering spatial patterns in comparison to, say, remembering sequences of numbers or symbols. For everyday life, we encounter, recognize and use spatial patterns ranging from letters and symbols, to routes to work and other places. For people who use logographic writing systems such as the Chinese script, being able to recognize thousands of visual-spatial patterns is a necessity for basic literacy.

Little gesture interface research to date has been linked to more general human memory research, which distinguishes memory into declarative memory and procedural memory [112]. Declarative memory is about knowledge and facts and is explicit. Procedural memory on the other hand is about skills and how to do things, particularly body control. Procedural memory is unconscious or implicit. Stroke gestures are likely to involve both declarative and procedural memory and possibly shift from the declarative side to the procedural side, falling below conscious awareness as learning progresses. Similarly,

motor control and learning research suggests that voluntary actions are initiated by a conscious goal, but the perceptual-motor integration, sequencing, spatial representation and movement dynamics are outside of awareness [119]. Procedural memory and motor skills are typically long lasting. Skills such as bicycling or skiing, once learned, are hardly ever forgotten [103].

8.1 Gesture's Comparative Advantage to Lexical Commands

Before menu-driven perceptually-based graphical user interfaces (GUI) became dominant, memory recall-based lexical commands, such as DO, SET, and JUSTIFY, were the main form of HCI in command line interfaces. It was therefore logical to compare and contrast the cognitive properties of gestural versus lexical commands. Morrel-Samuels [89] draws on a body of social and cognitive psychology literature to argue that in comparison to lexical commands, gestural commands, consisting of hand markings such as a circle, a cross or an arrow, "should be faster to use, more memorable, and more likeable." Borrowing from Ekman and Friesen's work [34] on hand movements, Morrel-Samuels separated stroke gestures into two groups: *emblematic* gestures with a specific meaning capable of being translated into words (e.g., an X gesture for delete) and *illustrative* gestures for pointing, demarcation, or emphasis (e.g., circles, braces, arrows). He postulated five advantages of gestural commands:

1. Gesture commands are terse in that they are already at the very minimum in complexity.
2. They are common in that different people tend to use the same gesture (e.g., X) for certain function (e.g., delete).
3. They are less ambiguous than words.
4. Gestures are strongly represented in human memory hence gestural commands can be more fluent (fewer pauses) than lexical commands.
5. Emblematic gestures are iconic in that the meaning and shape of gestures are not arbitrary.

Note the types of gestures considered by Morrel-Samuels was much narrower and smaller in capacity (the number of possible gestures matching the criteria given) than what is intended in today's user interfaces, but many of the arguments may still hold true.

Other researchers have identified and applied different cognitive insights and principles pertinent to gestures in HCI. For example, Buxton uses the notion of chunking and phrasing as key to organizing and structuring the interaction process [17]. The proprioception involved in gesture control, such as muscle tension, can facilitate chunking and phrasing. Gesture research projects such as GEDIT [62] and Scriboli [43] further articulated and demonstrated gesture's potential to integrate object selection and action into one continuous chunk of interaction.

Today, search-and-click type perceptually-based GUI interfaces continue to be the dominant form of HCI. However the lack of efficiency in this paradigm is also evident. Take the current generation of smartphones as a recent example, applications and functions are laid out as separate icons with labels on the touchscreen. This makes it rather easy to select a specific function the user wants to activate, by visually searching for and then touching on a particular icon. However, as the user accumulates more applications, the number of icons exceeds what one or two screens can accommodate, making it more tedious and time consuming to select an application. The situation calls for some form of faster recall-based means of accessing functions. The medium of recall can be either words (which requires typing) or gestures. The recent work of Li [74] on Android phones, which allows the user to gesture the first letters of names for retrieval has already begun to address the shortcomings of browsing-based mobile UI design.

8.2 Learning Stroke Gestures is Easier than Learning Keyboard Shortcuts

A recent experiment by Appert and Zhai demonstrating the cognitive advantage of stroke gestures is in the area of command shortcuts [10]. Keyboard command shortcuts, such as Ctrl-C for copy, are a memory recall-based, faster alternative means of accessing computing functions

to the menu-driven GUI interaction. For touchscreen devices without a physical keyboard, gestures could play a similar, or even stronger, role as a recall-based alternative to icon selection. Furthermore, Appert and Zhai's experiment demonstrated that people were about 200% better at mastering gesture shortcuts than keyboard shortcuts.

Their experiment tested people's speed of learning stroke gesture versus keyboard shortcuts. While both hot keys and gestures can be designed to be mnemonic, the experiment deliberately mapped a set of menu items to arbitrary gestures and arbitrary hot keys in order to establish an unbiased baseline comparison (Figure 8.1). In each trial, the experiment required its participants to select an item in one of the menus attached to the menu bar, either through menu navigation or through shortcuts. The shortcuts, as hot keys or as stroke gestures (depending on the experimental condition), were displayed beside the corresponding menu items (Figure 8.2). The participants were asked to remember and use as many shortcuts as possible. If they didn't remember the shortcuts, they could use the menu to select the command or look up the shortcuts in the menu. There were five menus (recreation, fruits, vegetables, office, and animals) with each menu containing 12

ICON	Keys	Gesture	ICON	Keys	Gesture
	Shift+W			Ctrl+W	
	Shift+D			Ctrl+D	
…	…	…	…	…	…

Fig. 8.1 Examples of keyboard and gesture shortcuts to arbitrary concepts.

None	mushroom	Warm-up (day 1)
Keyboard	mushroom ⇧S	Test (day 1)
Stroke	mushroom	Test (day 1)
Both	mushroom ⇧S 	Re-Test (day 2)

Fig. 8.2 Shortcut conditions tested in [10].

menu items in the experiment. In varying repeating frequency, 14 of the total of $5 \times 12 = 60$ items appeared to the participants as target commands to be activated. The rest of the 60 items were also assigned shortcuts and served as distracters.

After two blocks of warm-up trials in which only menu navigation was used to activate commands, each participant went through four blocks of 60 trials in each condition (gesture shortcuts versus keyboard shortcuts). The order of the conditions was balanced across participants.

The results of the experiment strongly supported the learning and memory advantage of gesture shortcuts over keyboard shortcuts. In terms of completion time, defined as the total time elapsed (in ms) from presentation of the command icon to completion of the correct command, on average the gesture condition was significantly faster than the keyboard condition ($F_{1,13} = 36$, $p < 0.0001$). As shown in Figure 8.3(a), while the completion time in both conditions decreased with the number of repetitions and given enough practice both would reach a similar

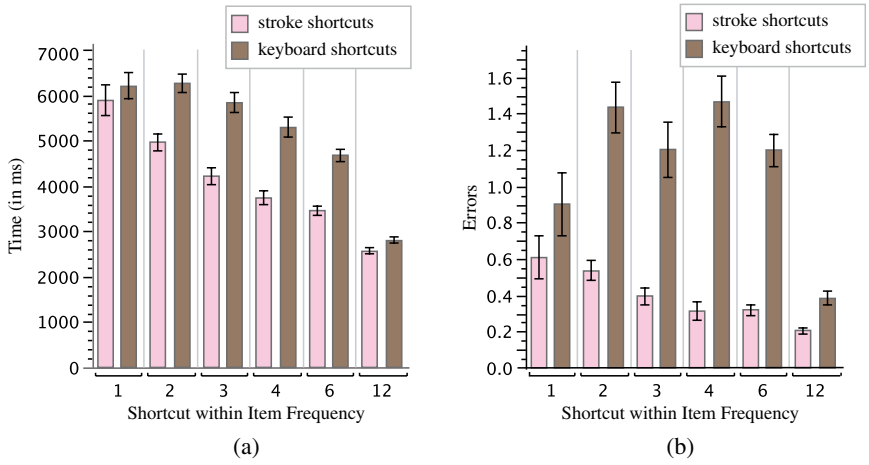


Fig. 8.3 Completion time (a) and error per trial (b) as a function of the number of times a command was repeated in each of the four testing blocks of trials. Both keyboard shortcuts and gesture stroke shortcuts improved with practice, but gesture shortcuts performed significantly better than keyboard shortcuts in both time and error measures particularly with a moderate amount of practice. Adapted from Appert and Zhai [10].

floor, the stroke gesture condition decreased much faster than the keyboard condition, indicating faster learning of gestures.

Error rate, defined as the number of times the participants entered a wrong shortcut before entering a correct one, was significantly lower in the stroke gesture condition than the keyboard condition. In the stroke gesture condition, 19% of the trials contained at least one error with participants trying 0.32 gestures on average before entering the right one. In the keyboard condition, the percentage of error trials was 22% with 0.49 attempts on average before actually triggering the right command. Figure 8.3(b) shows error per trial as a function of the number of times a command was repeated in each of the four testing blocks.

In terms of recall, defined as the number of times the participant was able to activate the right command with a shortcut without opening the menu and without any error, the stroke gesture condition also performed significantly better than the keyboard. As shown in Figure 8.4(a), the recall rate improved over practice in both conditions (and both would reach a near perfect ceiling eventually), but the gesture condition improved faster than the keyboard condition.

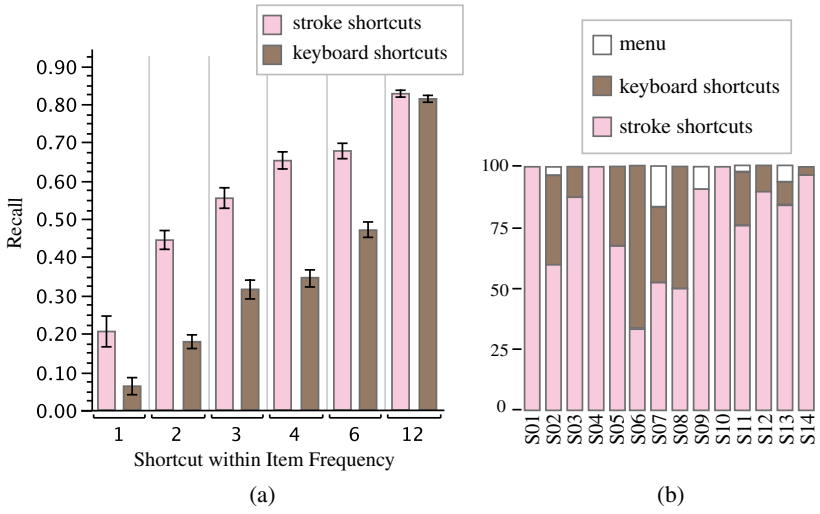


Fig. 8.4 Recall and reuse: (a) The ratio of correct recall as a function of the number of times a command was repeated in each of the four testing blocks of trials. (b) The percentage of gesture stroke shortcuts, keyboard shortcuts and menu selection used by each participant in the second day re-test. Adapted from Appert and Zhai [10].

Even stronger results in favor of gesture shortcuts were obtained in the data collected in a re-test session. On the second day of the experiment, the participants were told to complete two more blocks of tests as quickly as possible by using the method of their choice for each trial of command activation. Importantly the participants were not told what would be in the second day's experiment so that they would not consciously rehearse the shortcuts during the break between the two days. Although there were individual variations, on average significantly more stroke shortcuts than keyboard shortcuts were used ($p < 0.0001$). The overall mean percentages of use for the three techniques were: 77.7% gesture shortcuts, 20.3% keyboard shortcuts, 2% menu. In other words, given an equal amount of practice in the previous day and free choice, gesture shortcuts were used nearly four times (3.82 times precisely) as often as keyboard shortcuts. Figure 8.4(b) shows the percentage of gesture stroke shortcuts, keyboard shortcuts, and menu selection used by each participant in the second day re-test.

As a final test, the participants were asked to draw or write down both gesture and keyboard shortcuts as they remembered them, next to the list of 14 target commands that appeared in the experiment. The result again shows that people's ability to remember stroke gestures was much stronger than their ability to remember hot key combinations. Averaged across the participants, 11.6 gesture shortcuts were correctly answered and, in contrast, only 4 keyboard shortcuts were correctly answered. In other words, the participants did 190% better with gesture shortcuts than with keyboard shortcuts in the second-day re-test.

