

Enabling Expressive Keyboard Interaction with Finger, Hand, and Hand Posture Identification

by

Jingjie Zheng

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2017

© Jingjie Zheng 2017

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis includes first-authored peer-reviewed material that has appeared in conference proceedings published by the Association for Computing Machinery (ACM). The conference paper from which I have adapted content is the following:

- Jingjie Zheng and Daniel Vogel. Finger-Aware Shortcuts. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 4274-4285, New York, NY, USA, 2016. ACM.

DOI=10.1145/2858036.2858355

<http://dx.doi.org/10.1145/2858036.2858355>

Abstract

The input space of conventional physical keyboards is largely limited by the number of keys. To enable more actions than simply entering the symbol represented by a key, standard keyboards use combinations of modifier keys such as command, alternate, or shift to re-purpose the standard text entry behaviour.

To explore alternatives to conventional keyboard shortcuts and enable more expressive keyboard interaction, this thesis first presents Finger-Aware Shortcuts, which encode information from finger, hand, and hand posture identification as keyboard shortcuts. By detecting the hand and finger used to press a key, and an open or closed hand posture, a key press can have multiple command mappings. A formative study revealed the performance and preference patterns when using different fingers and postures to press a key. The results were used to develop a computer vision algorithm to identify fingers and hands on a keyboard captured by a built-in laptop camera and a reflector. This algorithm was built into a background service to enable system-wide Finger-Aware Shortcut keys in any application. A controlled experiment used the service to compare the performance of Finger-Aware Shortcuts with existing methods. The results showed that Finger-Aware Shortcuts are comparable with a common class of shortcuts using multiple modifier keys. Several application demonstrations illustrate different use cases and mappings for Finger-Aware Shortcuts.

To further explore how introducing finger awareness can help foster the learning and use of keyboard shortcuts, an interview study was conducted with expert computer users to identify the likely causes that hinder the adoption of keyboard shortcuts. Based on this, the concept of Finger-Aware Shortcuts is extended and two guided keyboard shortcut techniques are proposed: FingerArc and FingerChord. The two techniques provide dynamic visual guidance on the screen when users press and hold an alphabetical key semantically related to a set of commands. FingerArc differentiates these commands by examining the angle between the thumb and index finger; FingerChord differentiates these commands by allowing users to press different key areas using a second finger.

The thesis contributes comprehensive evaluations of Finger-Aware Shortcuts and proof-of-concept demonstrations of FingerArc and FingerChord. Together, they contribute a novel interaction space that expands the conventional keyboard input space with more expressivity.

Acknowledgements

Three and a half years later, I still vividly recall how overjoyed I felt when I learned Prof. Daniel Vogel would accept me as a Master's student. I could never imagine having walked down this road so far without his dedicated support and encouragement. I feel very, very lucky to have such a great supervisor. I am forever grateful for everything that he taught me.

My sincere thanks also go to Prof. Jesse Hoey and Prof. Meiyappan Nagappan at University of Waterloo for being my thesis readers and providing valuable feedback. It would not have been possible without their input that my thesis could be in such a great shape.

I would like to extend my gratitude to Shumin Zhai for being my mentor at Google Mountain View. Shumin expanded my view to a different research topic. He taught me to be rigorous, persistent, independent, and most importantly, create things that are useful. I also feel very lucky to have worked with Yang Li and Xiaojun Bi, both of whom set great examples of excellent researchers. I appreciate their continued help on my research.

I would like to express my appreciation to Stefan Gavrilovic and Raymond Wainman, who mentored me at Google Waterloo. I thank them for offering a wonderfully designed intern project that granted me the opportunity to explore and implement creative ideas. I appreciate their hands-on guidance and support that led to a genuinely rewarding internship experience.

I would like to convey my thankfulness to Chi Tse and Janseyit Tileubay, who mentored me at BlackBerry. Chi offered me my first internship position in Canada and taught me to think beyond individual roles. Janseyit built up my confidence as a software engineer with his considerate guidance and hearty support. He was always ready to answer my questions even when he was very, very busy. I would also like to thank Dmitri Pechkin, who taught me to write better code with his careful and rigorous code reviews.