The results of Appert and Zhai indicate a clear inherent cognitive advantage of stroke gestures in comparison to key combinations. Stroke gestures, even when they are arbitrarily assigned to a command, tend to give richer perceptual cues to the user, to form an association between the shape of the gesture and the meaning of the command. For example when a circle was assigned as fish, a participant "thought of this stroke as fish because the shape's stroke makes me think about a pond." For another example, "I associated this stroke with a jump and I see karate as a sport where people jump." For yet another example, when an upward straight stroke was arbitrarily assigned to the object "bat", the user may make the association of a bat flying upwards.

More theoretically, human memory research has suggested that elaboration, more levels of encoding and deeper processing help memory [29]. The spatial and iconic information in a stroke may better enable users to imagine (encode) an association between the gesture and its corresponding commands in a more elaborate fashion, and hence help their learning performance.

In summary, the experiment shows that gesture shortcuts can offer significant advantages over keyboard shortcuts in terms of memory and learning.

Note again that in the experiment of Appert and Zhai they chose a baseline comparison in which functions were assigned arbitrary gestures and arbitrary hot keys. Both were deliberately deprived of mnemonic design. Notwithstanding the findings in the study, in practice one can (and should) design both gestures and hot keys with analogue or mnemonic properties when possible. One simple way of making hot keys mnemonic is to use letters in a function's name as a shortcut. For example Ctrl-C is for *copy*. But this approach has limited capacity: i.e., Ctrl-v, not Ctrl-p, is for *paste*. This approach can also apply to gestures. Li's gesture search [74] allows the user to keep writing letters in the word until the desired function (an app, a setting menu, or a contact name) appears in a short list for selection.

8.3 People's Ability to Learn and Memorize Stroke Gestures

The last section reviewed an experiment that demonstrated that people's ability to learn gestures, even if arbitrarily mapped to semantics, was superior to learning key sequences. This section reviews an experiment that shows people's capacity to memorize gestures was only limited by their speed of learning, which could be as high as 15 gestures per 45 minutes of practice in an experimental set-up and training scheme.

Zhai and Kristensson [134] provided such an empirical measurement of the number of novel stroke gestures people could learn in a limited amount of time on a word-gesture keyboard. How well people can learn a new skill partly depends on the methods with which they practice

the new skill. Zhai and Kristensson used a revised version of the ERI (expanding rehearsal interval) method, which gradually increases the time interval between repetitions of the same task [68]. We will describe the ERI method in some detail in the section "Providing game-based training" later.

In Zhai and Kristensson's study, participants were asked to learn and memorize as many word gestures as possible on an experimental word-gesture keyboard (SHARK) in four 45 to 60 minute sessions. The word gestures in the experiment were defined on the ATOMIK keyboard layout [133], which was new to the participants so all gestures tested were novel. The gesture keyboard recognition algorithm used in the experiment was scale and location independent (albeit for a relatively small vocabulary).

For each practice trial, a target word was presented to the participant, who was asked to draw the gesture of that word in a blank window without any visual memory assistance. If the gesture was correctly recognized as the target word, the trial was a success. The target word would be rescheduled for practice with an increased interval. A new practice trial with a new word from the practice queue would be presented to the participant. If the gesture was not recognized as the target word, the participant could make another attempt at drawing the correct gesture. If the participant could not recall the gesture for the target word, or if the target word had never been practiced before, the participant could learn it (for a new word) or relearn it (for a forgotten word) by displaying the ATOMIK keyboard, together with the ideal gesture drawn in dotted lines connecting the letters in the word (Figure 8.5). The participants could copy or draw the word gesture anywhere on the keyboard in any scale, and practice the gesture as many times as they wanted to explore the tolerance of acceptable shapes for the word before moving onto the next target word.

Session 1 of the study was a 40-minute practice session only. Sessions 2, 3, and 4 consisted of two parts. Part one was a test session of words the participant had practiced in previous sessions, lasting from 6 to 20 minutes. There was a minimum period of one day between a test and the previous practice session. Part two was 45 minutes of practice

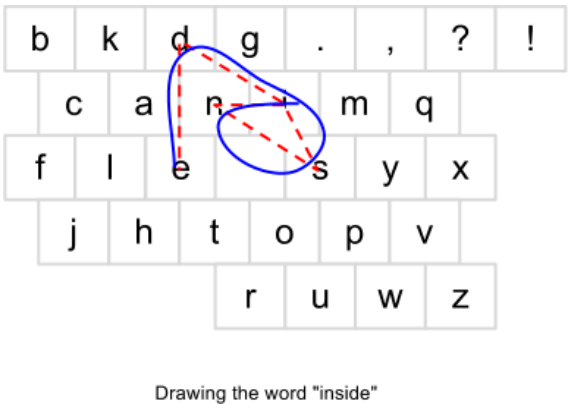


Fig. 8.5 Users' ability to learn gestures was tested on a novel (ATOMIK) gesture keyboard layout.

(including a five minute break in the middle). Session 5, the last session, was a final test session only.

In test sessions, words were presented in a random order. The participants were given at most two chances to correctly recall and write a word gesture in a blank window without any visual assistance.

The results of the experiment showed that all six participants could learn to draw correctly recognizable word gestures for any word presented to them, if practiced enough times. Typically it took 7 to 15 practice trials to fully memorize a word gesture.

As shown in Figures 8.6 and 8.7, participants were able to correctly write more words in each learning session, on average about 15 more words per session. In the final test, on average they correctly produced 48.83 (between 62 and 39) words in their first attempt, and 58.67 (between 77 and 49) words if counting the second attempt when the first failed. Interestingly, the number of new words learned per session was nearly constant (Figure 8.7).

Although the Zhai and Kristensson study was in a particular context (word-gesture keyboard) and with a specific learning regimen (ERI), it showed one data point regarding people's ability to learn arbitrary gestures. The participants in the experiment were able to master 50 to 60 gestures in four test sessions with no signs of slowing down, a

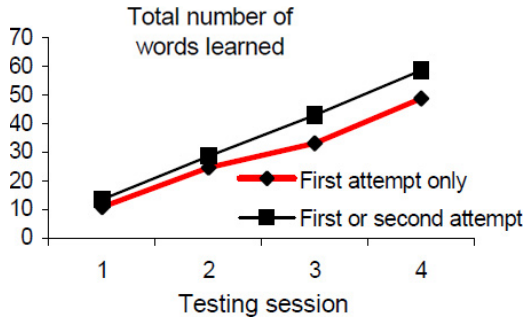


Fig. 8.6 Total number of words correctly written in test sessions.

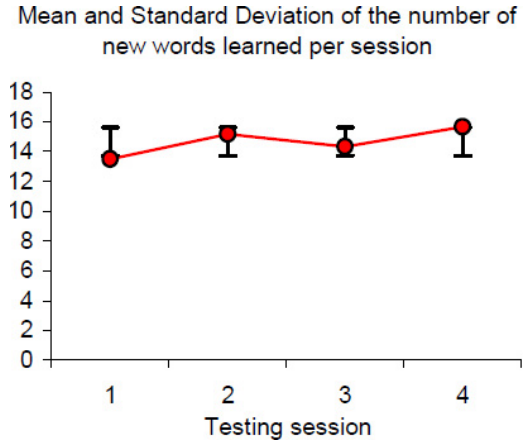


Fig. 8.7 More words learned in each session.

rather large number for most gesture systems. Even for text input, which requires thousands or tens of thousands of words, 50 to 60 words is still a very large number given the Zipf's law effect that a small number of common words covers a disproportional number of instances of input. For less common words, the common word fragments (and corresponding gesture parts) already mastered could still improve user's overall performance.

9

Gesture Design

The previous sections reviewed the basic theoretical and empirical issues in people's cognitive, perceptual, and motor control abilities in dealing with stroke gestures. We now turn to some of the mechanisms that can be potentially used in practical gesture system design. Although ultimately design is a creative process, the following basic principles should be applied or at least be considered in gesture design. None of these principles, however, is straightforward and they certainly are not necessarily compatible with each other. In fact, balancing these conflicting factors according to the overall product design goal may well be the key to the success of gesture-based user interfaces.

9.1 Making Gestures Analogous to Physics or Convention

For ease of learning, gestures should be designed to be analogous either to the physical effects of the real world or to well-known conventions and stereotypes. For example, an upward gesture can be used for volume up; a question mark gesture for help, a left to right gesture for next or forward etc.

In cognitive linguistics, it has been argued that real-world physical concepts, such as up and down, are deeply reflected in the way we express our thoughts [52]. While the number of gestures that can be designed in such a way may be limited, particularly when other dimensions discussed in the rest of this monograph also have to be considered, it is quite possible to find enough of them for a particular application, such as in a mobile music player [97]. Most gestures on the current generation of smartphones are analogue or direct manipulation gestures.

In the case of gesture-based text input, the strongest conventions are various existing writing systems, or "natural" hand writing scripts. These writing systems are natural only in the sense that people have already learned them and they have evolved in various cultures for hundreds or thousands of years. However because they are not designed for efficiency they are often inefficient as a text entry method. To write a word in English takes 4 to 5 letters on average and each letter involves at least one stroke. A Chinese word consists of one or more Chinese characters and each traditional Chinese character consists of about nine strokes on average (http://technology.chtsai.org/charfreq/). The simplified Chinese characters need fewer strokes but they are still rather complex.

Hence the dilemma of following a convention (which means natural handwriting-based on, for example, the Roman alphabet) or abandoning a convention for a new set of more efficient gestures, such as Unistrokes [38]. A promising solution is to bridge experience from a familiar convention to a more efficient behavior, which we will address in the following sections.

There can also be a middle ground between following conventions for ease of learning and making new designs for other considerations. Graffiti largely follows the Roman letters with simplification and reduction in strokes, so the new gesture set is easier to recognize, faster to write, but still not too difficult to learn and remember. Similarly EdgeWrite [124] approximates the Roman letters in kinematic motion but defines gestures along the edges of a physical square to assist people with motor control disability.

9.2 Making Gestures Guessable and Immediately Usable by Involving End Users Input in Design

As one of the design goals, gestures should be guessable and immediately usable by new users. Applying the previous design principle of making gestures analogous to physical effects and cultural conventions is one way of achieving these goals. Involving end users in defining gestures is another.

There is a body of literature on guessability and immediate usability of gesture interfaces. As reviewed earlier in this monograph, some of the earliest empirical work on stroke gestures focused on users' tendency to use the same gesture for a given function [39, 127]. If most users would intuitively guess/expect the same gesture for a given function, such a gesture would be immediately usable by most users with little learning.

Immediate usability is important to user adoption of gesture sets hence it has been studied for various systems. For example MacKenzie and Zhang [84] tested the immediate usability of Graffiti. Koltringer and Grechenig tested the immediate usability of Graffiti 2 and the virtual keyboard [56]. Wobbrock and Myers presented a design process for maximizing the guessability of a gesture set [121]. They applied such a process to redesign the final EdgeWrite alphabet to increase its immediate usability to a level on par with Graffiti.

Wobbrock et al. [120] used the same guessability-maximization method for designing gestures on tabletop touchscreens. The basic idea of their method was to present study participants with the functions that needed gestures and then ask them to make gestures to accomplish those functions. The final gesture set was selected to maximize users' agreement and resolve conflicts.

On the other hand, user-defined gestures tend to have high agreement on only a small number of gestures that are analogous to their physical effects or cultural conventions including user's previous PC experience. This is quite evident in the empirical results of Wobbrock et al. [120]. User-defined gestures may not be unique to one function and such gestures therefore need to be object-context dependent to remove ambiguity. In sum, while involving end users is a desirable and

often necessary phase of design, the entire gesture interface design often cannot be left to the users alone.

9.3 Making Gestures as Simple as Possible

For efficiency reasons, stroke gestures should be made as simple as possible. From the motor control complexity models of stroke gestures, we learned that the time to produce familiar gestures strongly depends on these gestures' complexity. Tools such as the CLC model can estimate quantitatively the complexity of a gesture stroke and give a good prediction of the gesture's production time, at least relative to other gestures.

For any given application or product, typically a set G of N gestures is needed $\{G_i, i = 1, \ldots, N\}$. For example, a letter-level writing system like English needs at least 26 symbols. Fortunately the frequency of the members in a set can often be estimated from existing usage data, which means we can derive the individual probabilities p_i for the member i. Since it is possible to estimate the time t_i to draw gesture G_i, then the mean efficiency of using gesture set G is simply:

$$t = \sum_{i=1}^{N} p_i t_i. \tag{9.1}$$

This would allow the designer to quickly evaluate two or more sets of alternative gestures from a motor efficiency point of view. Note that one can either tweak the shape of each gesture or move simpler gestures to represent more frequent commands, to optimize according to Equation (9.1).

9.4 Making Gestures Distinct

Gestures in a set should be as different from each other as possible. Well-separated gestures, by definition, increase error tolerance, recognition accuracy, and potentially help the user to remember them better. The visual similarity work of Long et al. [82], reviewed earlier in this monograph, can potentially guide designers to design well-separated gestures. However, doing so without assistance is often difficult because the distance measure between gestures is recognizer dependent and human imagination to create different shapes is often

limited. Indeed, in the study of Long et al. [80], participants were asked to obtain the best recognition accuracy they could by designing a set of stroke gestures for Rubine's recognition algorithm. No participant was able to go beyond a 95.4% recognition rate. A typical problem they observed is that participants tend to add strokes that are too similar to those already defined. A design toolkit that provides interactive assessment of the gesture set is often necessary. We will return to this issue in the gesture toolkit section.

More generally, members of a gesture set should be distinct, complementary, and coherent as a set, to support the user's higher-level tasks. Locally optimizing one gesture for each function at a time, or letting the user self-define them may not be good enough.

9.5 Making Gestures Systematic

One of the classic rules of good design is to project a clear conceptual model to the user, so there is a systematic view for the user to work with. This also applies in gesture design. Word-gesture keyboards, such as ShapeWriter, are systematic in the sense that the gesture for any word always follows the simple rule of traversing letters in the word on a keyboard. The keyboard is the mnemonic map, or crib sheet, for producing any gesture. For example the gesture for the word "word" follows the trace *w-o-r-d*. For any command, the corresponding gesture starts from a special key such as Cmd and traverses the name of the command. For example the gesture for the command *copy* is *Cmd-c-o-p-y*. [60].

It is often difficult to generate all gestures from a single systematic rule. Word-gesture keyboards such as ShapeWriter also have an exception to the simple rule of traversing letters to gesture all words. Uncommon names of people and places, technical jargon and acronyms may not be stored in ShapeWriter's default lexicon. For words that are not stored in the lexicon (i.e., OOV — out of vocabulary), their corresponding gestures are undefined. One would need to type such words once, add them into the lexicon, and gesture them the next time.

Other systematic design of gestures includes Quikwriting [96]. Although the gestures defined in Quikwriting tend to be more complex than ShapeWriter and Quikwriting does not involve pattern recognition

as ShapeWriter does, it is another systematic way of defining gestures for word entry.

Both gesture keyboards and Quikwriting provide a constant visual support (e.g., the keyboard) that occupies screen real estate. It is possible to dynamically reveal the visual support, which we discuss in the next section.

9.6 Making Gestures Self-revealing

When the gesture for a command is not intuitive or guessable from a system, it is important to make it self-revealing or discoverable.

There are a number of approaches to make gestures self-revealing. One is some form of "crib sheet." As described in Kurtenbach and Moran [66], in the Tivoli system if the user does not know which gestures are available or how to gesture a command, the user can press down the stylus and wait for a crib sheet to display available commands and their corresponding gestures. Similarly, the "gesture shortcuts" described earlier in this article use pull-down menus as the crib sheet [10]. One can select a menu item, but the same menu also displays a gesture shortcut to that command, to encourage the user to learn and memorize the faster alternative [10]. Bragdon and colleagues proposed to turn a traditional toolbar into a form of gesture crib sheet called GestureBar [15]. Clicking on a tool in GestureBar does not actually invoke the command but rather opens a window that displays an animation of the gesture for this command and an area for the user to practice. This dynamic crib sheet is more forceful toward the use of gestures than the static pull-down menu since it compels the user to exclusively use gestures by disabling the default toolbar behavior. Another mechanism is the "highlighter hints" used in InkSeine, which help reveal gestures that are drawn on top of a particular portion of a user interface [45].

Marking menus, or radial menus with delayed display [20, 48, 63, 64], are inherently self-revealing. The user gestures the command if the user remembers it. If not, the user presses down the pen and waits for the delayed radial menu to guide the stroke gesture.

A recent and interesting attempt is to make a dynamic guide to gestures. As crib sheets and marking menus do, the OctoPocus guide

[13] displays the set of possible gestures in response to the user's hesitation. With OctoPocus, the set of displayed gestures is reduced to the gesture candidates that match only the partially completed stroke gesture already drawn. As the user draws further, the content of the guide is continuously updated according to the user's input. The implementation of this kind of dynamic guide requires a recognizer for incomplete gestures. This is especially challenging when the system allows users to draw gestures in varying scale. To address this problem, Appert and Bau [8] proposed an algorithm to estimate the scale of an incomplete gesture and showed how to use it to implement a dynamic guide, which supports scale independence. Note that gesture systems such as EdgeWrite use a fixed scale so incremental recognition can be used with less ambiguity. EdgeWrite displays incrementally recognized results including in-stroke word completion as revealing feedback [123, 124].

Gesture keyboards such as ShapeWriter are also self-revealing in the sense that a gesture is constructed from elements (letter keys) constantly displayed on the screen. The extent of relying on the keyboard guidance in producing a gesture is at the user's discretion. If the user does not know or forgets a gesture for a word or a command, the user traces the word by going from letter to letter on the keyboard.

9.7 Supporting the Right Level of Chunking

A very important potential advantage of gestures lies in "chunking" — grouping task elements into a larger inseparable whole. For example, with word-gesture keyboards such as ShapeWriter, the user may initially view and draw each segment of the stroke gesture toward each letter in a word as a separate act. Over time, the user may view and draw the total shape of an entire word's gesture without separating it into segments. For longer words, the trace that links common word fragments, such as *t-i-o-n*, may be chunked together first. Although chunking is a fundamental human behavior, stroke gestures may facilitate such a behavior faster than typing key sequences because of the kinesthetic characteristics of gesturing. To make a stroke gesture, the distinctive state of maintaining pressure or contact with the surface may encourage the user to treat the process as a whole [17].

It is not assumed that gesture systems should always impose the greatest chunks of information possible on the user to process. While larger chunks of information being entered with a single gesture stroke may be efficient from a motor control point of view, this may impose an increasingly higher cognitive and learning burden on the user. For example, it is technically possible to use single strokes in a word-gesture keyboard to enter a whole phrase rather than a single word. In fact a greater degree of error tolerance can be achieved at a phrase level because the larger the language unit is, the more constrained and more distinct it is. However it could be too cognitively demanding to draw a whole phrase without a break. Furthermore, when an error is made, the whole phrase has to be corrected and rewritten, so the cost of error is also higher.

The issue of supporting the right level of chunking also faces the design of marking menus. The original marking menus used zigzag compound gesture marks for hierarchical menus selection [64]. For higher accuracy and speed reasons, it is also possible to use several straight strokes instead of a larger compound mark to invoke a command [139]. The long-term memory and chunking implications of the two choices, however, are unclear.

9.8 Supporting Progression from Ease to Efficiency

Chunking is a means toward efficient interaction. When designing gesture systems, efficiency is an important consideration, but it is also important to make a gesture system easy to learn. Ease (of learning) and efficiency are both critical and desirable qualities of a good user interface.

Unfortunately the ease and the efficiency of a user interface are often at odds with each other since the cognitive factors influencing ease and efficiency are different.

From a user perspective, an easy-to-learn interface should be human-visual recognition-based or display driven. A pull-down menu is an example of an easy-to-learn interface. With such an interface, the user should be able to recognize and select an action from the available choices displayed. It is fundamentally an outside-in, closed-loop, controlled, and

attention-demanding cognitive process. Such a process tends to be slow, but it requires little a priori memory of the action procedure.

In contrast, an efficient interface should be recall-based or memory driven. A keyboard shortcut is an example of an efficient interface. With such an interface, the user needs to execute the action procedure recalled from the user's memory. This is fundamentally an inside-out, open-loop, automatic, and low attention cognitive process. Such a process tends to be fast, but requires memory previously acquired through practice. All highly skilled performances, ranging from fast typing to sports, are of this kind.

For consumer products, interfaces designed only for ease of use tend to triumph over interfaces designed only for efficiency (i.e., high performance). There may be a bias toward short-term savings of learning at the cost of long-term loss in efficiency. This is very similar to, and in fact a version of, the human decision bias in facing alternative "prospects" [53]. When it comes to gaining, people tend to take the safer but less rewarding bet. When two alternative interfaces are available in the same system, users tend to choose and stay on the easy to learn but less efficient path. It is therefore desirable to design strategies to shift users from the easy path, such as the pull-down menus, to the efficient path, such as keyboard shortcuts [41]. However research shows the effective strategies tend to be quite heavy-handed/forceful. For example, disabling menu execution and making menus effectively crib sheets is one such strategy [41].

A solution to the ease versus efficiency conflict is to design "progressive user interfaces," which provide a progression path from ease to efficiency. In other words, the users should be able to start from visual recognition-based, display driven, and controlled behavior to recall-based, memory driven, and automatic behavior.

Cognitive psychology research shows that the key to the progression from a controlled and high-attention cognitive process to an automatic and low-attention process lies in consistent mapping between stimulus and response [105, 104]. However consistency in mapping is often lacking in UI design. For example, for the same task of printing a document, the keyboard shortcut response, $Ctrl + p$ (recall-based) involves a behavior that is completely different from the menu-driven response

(recognition-based) that novice users are used to. In contrast, marking menus and gesture keyboards afford the same action patterns regardless of whether the action is driven from visual guidance or from past memory. The consistency in movement facilitates the transition from recognition to recall, from an outside-in, controlled and closed-loop process to an inside-out, automatic and open-loop process.

However, consistent mapping is only a general principle. The specific user interface mechanisms can still vary greatly. One important issue that is still not well understood is how hard the interface should "push" the user to shift from recognition to recall.

Marking menus, for example, choose to push the user to make that switch by introducing a delay in the pie menu pop-up. Such a time cost may encourage the user to actively recall the gesture action from past memory rather than waiting for visual guidance from the menu display. On the other hand, gesture keyboards such as ShapeWriter, leave the degree of visual guidance from the keyboard relative to the degree of gesture recall from memory completely at the user's will. The user can choose to look more or less from the keyboard for guidance.

Cockburn and colleagues [27] specifically studied the notion of "no-pain no-gain hard lessons" UI design. They conducted two experiments, one with soft keyboard tapping and one with learning gestures on a keyboard. Both of them required the user to scratch off the "frost" that covers the user interface as a penalty of looking at the UI for visual guidance, hence forcing the user to perform as much recall as possible. While such an interface did improve users' spatial memory of a soft (tapping) keyboard, it was simply too hard for learning gestures on a keyboard.

The idea of supporting both novice and expert users, and bridging the two as smoothly as possible, has gone beyond marking menus and SHARK/ShapeWriter gesture keyboards to other gesture methods. For example, with EdgeWrite, users can make character-level unistrokes, but while those strokes are being made, word completions are shown in the four corners of the writing area. An experienced user can make a pigtail loop and then select those corners as part of a single stroke. EdgeWrite calls this "fluid in-stroke completion shorthand," or Fisch, in homage to SHARK [123].