I would like to thank my friends in the Human-Computer Interaction Lab at University of Waterloo, Hemant Surale, Qifan Li, Yunjia Sun, Alix Goguey, Qifeng Liu, Mingyu Liu, Yuexing Luo, William Saunders, Mathieu Nancel, Adam Fournery, Keiko Katsuragawa, Jeffery Avery, Anastasia Kuzminykh, Mike Schæckermann, Shaishav Siddhpuria, Lisa Elkin, Rina Wehbe, Jeremy Hartmann, Blaine Lewis, Bahareh Sarrafzadeh, Alex Williams, William Callaghan, Terence Dickson, Jay Henderson, and everyone else in the lab. I thank them for bringing up

constructive ideas, drinking beers together, and sharing life's ups and downs. I would like to thank other faculty members of the lab, Edith Law, Edward Lank, and Michael Terry, who have made the lab such a great place.

Finally, I wish to convey my gratitude to my girlfriend, Zhengkun Shang, for being my best friend, supporting me, making me happy, and inspiring me to become a better person. I wish to dedicate this thesis to my dearest parents and grandmother for their endless love and support.

Table of Contents

List of Figures	x
1 Introduction	1
1.1 Motivation	1
1.2 Finger-Aware Techniques	2
1.2.1 Finger-Aware Shortcuts	2
1.2.2 FingerArc and FingerChord	3
1.3 Contributions	5
1.4 Organisation	6
2 Related Work	7
2.1 Promoting Keyboard Shortcuts	7
2.2 Augmenting Keyboard Interaction	10
2.3 Identifying Fingers for Expressivity	11
2.4 Supporting Novice to Expert Transition	12
3 Finger-Aware Shortcuts	16
3.1 Concept	16
3.1.1 Keyboard Shortcuts	16
3.1.2 Finger-Aware Shortcuts	17
3.2 Experiment 1: Formative Study	18

3.2.1	Participants	18
3.2.2	Apparatus	19
3.2.3	Procedure	19
3.2.4	Design	22
3.2.5	Results	22
3.2.6	Discussion	25
3.3	Implementation	26
3.3.1	Keyboard Localisation	27
3.3.2	Hand Feature Extraction	27
3.3.3	Fingertip Localisation	28
3.3.4	Hand and Finger Identification	29
3.3.5	Hand and Finger Tracking	30
3.3.6	System-Wide Background Service	30
3.4	Experiment 2: Comparing with Shortcut Keys	31
3.4.1	Participants	31
3.4.2	Apparatus	31
3.4.3	Procedure	31
3.4.4	Design	34
3.4.5	Results	34
3.4.6	Discussion	36
3.5	Demonstrations	37
3.5.1	Command Mappings	37
3.5.2	Extensions	39
3.6	Summary	41

4 FingerArc and FingerChord	42
4.1 Interview with Expert Computer Users	43
4.1.1 Participants	43
4.1.2 Procedure	43
4.1.3 Results	44
4.1.4 Discussion	51
4.2 Concept	53
4.2.1 Design Goals	53
4.2.2 FingerArc and FingerChord	55
4.3 Proof-of-Concept Implementation	58
4.4 Summary	59
5 Conclusion	60
References	61

List of Figures

1.1	Finger-Aware Shortcuts basic concept with example.	3
1.2	FingerArc and FingerChord basic concepts with examples.	4
3.1	Finger-Aware Shortcuts formative study - experiment tasks.	20
3.2	Finger-Aware Shortcuts formative study - time and preference results for open hand postures.	24
3.3	Finger-Aware Shortcuts formative study - time and preference results for closed hand postures.	24
3.4	Finger-Aware Shortcuts implementation apparatus.	27
3.5	Finger-Aware Shortcuts algorithm.	28
3.6	Finger-Aware Shortcuts time performance compared to conventional keyboard shortcuts.	35
3.7	Finger-Aware Shortcuts demo – complementary command.	38
3.8	Finger-Aware Shortcuts demo – up/down control.	39
3.9	Finger-Aware Shortcuts demo – alternate modifier.	40
3.10	Finger-Aware Shortcuts demo – simultaneous parameter control.	40
3.11	Finger-Aware Shortcuts demo – two-handed key press.	41
4.1	FingerArc and FingerChord interaction model compared to traditional keyboard shortcuts.	55
4.2	FingerArc concept with example.	56
4.3	FingerChord concept with example.	57
4.4	FingerArc and FingerChord implementation.	59

Chapter 1

Introduction

1.1 Motivation

Physical keyboards were originally designed for text entry, but pressing keys can also issue commands with keyboard shortcuts (also called “hotkeys” [30]). Shortcut keys re-purpose standard text entry keys and may be differentiated from dedicated function keys like `F1` or `HOME`. In graphics software like Adobe Photoshop or Illustrator, switching between different tools can be achieved by simply pressing a single key. For example, `V` activates the selection tool, `P` enables the pen tool, and `I` accesses the eyedropper tool. In text-heavy applications like Google Docs or Microsoft Word, all shortcut keys need to include one or more special modifier keys like command `⌘`, control `ctrl`, or alternate `⌥`.