9.9 Providing Game-based Training with Expanding Rehearsal

One strategy to make learning new gestures more fun and engaging is to embed these gestures in game playing. The Graffiti letter writing system, which was central to the initial success of the Palm Pilot PDA, came with a practice game called Giraffe. ShapeWriter, as was released in the iPhone App Store and Android Market, embedded a balloon popping game that Kristensson and Zhai previously researched [61]. This game stands apart from other typing tutor-style games in two aspects. First, it uses the Expanding Rehearsal Interval (ERI) to optimize learning (more on this below). Second, if the user pressed an *Auto* button, the game would demonstrate how to articulate a particular stroke gesture for a word. The benefits of this kind of observational practice have been demonstrated in the psychomotor literature [55]. While a game could make practice more fun, the use of psychological theories and methods in the game design can potentially make learning more efficient and effective (Figure 9.1). There are a variety of theories, models and insights on how people learn and remember skills in the literature of skill acquisition [98, 102] and human memory [12].

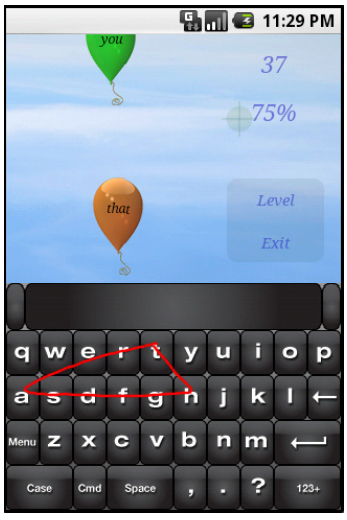


Fig. 9.1 A balloon popping game embedded in a version of Android ShapeWriter. The user has to draw a correct gesture to pop the balloon carrying the target word, to gain points.

A compelling method from that literature is the expanding rehearsal interval (ERI) method [68].

A central issue in traditional training methods is whether the distribution of practice repetition should be massed or evenly distributed. The ERI method is neither totally massed nor evenly distributed. Rather, it optimizes the schedule of practice by increasing the interval between repetitions.

The original ERI method increases rehearsal internally at a fixed rate. For example, it may double the time interval each time an item is practiced. Software driven ERI schedules can be adaptive to actual learning performance [131, 138]. For example, the interval of rehearsal for a particular gesture can expand only if the learner recalls the gesture correctly. Otherwise, the interval can stay the same or even shrink if the learner repeatedly fails to recall a gesture.

In Zhai and Kristensson's initial experiment on gesture keyboard learning [134], the ERI procedure worked roughly as follows. In each cycle of rehearsal of a particular word, the participant was asked to draw the gesture of that word without displaying the keyboard to the user. This forced the user to actively retrieve the gesture from memory. Research has shown that active retrieval is a key to memory retention [102]. The word that matched the user's gesture was then displayed to the user. If correct, the word would be rescheduled to appear at an interval twice the current value. The user could go on to the next word or practice the current word a few more times before moving on. If the participant could not recall or draw the word gesture correctly, the rehearsal interval would keep its current value.

The ERI scheduling program maintained two lists of words: a word list containing all words to be learned and a rehearse list keeping all words that had been rehearsed at various intervals. Each word in the rehearse list had its own timer, counting down from its current rehearsal interval value. The algorithm that managed the ERI scheduling worked as follows:

1. If the rehearse list is empty, pick a new word from the word list and initialize it with a rehearse interval of 30 sec and put it in the rehearse list.

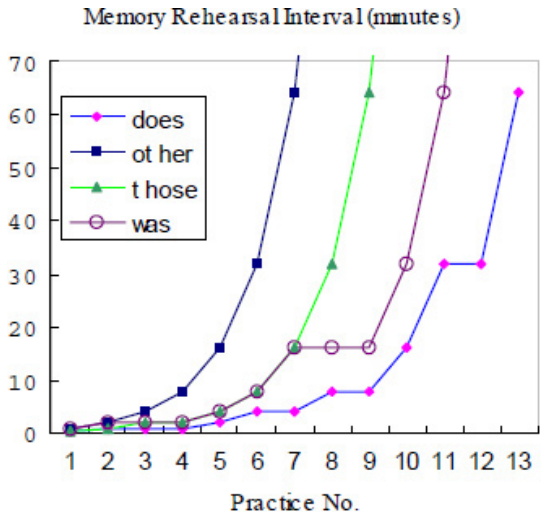


Fig. 9.2 The ERI traces of a few sample words by one participant in Zhai and Kristensson's experiment.

2. Pick the word from the rehearse list that has the earliest rehearse time according to the value left in its timer.
3. If the timer is below 30 sec, present the word to the user. Otherwise pick a new word from the word list and present it to the user.
4. If the user draws the correct gesture of the target word, return it to the rehearse list with a doubled rehearsal interval, else return it with an unaltered rehearsal interval, Go to 2.

As shown in Figure 9.2, in some cases the participant could keep up with the ERI expansion and correctly draw the word gesture every time (See "other" for example), suggesting that the expansion rate as described in the procedure was just right or conservative. In other cases (see "was" for example), the participant missed a step or two, suggesting the expansion rate might be on the aggressive side. Overall, while the Zhai and Kristensson [134] study showed ERI could be effectively adapted to one gesture learning application, its relative efficiency in comparison to other systematic learning methods, or the absence of them, still needs further investigation.

10

The Separation of Command from Scope Selection and Stroke Gesture from Inking

When stroke gestures are used as commands, they can serve different types of purposes. One is to express the action to be applied, such as "delete" or "move." The other is to select or scope the object(s) the action is to be applied to, by for example a lasso gesture that encircles a group of icons. Occasionally it is also necessary to specify the target location (such as "move these words there").

The computer system has to be able to separate selection gestures from command gestures, if they are drawn in different strokes. If they are drawn in the same stroke, the system has to delimit the stroke into selection and command segments.

There are many ways to achieve this separation. One is to design different stroke gestures for selection and for command, which can be difficult because the selection stroke gesture (e.g., a lasso) needs to be flexible in shape in order to enclose different objects to be selected. Another is an explicit mode switch by for example pressing on a button so that the same gesture stroke can continue to the command phase from the selection phase. One can always use a tool palette to switch modes. This is similar to regular drawing or graphics software that changes the cursor's mode depending on which button in the tool palette was clicked on.

To save the round trip of going to a tool palette outside the main operating area, it is also possible to pop up a soft button, a menu, or "a handle" at the end of the selection stroke [63]. Hinckley and colleagues [43] proposed using a pigtail, a small circle at the end of the selection lasso, as a delimiter to switch from selection to commands, which were issued through a marking menu. Such a design enables both selection and command to be chunked as one continuous gesture hence becoming potentially more efficient for proficient users. Another way to achieve selection and command in a single continuous gesture is to pause for a set period to delimit the two phases in the same stroke.

Hinckley and colleagues studied a pigtail, time-out (pause), handle, and physical button as delimiters in a comparative experiment. They found that time-out is slow, not surprisingly given a set time threshold has to be passed in order to continue the stroke; button is error prone due to the need to synchronize the button click with the gesture phase change; handle and pigtail (in another design iteration) were on par with each other in performance. Pigtail was slightly faster in repeated trials, presumably due to the greater degree of chunking. However most of their participants preferred the pop-up handle technique in their study.

For sketching applications, there is also a need to switch between the command gesture mode and the inking (drawing) mode. Li and colleagues [76] did a comprehensive study on methods of switching between the two modes. They found that pressing a button with another hand offered the fastest performance and was most preferred in comparison to pressing the button on the stylus, pressing and holding the stylus tip on the screen to wait for a mode change, pressing the pen tip to a different force level, and flipping the stylus to the eraser head side, as alternative mode-switching mechanisms.

Note that, in many cases, where the gesture is drawn specifies the object. For example if a gesture X (delete) is drawn on a specific icon, that icon will be deleted. In other words, object selection can sometimes be implicit and hence free of additional shape complexity to the gesture per se. Importantly, selection and commands are naturally separated in these cases.

11

Gesture Recognition Algorithms and Design Toolkits

This section reviews two major aspects of stroke gesture interface implementation. One is the design and development of gesture recognition engines and the other is the assignments and linking of stroke gestures to software functions.

11.1 Recognition Algorithms

In principle, gesture recognition is pattern recognition that has been studied via a wealth of approaches and methods for decades. Several excellent introductions to the field exist [33, 101, 110].

Single stroke gesture recognition was a critical part of early handwriting recognition systems. In the 1950s to 1960s such systems used various coding schemes, such as zone coding [32] or chain codes [36] to first encode both the user's input and the stroke gesture templates into digit sequences, and then perform recognition by matching the template whose digit sequence is identical (or nearly identical) to the digit sequence of the user's input [91]. This is essentially the same approach as hashing. The benefit of this approach is that it is very computationally efficient. The downside is that it is not very robust to noise and distortion.

The currently practiced methods of stroke gesture recognition belong to two broad categories. One is data training-based methods and the other is template matching methods descended from early handwriting recognition research.

Data-training approaches comprise machine learning methods such as neural networks, hidden Markov models or covariance matrices. These approaches represent a stroke gesture as an n-dimensional vector and use a training set to partition the n-dimensional space into multiple gesture classes. One method that has gained particularly popularity in HCI research is the recognizer proposed by Rubine [100]. The Rubine recognizer encodes a gesture as a vector of 13 features and uses a covariance matrix to partition this 13-dimensional space. Rubine's recognizer has been used in many gesture research projects [47, 70, 80, 90]. The quality of training is critical for the Rubine's recognizer, or for any other data-training algorithm. To make the recognizer accurate, it must be trained with a rather large set of examples diverse enough to reflect variance across users and contexts of use. It is often difficult to capture naturally occurring variances in conscious articulation of gestures in the laboratory. Training-based recognizers may also be difficult to debug because it is hard for a designer to get a clear mental model of what happens in a black box trained with data encoded in an n-dimensional space.

More recently, template matching-based approaches have regained research attention and applications in stroke gesture recognition [10, 59, 75, 125, 131]. Template-based methods that compute the distance between a user's gesture input and a list of gesture templates are conceptually simple. They were largely abandoned in natural handwriting recognition because the explicit templates may not capture the great amount of variance in people's different writing styles, which evolved over time, although elastic matching can stretch portions of a stroke in order to handle the natural variations in handwriting [109].

In most stroke gesture applications, the gesture sets are often well defined, either by the designer or by the end users. There are indeed clear templates to these gestures. Gestures defined on word-gesture keyboards also have well-defined templates — the stroke connecting letters in the word on the given keyboard layout. ShapeWriter [134]

first applied elastic template matching in its early prototypes but later used proportional shape distance (PSD) as a basic component of a more complex multi-channel recognition system [59].

Later, PSD was adopted, generalized and expanded in the $1 recognizer [125], which was in turn extended to the $N, a multistroke recognizer built upon the $1 [6]. Li further improved the $1 recognizer by measuring the angular difference between the input gesture vector and the template gesture vector [75], which enabled a closed-form solution for the best orientation match between the input and the template gestures. The sampling rate requirement for a PSD-based recognizer can be quite low. It has been shown that for a small gesture set (e.g., 30), six sampling points per gesture was sufficient [114].

The main steps involved in PSD-based template matching are: (1) Sampling both the input gesture and the template gestures into a fixed number of points. (2) (Optionally) translate, scale, and rotate the input gesture relative to the template gestures to minimize their distances. (3) Compute a distance metric between the input gesture and the template gestures. (4) Rank the template matching results according to the distance metric and output the label of the top ranked template.

Appert and Zhai [10] performed an experiment involving a set of 16 gestures, which is small but sufficient for many applications. They showed that a PSD template-matching recognizer yielded the following results: Participants made 7.4% errors on the first attempt using a mouse. Among these failed trials, 73% of them were corrected with the second attempt, 10% with the third attempt, and the remaining 17% with the subsequent attempts. When using a digital pen on a tablet computer, only 3% of the trials failed with the first attempt, of which 76% were corrected with the second attempt, 7% with the third attempt, and the remaining 17% with the subsequent attempts. Most of the errors were made with the three gestures that had small details (e.g., 3 and 13 in Figure 11.1). Removing these gestures from the data collected, the remaining errors were 0.001% among the mouse trials and 0% among the pen trials. Overall, their experiment shows that a PSD template-matching recognizer can work perfectly well for a small gesture set that does not involve very detailed features.

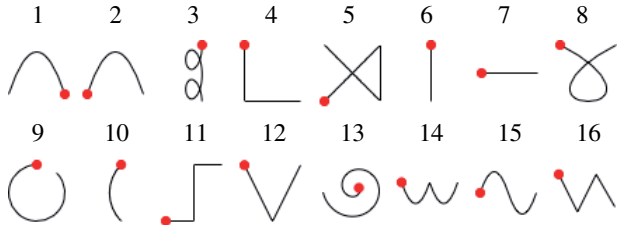


Fig. 11.1 Gestures used in Appert and Zhai's [10] experiment.

Wobbrock, Wilson and Li empirically compared PSD template matching with elastic template matching and Rubine's recognizer and found that the results of PSD template matching were nearly identical to those of elastic template matching and superior to Rubine's recognizer for the same gesture set [125].

It is optional to make template matching rotation, scale or location independent. In terms of the recognition process, this means whether the user-drawn gesture is transformed in orientation, scale, or location to match the templates before the similarity between the input and the templates is computed. These choices depend on the application. For example some applications may require a left to right stroke to be assigned to a different function (aka a gesture label or class) than a right to left stroke, whereas in other applications a straight line stroke may mean the same regardless of orientation.

In one implementation of word-gesture keyboards, Kristensson and Zhai [59] used both scale and location independent template matching (called the shape channel) and scale and location dependent template matching (called the location channel). The goal was to accommodate both beginners and advanced users. Beginners are likely to look closely at the letters on the keyboard and draw word gestures accordingly, whereas advanced users are likely to remember some and parts of the gestures and draw gestures more according to their shapes. The weights of the two channels can be dynamically adjusted according to the user's gesture speed. Faster gestures are more likely produced by shape recall, hence more compatible with the shape channel. Conversely, slow gestures are more likely produced by letter tracing, hence more compatible with the location channel. Note that the speed measured there was relative to the target word's tapping

speed as predicted by Fitts' law [35]. Surpassing Fitts' law prediction was interpreted as more shape-recall based.

When the number of templates is very large, for example in gesture keyboards, it is impossible to thoroughly compute the user's input against all templates. Pruning techniques that reject templates that are far away from the user's input, based on coarse and simple features, can be used to reduce the number of templates to a smaller set for complete matching calculation. For example, a simple pruning technique is to downsample both the user's input and the templates of stroke gestures into a very few sampling points and then discard templates whose distance to the user's input is larger than a set threshold.

As shown in the Appert and Zhai experiment [10], PSD template matching tends to be insensitive to small details. For example, a small pigtail as part of a larger gesture is very noticeable to human perception, but does not make a big difference to PSD template matchers. A future research direction would be to combine training-based methods with template matching in order to take advantage of both approaches, particularly when a large body of high-quality representative training data are available.

Gesture recognition can be either post hoc (recognizing a gesture after it is completely drawn) or incremental (recognizing a gesture as it is drawn in real time). The latter is more challenging but sometimes necessary to provide real-time feedback or forward projection during gesture production.

11.2 Stroke Gesture Toolkits

Software toolkits have greatly decreased the programming effort and reduced the cost of implementing conventional graphical interfaces. Today, menu-based interfaces can be relatively easily developed thanks to the support of toolkits such as Java Swing, Qt, and Cocoa. However programming with these toolkits uses standard widgets. A standard widget is a graphical component that is location (or visual-spatially) dependent. It only responds to mouse or keyboard events on the widget. As we discussed earlier, gestures may or may not be visual-spatially dependent. Often it is desirable to have visual-spatially independent gestures that can be drawn anywhere on the screen. In

order to ease gesture interface programming, several research toolkits have been proposed to support different stages of the development process.

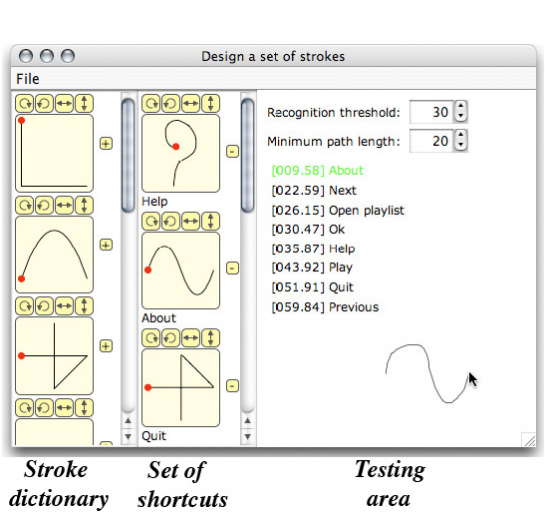
First, gesture interface toolkits can embed recognition algorithms to save the developer's effort since implementing recognition algorithms requires specialized expertise. ArtKit [42], Garnet [69] and later SATIN [47], iGesture [106] and SwingStates [9] all contain a Rubine recognizer. They all encapsulate the recognizer in a modular and flexible way so that it can be replaced by another recognizer without modifying the whole toolkit. Most of them define a recognizer as an object that takes a stroke as input and outputs an ordered list of gesture classes. ArtKit uses the concept of a recognition object, which is a combination of a recognition algorithm and a particular gesture class that computes a binary output. iGesture takes a combination of recognizers to compute the n-best list. All these toolkits provide pre-processing and segmentation functions that turn raw gesture input into higher level stroke events. In other words they extend the standard GUI input vocabulary so that toolkit users can deal with stroke events rather than low-level 2D sampling points.

Second, gesture toolkits can include a training and design support environment for interactive gesture design and experimentation. The users of the toolkits can perform fast trial-and-error design iterations in the environment. With Quill in SATIN [81], the management console in the iGesture framework [106] or the SwingStates' training application [9] it is possible to define mappings between strokes and commands and test recognition accuracy by drawing in a dedicated area with immediate recognition results. If the accuracy is not satisfactory, some toolkits propose alternative recognition algorithms (iGesture, SwingStates), advise the users on making the gestures more different based on recognition performance (Quill) or edit gesture examples for data-training algorithms. Potentially, tools like these can also embed motor control complexity models and visual similarity models so the developers can quickly evaluate more ramifications of the stroke gestures selected.

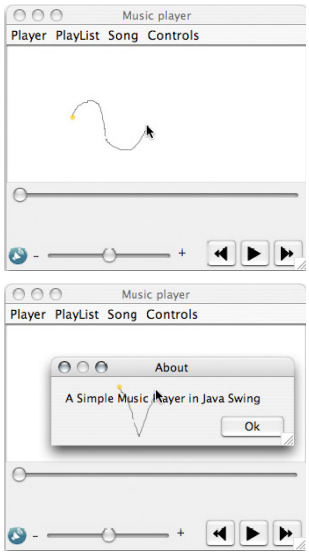
In addition to stroke-to-command mapping and recognition engine selection and tuning, gesture toolkits also need to support the design and implementation of gesture interface features such as ink render-

ing, feedback, and previews to enable gesture discovery. SATIN is a good example of a more general purpose toolkit in that regard. It is intended to support a wide range of informal ink-based applications so it does not pre-program many specific behaviors but rather offers flexibility in how ink is processed. In contrast, The Stroke-Shortcut Toolkit (SST) [10] is intended to support adding gesture stroke-based shortcuts to commands of a Java Swing graphical interface. Because of its narrower focus, SST reduces to the very minimum the amount of programming needed to add gesture shortcuts to Java applications. Here we review SST in some detail.

Users of SST can invoke the Design Shortcuts environment on any Java Swing interface. Figure 11.2 shows this application, which has a "gesture stroke dictionary" on its left panel, the set of shortcuts added to the Java Swing interface on the middle panel, and a testing area on the right panel (Figure 11.2(a)). To define a new gesture, the developer clicks on the "+" button displayed to the right of a gesture



(a) The Design Shortcuts Environment in SST



(b) Testing shortcuts in their real context

Fig. 11.2 The SST stroke gesture toolkit.

stroke in the dictionary. This pops up the list of commands found in the attached Java Swing interface. The developer can now pick one of these commands in the list and use the testing area to assess the recognition accuracy. If the Java Swing interface runs at the same time (and this is achieved by a simple line of code), shortcuts are also added to the interface so they can be tested in their real context (Figure 11.2(b)). The developer has little more to do to implement stroke shortcuts in her application. Ink feedback, stroke morphing and help techniques for end users (e.g., stroke previews in menu items and tooltips) can also be added with a single line of code.

An interesting feature in SST is the guidance provided to define mappings between stroke shape and command name. Figure 11.2(a) shows the gesture stroke dictionary that contains an initial set of nine predefined strokes for the developer to choose from. With these predefined strokes, the developer can already define a large set of gesture commands by combining several of these strokes and/or applying geometrical transformations to them. One can use the transformation buttons displayed on top of each gesture to rotate or mirror (horizontally or vertically) a gesture before adding it to the set of shortcuts. One can also select several strokes to build a new gesture that is the concatenation of the selected strokes. Compared to starting with a "blank page," providing a set of primitive strokes and a set of operations on these primitives afford a structured design space that can be systematically explored. Developers can also draw a stroke and add it to the gesture stroke dictionary.

12

Evaluation of Stroke Gesture Interfaces

The role of evaluation and the ways of conducting it within the field of HCI is often discussed and debated among researchers and practitioners [40, 77, 94, 131]. In our view, evaluation of interactive systems in general is always essential but rarely definitive. As a result the evaluation methods can and should be varied and broadly defined. Evaluation is often narrowly viewed as simple empirical user studies involving a group of users to perform a task and measuring the speed or accuracy of the users as a function of the design choices. However evaluation can and should take many of the following forms, all with pros and cons and all should be applied and interpreted according to application context.

- Conceptual analysis. Why the system is good or bad, what features and effects does it have or not have?
- Mathematical analysis. Some aspects of a system, such as the amount of movement the user has to make, the mean time to generate an output, can often be mathematically calculated with the help of well-established empirical laws.
- Actual use by designers and developers. Often known as "eat your own dog food." This could uncover many unexpected

problems, but they are also limited to the designers' and developers' own experiences and biases.
- Trial users and log analysis.
- Lab experiments, producing both performance and questionnaire results.
- In-context observations and field studies.

There are a number of particularly difficult challenges in evaluating stroke gesture interfaces. One reason is that the speed-accuracy tradeoff function in stroke gesture production is not well understood or modeled. For example a 10% less precise stroke gesture is faster to produce, but we do not have models to predict how much faster. Typically participants are expected to perform an experimental task "as fast as possible and as accurately as possible" so they operate at their own speed-accuracy tradeoff point. However without a speed-accuracy tradeoff model it is often difficult to interpret experimental results. For example if System A were measured 10% faster but 5% less accurate than System B, we could not infer if A were still faster than B if it were held at the same accuracy level at which System B performed. A simplified but certainly incomplete approach to stroke gesture evaluation is to aim imprecision at a "reasonable" level, such as 10% error, as measured by, for example, PSD distance, and ask the participants to perform near that set level of imprecision. Speed measured under such a condition might be more comparable and meaningful than speed measured with floating and varied imprecision. How to more completely evaluate stroke gesture interface against speed-accuracy tradeoff is still an open topic.