In desktop computing, where the Windows-Icons-Menus-Pointing (WIMP) interaction paradigm dominates, users start using software by exploring commands in their drop-down menus and toolbars. Keyboard shortcuts provide users with a faster alternative to graphical input – by learning the mappings between commands and keys and mastering their corresponding motor skills, experienced users can quickly express a large number of frequent commands [41, 49, 54, 68, 87] to improve the fluidity of interaction [27, 54, 71]. Good design of shortcut keys also helps reduce unnecessary physical movement [71], take advantage of bimanual interaction [60, 68], and allow users to concentrate on their objects of interest [5, 71]. Expert users can leverage keyboard shortcuts as a powerful tool to improve their efficiency, whereas novice

users can still count on menus and toolbars to easily learn new software functionality [64, 87].




Although commonly used by experts for applications like video editing [40] and programmers (like Donald Knuth [45]), studies show keyboard shortcuts are underused by most computer users [54, 71, 87]. Researchers attribute this to a gulf between graphical input and pressing keys [48, 62], poor visibility and mnemonics [30, 22], the uncomfortable and error-prone act of pressing multiple keys simultaneously [60], and the lack of motivation from users to spend time learning more efficient strategies [7, 8, 12, 65, 87]. Several techniques have been proposed to foster shortcut key usage with interventions and visualisations for training and encouragement [46, 30, 56, 87], as well as augmenting the keyboard or mouse to make shortcut keys more available [60, 73] and expressive [3].

The focus of this thesis is also on the availability and expressivity of keyboard shortcuts. We enable this by encoding information from finger, hand, and hand posture identification as part of keyboard input to make shortcut keys. We refer to such ability as *finger awareness* and techniques with such ability as *finger-aware techniques*. We present and discuss three finger-aware techniques: Finger-Aware Shortcuts, FingerArc, and FingerChord. The next section will briefly introduce these techniques with examples.

1.2 Finger-Aware Techniques

1.2.1 Finger-Aware Shortcuts

Finger-Aware Shortcuts detect which hand and finger are used to press a key, and whether the hand posture is open or closed, so that pressing the same key can have multiple command mappings. This enables a larger input space for traditional keyboards with increased expressivity by pressing keys in different ways to access more shortcuts, and increased availability by differentiating between normal keyboard input and shortcut input.

For example, pressing  with the left index finger and open hand simply enters the letter “G” as would be expected when touch typing (Figure 1.1a), but when the hand is closed, pressing  with the index finger could mean *Goto Line* (Figure 1.1b), and when pressed with the right hand, the same  could

mean *Goto Page*, *Goto Class*, *Grab*, or *Gap* depending on which finger and what posture is used (Figure 1.1c).



Figure 1.1: Finger-Aware Shortcuts trigger different commands by detecting the finger, hand, and posture used to press the key: (a) for example, pressing `G` with the left index finger and open hand simply enters the letter “G”; (b) pressing `G` with the left index and closed hand could trigger a command like “Goto Line”; and (c) pressing `G` with different right-hand fingers and postures could trigger other commands like “Goto Page”, “Goto Class”, etc.

Based on a formative study, we recommend fingers and hand postures to use. A controlled experiment shows that although there is a performance cost compared to simply pressing different keys individually or with a single modifier key, Finger-Aware Shortcuts are comparable to a common and practically-necessary class of shortcuts using multiple modifier keys with the same key.

We accomplish this by monitoring both typing hands using computer vision and the built-in laptop camera augmented with a small reflector. Previous work has tracked hands above a keyboard for mouse-like input or gestures [63, 96, 95, 70, 88], but to our knowledge, identifying which finger has pushed a key for the purpose of issuing shortcut commands has not been explored. Our detection algorithm is packaged into a background service to enable system-wide Finger-Aware Shortcut keys in any application. Based on the guidelines from the formative study, and using our background service, we demonstrate multiple applications illustrating different use cases and mappings for Finger-Aware Shortcuts.