Another particular challenge to stroke gesture interface evaluation is learning. Performance is surely a function of learning, which typically follows a power law curve. Classic manual-skill studies show how humans could continuously improve, with decreasing increment from one trial to the next and eventually approach a machine or system limit [30]. Ideally we should observe and measure participants' learning curve over a long period of time, producing initial (e.g., first half hour of use) performance, and practically saturated ceiling performance, and the time period from one level of performance to another. Often this is

impractical with complex systems (e.g., gesture keyboards, which may take weeks or months of time, if not more, for a user to become proficient with thousands or tens of thousands of gestures). There are a number of incomplete but still informative approaches to evaluate performance as a function of learning. To measure initial performance can be informative in and of itself. To complement it with an empirically estimated ceiling performance is even better, although the learning time from one performance level to another would still be missing.

One way to empirically estimate ceiling performance is as follows. Pick a small set of representative or high frequency stroke gestures and ask participants to repeatedly produce them until their performance is "saturated." Then calculate the mean ceiling performance based on the results of these gestures weighted by their frequency. This approach was used in [57] to evaluate the ceiling performance of the gesture keyboard ShapeWriter.

13

From Research to Adoption

Although stroke gesture research has had a long history, mainstream applications of gesture UI have been sporadic. The types of stroke gestures in the current generation of smart phones are still relatively simple and limited. This section analyzes the barriers and possible solutions to realizing stroke gesture interface's full potential.

13.1 Qwertynomics and the "Workstation" Hardware

The path dependence theory of socio-technical system development, also known as Qwertynomics [31], argues that a sub-optimal solution, once taken hold, cannot be replaced with optimal solutions. Prime examples of Qwertynomics include the QWERTY keyboard as opposed to the Dvorak simplified keyboard and VHS as opposed to the Beta videotape format. Although this argument has been debated [78], users' existing skills and familiarity with the current technology, and an ecosystem that evolved around the practice, do form barriers for new technologies to overcome. In HCI, the display, keyboard and mouse-based "workstation" format has repelled other alternatives in the past two decades. For the most part, PCs and laptops in such a format have

been good enough for quite some time. However, such a format is particularly unsuited to gesture interface. Drawing on a desktop or laptop screen for a sustained period of time is often fatiguing and less convenient than mouse pointing.

However the IT industry and the HCI field have long realized that the form of interacting with computers and information would move beyond the basic desktop [88]. Indeed, over the past few years, new touchscreen-based products have taken off rapidly, enabling gesture-based interaction methods.

13.2 Learning

Touchscreen hardware alone is not a sufficient reason to adopt gesture-based interaction beyond the very basic gestures such as scrolling, panning, and zooming. In fact, menu selection adapted to a finger friendly size and format is likely to remain the most basic interaction method, for the simple reason that these visual recognition-based actions require little learning.

However, visual recognition-based interaction does suffer from efficiency drawbacks. This is particularly acute when the screen is small. For example flipping through a large number of pages to look for one function is barely acceptable. Gesture interaction offers higher efficiency and larger capacity on a limited screen space. As we have discussed, it is very important to make gesture interface "progressive," from an easy start to an eventually higher performance.

13.3 Backward Compatibility

A more controversial issue is to make UIs "backward compatible" with more traditional menu-driven UIs. Providing gesture shortcuts to traditional menus and buttons is one such method. For example to enable or disable the 3G network on the iPhone one has to quit the current application, find "Settings" on the home screen, tap it, find and tap on "General," find and tap on "Network," and tap the on/off switch there. After that the user has to press the home button, and find and tap the application to return. If there is a gesture-shortcut alternative,

at least the more advanced user would not have to take the multi-step navigation approach. Importantly the gesture alternative should not get in the way of novice users. Similarly, word-gesture keyboard users should always be able to use the traditional hunt and tap approach, one letter at a time. When the user-realized gesture-based "shape writing" is more efficient or preferable, the user can start to gesture on the keyboard without even switching modes.

14

Summary and Conclusions

We have reviewed and synthesized a body of research on stroke gesture interfaces. The synthesis is primarily focused on our own research but we also touched on the work of many other researchers. In this concluding section, we summarize key concepts and observations, main take-away conclusions, and forward-looking calls to actions.

First, stroke gestures fall into a multiple dimensional design space. The dimensions we identified include analogue versus abstract stroke gestures, stroke gestures representing symbols (particularly text) versus commands, order of complexity of stroke gestures, degree of visual-spatial dependency, and implementation (finger versus stylus) and its associated sensor type. Each gesture system, such as the Apple iOS interface, the Graffiti text entry method for Palm devices, marking menus and the SHARK/ShapeWriter word-gesture keyboard, comprised a subspace in this multiple dimensional design space.

Although still limited, there is a body of scientific knowledge that is beginning to address the human factors and cognitive issues of stroke gestures, ranging from the motor control, to the visual, and to the memory aspects of human performance. Early basic human factors and usability studies on stroke gestures identified consistent patterns across

users, in gestures they would anticipate for simple functions, such as move and delete. An important gesture performance topic is modeling a gesture's complexity so gestures or a set of gestures can be quantitatively optimized. The models that may apply here range from counting the number of line segments that can approximate a gesture to the CLC model that breaks down a gesture into curves, lines and corners, each modeled by a lower order model.

On the visual side, research has identified key computation features that can characterize the visual similarities of stroke gestures to users. Research has also shown that visual feedback has impact on some aspects of gestures such as size and closure (or, more generally, referential features), but not on global features such as the overall shape distance. Audio feedback has even less of an impact on the process of articulating gestures but it can help to inform the product of a gesture hence reducing the visual demand of a gesture interface. Both visual and audio feedback can potentially enhance the subjective and emotional aspects of gesture experience.

In comparison to point-and-click graphical interfaces (or tapping — zero order gestures on touchscreens), gestures interfaces can be, optionally, made visual-spatially independent. Such independency means different functions can be activated with differently shaped gestures in the same place (or anywhere), resulting in three advantages — space saving, direct/random access to a large number of functions, and lowered visual attention demand — all may be desirable on a mobile device. The challenge, however, is how users can learn and memorize such gestures. We have shown that in one study users could learn 15 gestures on an unfamiliar word-gesture keyboard per 40 minutes of practice. We have also shown that 200% more gestures than keyboard shortcuts were memorized with the same amount of practice. Gestures, even if arbitrarily defined, afford the user the opportunity to elaborate and more deeply encode their assigned meaning than keyboard shortcuts do. Furthermore, in contrast to the visual icons vs. keyboard shortcuts dichotomy, it is possible to design gesture interfaces that use the same motor control patterns between a visually guided process and a memory recall-driven process. Such motor control constancy facilitates

skill progression. Marking menus and word-gesture keyboards are two examples of this.

There is a wide range of design principles for creating stroke gesture interfaces. These include making gestures analogous to physical effects or cultural conventions, gestures simple and distinctive, defining stroke gestures systematically, making them self-revealing, supporting chunking, and facilitating progress from visually guided performance to recall-driven performance. We also discussed how to support learning via game-based training programs and highlighted the usefulness of the expanding rehearsal interval algorithm for this purpose.

There are important implementation aspects of stroke gestures. We outlined the issues involved in separating commands from scope selection, and inking from stroke gesturing. We gave an overview of recognition algorithms and approaches for classifying stroke gestures, and toolkits that can aid implementers and designers.

As in any research field, research in gesture interfaces does not always impact mass products and the society. We raised three types of challenges that the research field faces in translating research results into products. First, the Qwertynomics effect helps conventional technologies that are "good enough" to prevail. The dominance of icon- and selection-based interfaces is likely to continue as a result. Second, user learning, even if paid off rather quickly in efficiency, remains a hurdle to gesture interface adoption. Third, gesture interfaces may need to be backward compatible with pointing or tapping types of interfaces.

Finally, gesture interfaces need to be evaluated with various evaluation methods, such as conceptual analysis, mathematical analysis, controlled experiments, and studies of logs and actual deployments. It is important each design choice is evaluated in its application context.

Through this survey we have demonstrated that stroke gestures have a deep and fundamental role in both user interface research and in consumer product development. Contribution from research to practice can span a wide range. At one end of the spectrum is addressing basic human performance questions, such as developing accurate models of the motor complexity of stroke gestures. At the other end, the

research field can contribute better engineering solutions, such as new recognizers, toolkits and systems.

We see many opportunities and challenges open for future research. By way of example, we name only the following four call-to-action topics. First, a strong model of stroke gesture complexity would contribute both theoretically and practically to the field. It could either be simpler than existing models or have a closed form, rather than being a combination of algorithms or equations for different types of strokes or stroke segments. Most likely it should reflect some form of entropy measure of the stroke gestures. Second, a model of the capacity, density and bandwidth of stroke gesture systems is currently lacking. Such a model could aid us in understanding the speed-accuracy tradeoff in stroke gesture interfaces, or, more broadly, the gain and cost in using machine intelligence in user interfaces. Third, we need deeper theoretical, empirical, and comparative studies in a variety of settings about the methods and approaches, to help users adopt and learn advanced gestures. These methods include crib sheets, marking menus, tracing-to-gesture progressions, as in ShapeWriter-like systems, and potential other designs. Fourth and more generally, we need to understand more deeply human memory and the skill acquisition mechanisms involved in gesture interaction.

Acknowledgments

We thank Ben Bederson, the editor-in-chief of *Foundations and Trends in Human–Computer Interaction*, for initiating the idea of writing this review and for his suggestions as to its scope and emphases. We thank FnT editor James Finlay for his patient and persistent support, encouragement, and "nagging" without which we could not have done this work. Most of the original research that laid the foundation of this current synthesis was conducted at the IBM Almaden Research Center where we were Research Staff Member, graduate intern, postdoctoral fellow, visiting researcher, and graduate intern, respectively. The scientific atmosphere, helpful and insightful colleagues, and management support for basic research there made this work possible. We are also grateful to our current respective employers and colleagues for their support. We are indebted to three outstanding colleagues in the gesture interface research field, Ken Hinckley, Jacob O. Wobbrock, and Yang Li, who made extensive, constructive, and insightful comments, criticisms and suggestions throughout this monograph. This monograph was revised multiple times according to these comments, some of which were quite directly incorporated into the final version. Of course we authors are solely responsible for the errors, omissions, limitations and debatable positions that remain in this work.

References

[1] J. Accot, "Les Tâches Trajectorielles en Interaction Homme-Machine — Cas des tâches de navigation," Unpublished Ph.D. Thesis, Université de Toulouse 1, Toulouse, France, 2001.

[2] J. Accot and S. Zhai, "Beyond Fitts' law: Models for trajectory-based HCI tasks," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 295–302, 1997.

[3] J. Accot and S. Zhai, "Performance evaluation of input devices in trajectory-based tasks: An application of steering law," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 466–472, 1999.

[4] J. Accot and S. Zhai, "More than dotting the i's — foundations for crossing-based interfaces," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 73–80, Minneapolis, Minnesota, USA, 2002.

[5] T. H. Andersen and S. Zhai, "Writing with music: Exploring the use of auditory feedback in gesture interfaces," *ACM Transactions on Applied Perception*, vol. 7, no. 3, pp. 1–24, 2010.

[6] L. Anthony and J. O. Wobbrock, "A lightweight multistroke recognizer for user interface prototypes," in *Proceedings of Graphics Interface (GI)*, pp. 245–252, Ottawa, Ontario, Canada, 2010.

[7] G. Apitz and F. Guimbretière, "CrossY: A crossing-based drawing application," in *Proceedings of the Annual ACM Symposium on User Interface and Software Technology (UIST)*, pp. 3–12, 2004.

[8] C. Appert and O. Bau, "Scale detection for a priori gesture recognition," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 879–882, Atlanta, Georgia, USA, 2010.

[9] C. Appert and M. Beaudouin-Lafon, "SwingStates: Adding state machines to Java and the swing toolkit," *Software Practice and Experience*, vol. 38, no. 11, pp. 1149–1182, 2008.