1.2.2 FingerArc and FingerChord

FingerArc and FingerChord both activate keyboard shortcuts by detecting whether a special hand posture is formed while pressing a key. Similar to Finger-Aware Shortcuts, each key can be mapped to multiple commands to enable more expressive keyboard interaction. FingerArc differentiates the activation of these commands by examining the angle between the user’s thumb and index finger.

FingerChord differentiates these commands by letting the user press different key areas.

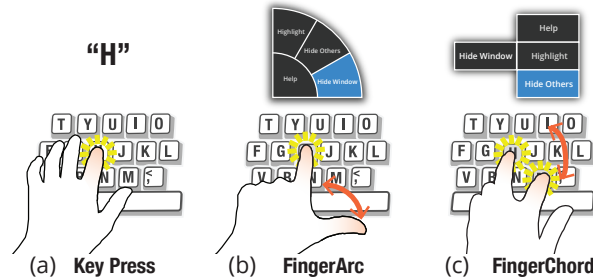


Figure 1.2: FingerArc and FingerChord both activate keyboard shortcuts by detecting whether a special posture is presented while pressing a key. For example: (a) pressing **H** with a natural hand posture simply enters the letter “H”; (b) pressing and holding the same key with the index finger and tucking in the little, ring, and middle fingers triggers the FingerArc shortcut interface; (c) pressing and holding with the middle finger and tucking in the little and ring fingers triggers FingerChord shortcut interface. The user can then rely on the visual guidance of the shortcut interfaces to choose from different commands semantically related to the key they have just pressed: (b) FingerArc differentiates these commands by examining the angle between the thumb and the index finger; (c) FingerChord differentiates the activation of these commands by letting the user press different key areas using a second finger. For both techniques, experienced users can directly issue the same command with the same posture without waiting for the shortcut interfaces to display.

Both techniques allow users with different levels of proficiency to interact with them easily. Similar to standard shortcut keys, a novice user may still rely on drop-down menus or toolbars to find out the shortcut mapping to a command. However, an intermediate user can press and hold the action key for a predesignated delay time to pop up the shortcut interface, then adjust the posture or press a second key to choose from different commands with the aid of the dynamic guidance shown in the shortcut interface. An experienced user can directly use the same action to activate the commands without waiting for the shortcut interface to display.

For example, pressing **H** with a natural hand posture simply enters the letter “H” (Figure 1.2a); pressing and holding the same key with the index finger and tucking in the little, ring, and middle fingers triggers the FingerArc shortcut interface (Figure 1.2b); pressing and holding with the middle finger and tucking in the little and ring fingers triggers FingerChord shortcut interface (Figure 1.2c). The user can then rely on the visual guidance of the shortcut interfaces to choose from a set of different commands semantically related to the key they

have just pressed: FingerArc differentiates these commands by examining the angle between the thumb and the index finger (Figure 1.2b); FingerChord differentiates these commands by letting the user press different key areas using a second finger (Figure 1.2c). For both techniques, experienced users can directly issue the same command with the same posture without waiting for the shortcut interfaces to display.

FingerArc and FingerChord enable more expressive keyboard shortcut activation by detecting more enriched hand postures. In addition, they also reduce the gap between graphical input and keyboard shortcuts. The intermediate step of visual guidance allows users to rehearse the expert behaviour by approximating the same movement with dynamic feedback. The alignment of the novice and expert movements is considered to be critical in supporting users to smoothly transition from novice to expert behaviour [50]. Some researchers refer to this as the *rehearsal hypothesis* [33]. Many techniques have been proposed following the same principle to exploit its benefits [1, 4, 34, 52, 56, 90, 101], but to our knowledge, designing guided shortcut interfaces responding to hand posture identification to enable the rehearsal of keyboard shortcuts has never been explored. After reporting an interview of expert computer users on keyboard shortcut usage, we demonstrate FingerArc and FingerChord as a proof of concept, enabled with a similar hardware setup to Finger-Aware Shortcuts.

1.3 Contributions

The thesis makes the following contributions:

- A novel concept of finger-aware keyboard shortcuts that encodes the information from finger, hand, and hand posture identification as part of keyboard input.
- A formative study that empirically assessed the interaction space enabled by finger, hand, and hand posture identification.
- A guideline that advises which fingers and hand postures are recommended and to be avoided for key press tasks in practice.
- A minimal, system-wide working system of Finger-Aware Shortcuts, with the aid of a green keyboard cover, a built-in camera, and a reflector.

- A comparison of the performance of Finger-Aware Shortcuts with three conventional shortcut key mapping strategies.
- An interview study with expert computer users that identified the likely causes as to why keyboard shortcuts are underused by experienced users.
- An extension to the basic concept of Finger-Aware Shortcuts that provides dynamic visual guidance and allows for more expressive hand postures.

1.4 Organisation

The remainder of this thesis is organised as follows:

- Chapter 2 describes the previous work on keyboard shortcuts, methods to augment physical keyboards, finger identification for expressive interaction, and techniques that support novice to expert behaviour transition.
- Chapter 3 describes the concept, implementation, and two empirical evaluations of Finger-Aware Shortcuts.
- Chapter 4 describes the concept and proof-of-concept implementation of FingerArc and FingerChord.
- Chapter 5 summarises the contributions of previous chapters and concludes the thesis.

Chapter 2

Related Work

The finger-aware techniques proposed in this thesis, Finger-Aware Shortcuts, FingerArc, and FingerChord, all relate to the research areas of understanding, improving, and promoting keyboard shortcuts, augmenting conventional keyboard input, and identifying fingers to improve the expressivity of human-computer interaction. In addition, FingerArc and FingerChord also relate to the research that aims at supporting the smooth transition from novice to expert performance. In this chapter, we detail how our techniques are related to the previous research in these domains.

2.1 Promoting Keyboard Shortcuts

In desktop computing, interaction with computers often takes the form of either providing graphical input using a pointing device or activating shortcuts with a keyboard. The two different interaction methods satisfy the needs of both novice and expert users, so the literature (e.g. [14]) often refers to the former as the *novice mode* and the latter as the *expert mode*.

Graphical user interfaces (GUIs) draw metaphors from the real world and use elements like windows, icons, menus, and a pointer (WIMP) to minimise the effort from users in learning computers. Novice users may rely on drop-down menus and icon toolbars to visually inspect possible actions in an application and attain a certain level of proficiency in the GUI through repeated practice. The power of this interaction paradigm, as Norman [67] theorised, is that it

allows users to distribute knowledge and cognition to the world (graphic elements) and minimise the need for memorising actions in the head (keyboard commands). Evidence also suggests that menu commands can be more easily relearned than keyboard shortcuts after disuse [44].

Keyboard shortcuts, on the other hand, provide a faster alternative to graphical input. With the GUI, repeated access to the same command causes users to make frequent round trips between menus, toolbars, and objects of concern [5], but with keyboard shortcuts, they can simply press one or more keys on a keyboard. Anecdotal evidence suggests some expert users regularly use keyboard shortcuts: Jacob et al. claimed video editors do [40], and Knuth said he uses so many Emacs shortcuts that it is “a little bit like playing the organ” [45]. This makes sense considering Lane et al. [54], Karat et al. [42], and others [68, 75] have shown that triggering commands with keys can be faster than pointing at graphical widgets using a mouse.

Yet, keyboard shortcuts do require a substantial amount of conscious training before the benefits start to show [75]. Much research [30, 37, 54, 87] has found that keyboard shortcuts are not frequently used by users, even the experienced ones. To some extent, this can be ascribed to the poor visibility [30, 87], the cumbersome and error-prone movement of simultaneously pressing multiple keys [49, 60, 87], and the lack of motivation from users to spend time learning more efficient strategies [7, 8, 12, 65, 87]. More importantly, the cognitive process and physical movement of activating keyboard shortcuts are radically different from choosing items from a drop-down menu [37, 48, 49, 52, 51]. This difference means that an efficient and smooth transition to using keyboard shortcuts with the current GUI paradigm is nearly impossible [50]. On top of this, the often poor mnemonics of shortcut key mappings make it even harder for users to learn [30, 37, 49, 71].