[10] C. Appert and S. Zhai, "Using strokes as command shortcuts: Cognitive benefits and toolkit support," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 2289–2298, Boston, MA, USA, 2009.

[11] J. Arvo and K. Novins, "Fluid sketches: Continuous recognition and morphing of simple hand-drawn shapes," in *Proceedings of the Annual ACM Symposium on User Interface Software and Technology (UIST)*, pp. 73–80, San Diego, California, USA, 2000.

[12] A. Baddeley, *Human Memory — Theory and Practice*. Boston: Allyn and Bacon, Revised ed., 1998.

[13] O. Bau and W. E. Mackay, "OctoPocus: A dynamic guide for learning gesture-based command sets," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 37–46, Monterey, CA, USA, 2008.

[14] O. Bau, I. Poupyrev, A. Israr, and C. Harrison, "TeslaTouch: Electrovibration for touch surfaces," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 283–292, New York City, New York, USA, October 3–6 2010.

[15] A. Bragdon, R. Zeleznik, B. Williamson, T. Miller, and J. J. LaViola, "GestureBar: Improving the approachability of gesture-based interfaces," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 2269–2278, 2009.

[16] B. Buxton, "Marking interfaces," Chapter 13 of "Human Input to Computer Systems: Theories, Techniques and Technology," http://www.billbuxton.com/inputManuscript.html. Unpublished manuscript, 2010.

[17] W. Buxton, "Chunking and phrasing and the design of human-computer dislogues," in *Proceedings of the IFIP World Computer Congress*, pp. 475–480, Dublin, Ireland, 1986.

[18] W. Buxton, "A three-state model of graphical input," in *Proceedings of Human-Computer Interaction (INTERACT)*, pp. 449–456, Amsterdam, The Netherlands, 1990.

[19] W. Buxton, R. Hill, and P. Rowley, "Issues and techniques in touch-sensitive tablet input," *Computer graphics, Proceedings of SIGGRAPH*, vol. 19, pp. 215–224, 1985.

[20] J. Callahan, D. Hopkins, M. Weiser, and B. Shneiderman, "An empirical comparison of pie vs. linear menus," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, Washington, D.C., USA, 1988.

[21] X. Cao and S. Zhai, "Modeling human performance of pen stroke gestures," Research Report No. RJ10392. San Jose, CA: IBM Research Report RJ10392, 2006.

[22] X. Cao and S. Zhai, "Modeling human performance of pen stroke gestures," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, San Jose, CA, USA, 2007.

[23] S. J. Castellucci and I. S. MacKenzie, "Graffiti vs. unistrokes: An empirical comparison," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 305–308, Florence, Italy, 2008.

[24] B.-W. Chang and D. Ungar, "Animation: From cartoons to the user interface," in *Proceedings of the ACM Symposium on User Interface Software and Technology (USIT)*, Atlanta, Georgia, USA, 1993.

[25] S. Chatty and P. Lecoanet, "Pen computing for air traffic control," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Vancouver, British Columbia, Canada, 1996.

[26] E. Clarke, "Rhythm and timing in music," in Chapter 13*: The Psychology of Music*, pp. 473–500, Academic Press, 1999.

[27] A. Cockburn, P. O. Kristensson, J. Alexander, and S. Zhai, "Hard lessons: Effort-inducing interfaces benefit spatial learning," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, San Jose, California, USA, 2007.

[28] F. Coulmas, *The Writing Systems of the World*. Oxford: Blackwell, 1989.

[29] F. Craik and R. Lockhart, "Levels of processing: A framework for memory research," *Journal of Verbal Learning and Verbal Behavior*, vol. 11, pp. 671–684, 1972.

[30] E. R. F. W. Crossman, "A theory of acquisition of speed-skill," *Ergonomics*, vol. 2, no. 2, pp. 153–166, 1959.

[31] P. A. David, "Clio and the economics of QWERTY," *American Economic Review*, vol. 75, pp. 332–337, 1985.

[32] T. L. Dimond, "Devices for reading handwritten characters," in *Proceedings of Eastern Joint Computer Conference of American Federation of Information Processing (EJCC)*, pp. 232–237, 1957.

[33] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. New York: John Wiley & Sons, 2001.

[34] P. Ekman and W. Friesen, "Hand movements," *Journal of Communications*, vol. 22, pp. 353–374, 1972.

[35] P. M. Fitts, "The information capacity of the human motor system in controlling the amplitude of movement," *Journal of Experimental Psychology*, vol. 47, pp. 381–391, 1954.

[36] H. Freeman, "On the encoding of arbitrary geometric configurations," *IRE Transactions on Electronic Computers*, vol. 10, no. 2, pp. 260–268, 1961.

[37] J. J. Gibson, *The Ecological Approach to Visual Perception*. Boston: Houghton Mifflin Company, 1979.

[38] D. Goldberg and C. Richardson, "Touch-typing with a stylus," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (INTERCHI)*, pp. 80–87, Amsterdam, The Netherlands, 1993.

[39] J. D. Gould and J. Salaun, "Behavioral experiments on handmarkings," *ACM Transactions on Information Systems*, vol. 5, no. 4, pp. 358–377, 1987.

[40] S. Greenberg and B. Buxton, "Usability evaluation considered harmful (some of the time)," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 111–120, Florence, Italy, 2008.

[41] T. Grossman, P. Dragicevic, and R. Balakrishnan, "Strategies for accelerating on-line learning of hotkeys," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 1591–1600, San Jose, California, USA, 2007.

[42] T. R. Henry, S. E. Hudson, and G. L. Newell, "Integrating gesture and snapping into a user interface toolkit," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 112–122, 1990.

[43] K. Hinckley, P. Baudisch, G. Ramos, and F. Guimbretiere, "Design and analysis of delimiters for selection-action pen gesture phrases in Scriboli," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 451–460, 2005.

[44] K. Hinckley, K. Yatani, M. Pahud, N. Coddington, J. Rodenhouse, A. Wilson, H. Benko, and B. Buxton, "Pen + touch = new tools," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 27–36, 2010.

[45] K. Hinckley, S. Zhao, R. Sarin, P. Baudisch, E. Cutrell, and M. Shilman et al., "InkSeine: In situ search for active note taking," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 251–260, San Jose, California, USA, 2007.

[46] C. Holz and P. Baudisch, "Understanding touch," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 2501–2510, Vancouver, Canada, 2011.

[47] J. I. Hong and J. A. Landay, "SATIN: A toolkit for informal ink-based applications," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 63–72, San Diego, California, USA, 2000.

[48] D. Hopkins, "The design and implementation of pie menus," *Dr. Dobb's Journal*, vol. 16, no. 12, pp. 16–26, 1991.

[49] T. Igarashi, S. Matsuoka, and H. Tanaka, "Teddy: A sketching interface for 3D freeform design," in *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics and Interactive Techniques*, 1999.

[50] P. Isokoski, "Model for unistroke writing time," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 357–364, Seattle, Washington, USA, 2001.

[51] G. Johnson, M. D. Gross, J. Hong, and E. Y.-L. Do, "Computational support for sketching in design: A review," *Foundations and Trends in Human-Computer Interaction*, vol. 2, no. 1, 2008.

[52] M. Johnson, *The Body in the Mind: The Bodily Basis of Meaning, Imagination, and Reason*. University of Chicago Press, 1990.

[53] D. Kahneman and A. Tversky, "Prospect theory: An analysis of decision under risk," *Econometrica*, vol. 47, no. 2, pp. 263–292, 1979.

[54] J. Kaplan, *Startup: A Silicon Valley Adventure*. New York: Penguin, 1996.

[55] R. M. Kohl and C. H. Shea, "Pew(1966) revisited: Acquisition of hierarchical control as a function of observational practice," *Journal of Motor Behavior*, vol. 24, no. 3, pp. 247–260, 1992.

[56] T. Költringer and T. Grechenig, "Comparing the immediate usability of Graffiti 2 and virtual keyboard," in *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 1175–1178, Vienna, Austria, April 24–29 2004.

[57] P. O. Kristensson, "Discrete and continuous shape writing for text entry and control," Ph. D. Thesis, Linköping University, Sweden, 2007.

[58] P. O. Kristensson and L. C. Denby, "Continuous recognition and visualization of pen strokes and touch-screen gestures," in *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling (SBIM)*, pp. 95–102, New York, NY, USA: ACM, 2011. Doi=10.1145/2021164.2021181 http://doi.acm.org/10.1145/2021164.2021181.

[59] P. O. Kristensson and S. Zhai, "SHARK^2: A large vocabulary shorthand writing system for pen-based computers," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 43–52, 2004.

[60] P. O. Kristensson and S. Zhai, "Command strokes with and without preview: Using pen gestures on keyboard for command selection," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 1137–1146, San Jose, CA, USA, 2007.

[61] P. O. Kristensson and S. Zhai, "Learning shape writing by game playing," in *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 1971–1976, San Jose, CA, USA, 2007.

[62] G. Kurtenbach and B. Buxton, "GEdit: A test bed for editing by contiguous gestures," *SIGCHI Bulletin*, vol. 23, no. 2, pp. 22–26, 1991.

[63] G. Kurtenbach and W. Buxton, "Issues in combining marking and direct manipulation techniques," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 137–144, 1991.

[64] G. Kurtenbach and W. Buxton, "User learning and performance with marking menus," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 258–264, 1994.

[65] G. Kurtenbach, G. W. Fitzmaurice, R. N. Owen, and T. Baudel, "The hotbox: Efficient access to a large number of menu-items," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, Pittsburgh, Pennsylvania, USA, 1999.

[66] G. Kurtenbach and T. P. Moran, "Contextual animation of gestural commands," *Eurographics Computer Graphics Forum*, vol. 13, no. 5, pp. 305–314, 1994.

[67] G. Kurtenbach, A. Sellen, and W. Buxton, "An empirical evaluation of some articulatory and cognitive aspects of marking menus," *Human-Computer Interaction*, vol. 8, no. 1, pp. 1–23, 1993.

[68] T. K. Landauer and R. A. Bjork, "Optimum rehearsal patterns and name learning," in *Practical Aspects of Memory*, (M. M. Gruneberg, P. E. Morris, and R. N. Sykes, eds.), pp. 625–632, London: Academic Press, 1978.

[69] J. A. Landay and B. A. Myers, "Extending an existing user interface toolkit to support gesture recognition," in *Proceedings of INTERACT '93 and CHI '93 Conference Companion on Human Factors in Computing Systems*, pp. 91–92, Amsterdam, The Netherlands, 1993.

[70] J. A. Landay and B. A. Myers, "Interactive sketching for the early stages of user interface design," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 43–50, Denver, Colorado, USA, 1995.

[71] E. Lank and E. Saund, "Sloppy selection: Providing an accurate interpretation of imprecise selection gestures," *Computers and Graphics*, vol. 29, no. 4, pp. 490–500, 2005.

[72] S. Lee and S. Zhai, "The performance of touch screen soft buttons," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 309–318, 2009.

[73] A. Leganchuk, S. Zhai, and W. Buxton, "Manual and cognitive benefits of two-handed input: An experimental study," *ACM Transactions on Computer-Human Interaction*, vol. 5, no. 4, pp. 326–359, 1998.

[74] Y. Li, "Gesture search: A tool for fast mobile data access," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 87–96, 2010.

[75] Y. Li, "Protractor: A fast and accurate gesture recognizer," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 2169–2172, Atlanta, Georgia, USA, 2010.

[76] Y. Li, K. Hinckley, Z. Guan, and J. A. Landay, "Experimental analysis of mode switching techniques in pen-based user interfaces," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 461–470, Portland, Oregon, USA, 2005.

[77] H. Lieberman, *The Tyranny of Evaluation, CHI Place Essay*. 2003. Retrieved 2/5, 2010 http://web.media.mit.edu/~lieber/Misc/Tyranny-Evaluation.html.

[78] S. Liebowitz and S. E. Margolis, "Typing errors," *Reason Magazine*, http://reason.com/9606/Fe.QWERTY.shtml, 1996.

[79] S. J. Liebowitz and S. E. Margolis, "The fable of the keys," *Journal of Law and Economics*, vol. 33, no. 1, pp. 1–25, 1990.

[80] A. C. Long, J. A. Landay, and L. A. Rowe, "Implications for a gesture design tool," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 40–47, 1999.

[81] A. C. Long, L. A. Landay, and L. A. Rowe, "Those look similar! Issues in automating gesture design advice," in *Proceedings of the Workshop on Perceptive User Interfaces (PUI)*, pp. 1–5, Orlando, Florida, USA, 2001.

[82] A. C. Long, L. A. Landay, L. A. Rowe, and J. Michiels, "Visual similarity of pen gestures," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 360–367, 2000.

[83] G. Lucchese, M. Field, J. Ho, R. Gutierrez-Osuna, and T. Hammond, "GestureCommander: Continuous touch-based gesture prediction," in *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI EA)*, pp. 1925–1930, New York, NY, USA, 2012. DOI=10.1145/2212776.2223730 http://doi.acm.org/10.1145/2212776.2223730.

[84] I. S. MacKenzie and S. X. Zhang, "The immediate usability of Graffiti," in *Proceedings of Graphics Interface (GI)*, pp. 129–137, Kelowna, British Columbia, Canada, 1997.

[85] J. Mankoff and G. D. Abowd, "Cirrin: A word-level unistroke keyboard for pen input," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST),* Technical Note, pp. 213–214, 1998.

[86] M. S. Mayzner and M. E. Tresselt, "Tables of single-letter and digram frequency counts for various word-length and letter-position combinations," *Psychonomic Monograph Supplements*, vol. 1, no. 2, pp. 13–32, 1965.

[87] S. Mitra and T. Acharya, "Gesture recognition: A survey," *IEEE Transactions on Systems, Man and Cybernetics — Part C: Applications and Reviews*, vol. 37, no. 3, pp. 311–324, 2007.

[88] T. P. Moran and S. Zhai, "Beyond the desktop metaphor in seven dimensions," in *Beyond the Desktop Metaphor — Designing Integrated Digital Work Environments*, (V. Kaptelinin and M. Czerwinski, eds.), pp. 335–354, The MIT Press, 2007.

[89] P. Morrel-Samuels, "Clarifying the distinction between lexical and gestural commands," *International Journal of Man-Machine Studies*, vol. 32, no. 5, pp. 581–590, 1990.

[90] M. W. Newman, J. Lin, J. I. Hong, and J. A. Landay, "DENIM: An informal web site design tool inspired by observations of practice," *Human-Computer Interact*, vol. 18, no. 3, pp. 259–324, 2003.

[91] W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics.* New York: McGraw-Hill, 1979.

[92] D. Norman and J. Nielsen, "Gestural interfaces: A step backwards in usability," *Interactions*, vol. 17, pp. 46–49, 2010.

[93] D. A. Norman, *Emotional Design: Why We Love (or Hate) Everyday Things.* Basic Books, 2003.

[94] D. R. J. Olsen, "Evaluating user interface systems research," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 251–258, Newport, Rhode Island, USA, 2007.

[95] E. R. Pedersen, K. McCall, T. P. Moran, and F. G. Halasz, "Tivoli: An electronic whiteboard for informal workgroup meetings," in *Proceedings of INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, pp. 391–398, Amsterdam, The Netherlands, 1993.

[96] K. Perlin, "Quikwriting: Continuous stylus-based text entry," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST),* Technical Note, pp. 215–216, 1998.

[97] A. Pirhonen, S. Brewster, and C. Holguin, "Gestural and audio metaphors as a means of control for mobile devices," in *Proceedings of the Conference on Human Factors in Computing Systems (CHI): Changing Our World, Changing Ourselves*, 2002.

[98] R. W. Proctor and A. Dutta, *Skill Acquisition and Human Performance.* Thousand Oaks, CA: Sage, 1995.

[99] G. Ramos, M. Boulos, and R. Balakrishnan, "Pressure widgets," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 487–494, Vienna, Austria, 2004.

[100] D. Rubine, "Specifying gestures by example," in *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics and Interactive Techniques*, pp. 329–337, 1991.

[101] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd ed., 2002.

[102] R. A. Schmidt and R. Bjork, "New conceptionalizations of practice: Common principles in three paradigms suggest concepts for training," *Psychological Science*, vol. 3, no. 4, pp. 207–217, 1992.

[103] R. A. Schmidt and T. D. Lee, *Motor Control & Learning: A Behavioral Emphasis*. Human Kinetics, 5th ed., 2011.

[104] W. Schneider and R. M. Shiffrin, "Controlled and automatic human information processing: I. detection, search, and attention," *Psychological Review*, vol. 84, no. 1, pp. 1–66, 1977.

[105] W. Schneider and R. M. Shiffrin, "Controlled and automatic human information processing: II. perceptual learning, automatic attending and a general theory," *Psychological Review*, vol. 84, no. 2, pp. 127–190, 1977.

[106] B. Signer, M. C. Norrie, and U. Kurmann, "iGesture: A Java framework for the development and deployment of stroke-based online gesture recognition algorithms," Technical Report ETH Zurich, TR561, 2007.

[107] M. Smyth and G. Silvers, "Functions of vision in the control of handwriting," *Acta Psychologica*, vol. 66, pp. 47–64, 1987.

[108] I. E. Sutherland, "Sketchpad: A man-machine graphical communication system," MIT Lincoln Laboratory, Technical Report No. 296, 1963.

[109] C. C. Tappert, "Cursive script recognition by elastic matching," *IBM Journal of Research & Development*, vol. 26, no. 6, pp. 756–771, 1982.

[110] C. C. Tappert, C. Y. Suen, and T. Wakahara, "The state of the art in online handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 8, 1990.

[111] H. Tu, X. Ren, and S. Zhai, "A comparative evaluation of finger and pen stroke gestures," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 1287–1296, 2012.

[112] E. Tulving, "Introduction to memory," in *The New Cognitive Neurosciences*, (M. S. Gazzaniga, ed.), pp. 727–732, Cambridge, MA: MIT Press, 2nd ed., 2000.

[113] R. van Doorn and P. Keuss, "The role of vision in the temporal and spatial control of handwriting," *Acta Psychologica*, vol. 81, pp. 269–286, 1992.

[114] R.-D. Vatavu, "The effect of sampling rate on the performance of template-based gesture recognizers," in *Proceedings of the International Conference on Multimodal Interfaces*, pp. 271–278, Alicante, Spain, 2011.

[115] D. Venolia and F. Neiberg, "T-cube: A fast, self-disclosing pen-based alphabet," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 265–270, 1994.

[116] P. Viviani and T. Flash, "Minimum-jerk, two-third power law, and isochrony: Converging approches to movement planning," *Journal of Experimental Psychology: Human Perception and Performance*, vol. 21, no. 1, pp. 32–53, 1995.

[117] P. Viviani and C. Terzuolo, "Trajectory determines movement dynamics," *Neuroscience*, vol. 7, no. 2, pp. 431–437, 1982.

[118] F. Wang, X. Cao, X. Ren, and P. Irani, "Detecting and leveraging finger orientation for interaction with direct-touch surfaces," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, Victoria, BC, Canada, 2009.

[119] D. B. Willingham, "A neuropsychological theory of motor skill learning," *Psychological Review*, vol. 105, pp. 558–584, 1998.

[120] J. O. Wobbrock, M. R. Morris, and A. D. Wilson, "User-defined gestures for surface computing," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, 2009.

[121] J. O. Wobbrock and B. A. Myers, "Gestural text entry on multiple devices," in *Proceedings of the ACM SIGACCESS Conference on Computers and Accessibility*, pp. 184–185, Baltimore, MD, USA, 2005.

[122] J. O. Wobbrock and B. A. Myers, "Trackball text entry for people with motor impairments," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 479–488, 2006.

[123] J. O. Wobbrock, B. A. Myers, and D. H. Chau, "In-stroke word completion," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 333–336, Montreux, Switzerland, 2006.

[124] J. O. Wobbrock, B. A. Myers, and J. A. Kembel, "EdgeWrite: A stylus-based text entry method designed for high accuracy and stability of motion," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 61–70, 2003.

[125] J. O. Wobbrock, A. D. Wilson, and Y. Li, "Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 159–168, Newport, Rhode Island, USA, 2007.

[126] C. G. Wolf, "Can people use gesture commands?," *ACM SIGCHI Bulletin*, vol. 18, no. 2, pp. 73–74, 1986.

[127] C. G. Wolf and P. Morrel-Samuels, "The use of hand-drawn gestures for text editing," *International Journal of Man-Machine Studies*, vol. 27, no. 1, pp. 91–102, 1987.

[128] C. G. Wolf and J. R. Rhyne, "Gesturing with shared drawing tools," in *Proceedings of INTERACT '93 and CHI '93 Conference Companion on Human Factors in Computing Systems*, Amsterdam, The Netherlands, 1993.

[129] J. Yin, X. Ren, and S. Zhai, "Pen pressure control in trajectory-based interaction," *Behaviour & Information Technology*, vol. 29, no. 2, pp. 137–148, 2010.

[130] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes, "SKETCH: An interface for sketching 3D scenes," in *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics and Interactive Techniques*, 1996.

[131] S. Zhai, "Evaluation is the worst form of HCI research except all those other forms that have been tried," *CHI Place Essay*, Retrieved 2/5, 2010 from http://www.almaden.ibm.com/u/zhai/papers/EvaluationDemocracy.htm, 2003.

[132] S. Zhai, J. Accot, and R. Woltjer, "Human action laws in electronic virtual worlds — an empirical study pf path steering performance in VR," *Presence*, vol. 13, no. 2, pp. 113–127, 2004.

[133] S. Zhai, M. Hunter, and B. A. Smith, "Performance optimization of virtual keyboards," *Human-Computer Interaction*, vol. 17, no. 2, 3, pp. 89–129, 2002.

[134] S. Zhai and P.-O. Kristensson, "Shorthand writing on stylus keyboard," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 97–104, Fort Lauderdale, Florida, USA, 2003.

[135] S. Zhai and P. O. Kristensson, "Introduction to shape writing," IBM Research Report RJ10393, also as Chapter 7 of I. S. MacKenzie and K. Tanaka-Ishii (eds.), Text Entry Systems: Mobility, Accessibility, Universality, Morgan Kaufmann Publishers, pp. 139–158, 2006.

[136] S. Zhai and P. O. Kristensson, "The word-gesture keyboard: Reimagining keyboard interaction," *Communications of the ACM*, vol. 55, no. 9, pp. 91–101, 2012.

[137] S. Zhai, P. O. Kristensson, P. Gong, M. Greiner, S. A. Peng, and L. M. Liu et al., "Shapewriter on the iPhone: From the laboratory to the real world," in *Extended Abstracts of ACM Conference on Human Factors in Computing Systems (CHI) (Design Practice)*, pp. 2667–2670, Boston, MA, USA, 2009.

[138] S. Zhai, A. Sue, and J. Accot, "Movement model, hits distribution and learning in virtual keyboarding," in *Proceedings of the ACM conference on Human Factors in Computing Systems (CHI)*, pp. 17–24, Minneapolis, Minnesota, USA, 2002.

[139] S. Zhao and R. Balakrishnan, "Simple vs. compound mark hierarchical marking menus," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, Santa Fe, NM, USA, 2004.

[140] S. Zhao, P. Dragicevic, M. Chignell, R. Balakrishnan, and P. Baudisch, "Earpod: Eyes-free menu selection using touch input and reactive audio feedback," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 1395–1404, San Jose, California, USA, 2007.