Researchers have motivated people to learn and use keyboard shortcuts using different strategies:

- *Reinforcement-based strategy* loads more information to users when they preview or activate menu or toolbar commands to reinforce their perception to keyboard shortcuts. For example, Grossman et al. [30] tapped the auditory channel to provide feedback when clicking on a menu item in order to remind users of its corresponding keyboard shortcut. Giannidakis et al. [23] blended the shortcut key characters as part of toolbar icons to increase the visibility of their corresponding keyboard shortcuts.

- *Cost-based strategy* adds a cost to activating commands in the novice mode to encourage users to learn and rehearse keyboard shortcuts. The cost typically takes the form of a delay in performing the action after clicking a menu item [30] or completely disabling the menu items to force users to use keyboard shortcuts [46, 30]. Grossman et al. [30] showed that the latter is more effective, but Malacria et al. [56] and Gutwin et al. [33] argued that this strategy may harm the usability of the graphics-based interaction for novice users.
- *Rehearsal-based strategy* lets users voluntarily switch to using keyboard shortcuts, but when they do, it helps users smoothly transition to experts by making the access and physical rehearsal of keyboard shortcuts easier. For example, Tak et al. [87] displayed a crib sheet of all possible commands when a modifier key is being held. Similarly, Malacria et al. [56] showed shortcut information next to icons or on an opened menu upon a hold on a modifier key.

There are also other strategies, although less common. For example, Malacria et al. [57] displayed an efficiency score to encourage shortcut usage; Bailly et al. [3] physically raised actual keys on a keyboard when users press a modifier key to increase haptic and visual feedback; and Pietrzak et al. [73] duplicated modifier keys to the mouse to help reduce the movement of hands.

Along this line, FingerArc and FingerChord apply both the cost- and rehearsal-based strategies: the predesignated delay while pressing an action key adds a cost to the interface pop-up to nudge users towards actively retrieving the shortcut posture; the alignment of physical actions with and without the visual display allows users to rehearse the expert behaviour with guidance and feedback. We explain this in more detail when reviewing theories and techniques that support the smooth transition from novice to expert performance.

On the other hand, Finger-Aware Shortcuts, FingerArc, and FingerChord all increase shortcut expressivity and availability. Expressivity [3] means a key can be pressed in different ways to increase input capability. We achieve expressivity by letting the user press a key with different fingers, hands, and hand postures to access more shortcuts. Availability [58, 71] means shortcut commands are accessed with a single key press rather than a key combination. We increase availability by differentiating between normal keyboard input and shortcut input by detecting how a key is pressed.

2.2 Augmenting Keyboard Interaction

Keyboard input can be expanded by pressing keys in unconventional ways, such as Zhang and Li’s GestKeyboard [103] that recognises motion gestures as multiple keys are stroked like a touchscreen, but a more common approach is augmenting the simple key switch with additional sensors. PreSense Keypad [74] senses finger contact before pressing and Dietz et al. [17] developed a pressure sensitive keyboard for continuous input while pressing a key. Touch-Display Keyboard [9] augments each key with a touch-sensor and a display to increase the input space when the keyboard is used as a single surface and increase shortcut visibility by displaying commands on each key.

Another approach is to augment keyboard input by tracking hand and finger movement above the keyboard. SpaceTop [55] placed a transparent screen above the keyboard and a downward facing depth camera to detect finger touch and hand gestures. FlowMouse [96] and TAFFI [95] used a downward facing camera to track hands moving above a conventional keyboard for mouse and touchscreen style input performed mid-air. FingerMouse [63] and AirMouse [70] did the same, but combine mid-air hand movements with key presses (or touch pad taps) to simulate mouse clicks. Air Typing Keyboard [99] enabled mid-air typing with a Leap Motion sensor facing up on the desktop. Taylor et al.’s [88] keyboard with a matrix of infrared proximity sensors embedded between the keys detects a large set of motion gestures that can be easily interlaced with key presses. However, all these projects essentially treat hand gestures and key presses as two independent input techniques.

It is worth noting the body of research combining vision-based mid-air or near-surface hand gesture input with interactive surfaces, but the use case is very different from desktop computing and soft keyboards are not a focus for interaction. For example, Visual Touchpad [58] demonstrated the robustness of their vision-based system by typing on a virtual keyboard, RetroDepth [43] included an example of a passive retro-reflective keyboard, and a keyboard “SLAP widgets” [94] is a motivating example for the idea of tangible, translucent widgets for multi-touch tabletops.

2.3 Identifying Fingers for Expressivity

A touch on a surface – whether it is a multi-touch screen, a keyboard key, or a piano key – can be augmented with additional finger information to make interaction more expressive. For example, Wang and colleagues [91, 92] detected finger orientation on a touchscreen; Harrison and Hudson [35] considered also the shear force of fingers tangential to the touchscreen surface; Surale et al. [86] enabled users to signal an alternative input mode by bringing the thumb and index finger together; Harrison et al. [36] differentiated a finger tap and a knuckle knock on a mobile phone; Huang et al. [39] expanded input vocabulary with different areas of finger pads on a smartwatch; Boring et al. [10] detected the contact size of the thumb to enable different input modes; and McPherson et al. [61] used the position information on a piano key to express continuous drift from the original sound pitch. Like how a violin string can be manipulated with a bow to produce different sounds, these sensing techniques re-purpose the functionality of touchscreens, which were originally designed for registering input coordinates, to enable more expressivity.

Our work is based on the idea of encoding information from finger, hand, and hand posture recognition as part of keyboard input. Although not exactly the same, it is largely relevant to the concept of *finger identification* (see Goguey et al. [27] for a comprehensive review). Researchers have explored finger identification on different form factors. For example, Gupta et al. [32] and Gil et al. [24] both extended smartwatch touch input with finger identification; Gorodnichy and Yogeswaran [28] and Oka and Hashimoto [69] both enabled identifying fingers on piano keys; Colley and Hakkila [16] identified fingers to trigger different commands on a mobile phone; Gupta et al. [31] did the same, but applied it to enable multitasking. Goguey et al. [26] systematically evaluated the performance and preference of using different fingers for pointing on a tablet; Masson et al. [59] achieved finger identification on touch surfaces and physical keyboards by attaching vibration sensors to each finger; and Goguey et al. [25, 27] proposed FingerCuts to enable richer vocabulary with finger information from both hands on tabletops, tablets, and mobile phones.

More relevant to our work are Sugiura and Koseki [85] and Wang and Canny [93]. They both introduced the general idea of pressing a button with different fingers to activate different commands. However, they were early proof-of-concept studies using devices that can hardly be called “buttons”, much less a full keyboard: Sugiura and Koseki used a commercial fingerprint scanner

and Wang and Canny used a piece of glass. Wang and Canny evaluated time to switch fingers, we evaluate the more practical aspect of performance and preferences when using different fingers for shortcuts after typing and pointing tasks. Neither applied this to a physical keyboard, added the idea of hand postures, or explored the performance and preference design space for shortcut-like command activation in real applications.

2.4 Supporting Novice to Expert Transition

FingerArc and FingerChord extend the basic concept of finger awareness with more enriched hand postures, visual guidance, and feedback. Their designs are influenced by the large body of research aiming at supporting novices to effectively transition to experts.

Supporting users to efficiently transition from novice to expert performance is a recurrent theme in Human-Computer Interaction (see Cockburn et al. [14] for a review). While it is commonly agreed that user interfaces should provide mechanisms for expert users to attain higher performance [64, 72], much evidence has shown that people very rarely use these mechanisms. For example, Bhavnani and John [7, 8] found that computer-aided design (CAD) software users do not progress to using efficient strategies with long-term experience; Rosson [76] reported that skills of text editor users stabilised at a non-optimal level; and Nilsen et al. [66] found the same for spreadsheet users.

Carroll and Rosson [12] referred to this common observation that users' knowledge and performance of software tend to converge to a non-optimal level as the "*paradox of the active user*". They suggested that there are two cognitive biases that impede the learning of high-performance mechanisms: *production bias* refers to the observation that users tend to persist with the methods they have already learned when approaching the same tasks; *assimilation bias* refers to the observation that users tend to apply the existing knowledge when faced with new tasks.

Past research has suggested different explanations to the paradox of the active user. Tak et al. [87] interpreted this with "*satisficing*", a behaviour in economics that people tend to accept a "good enough" level of outcome without trying to maximise it [84]. Gray et al. [29] also referred to this phenomenon as "*local optimality*". These interpretations seem to suggest that stabilising at a

suboptimal level of performance is inherent to our cognition. Nevertheless, Fu and Gray [20] argued that the novice methods are preferred because they are indeed more beneficial – they showed that novice methods are typically more generic, applicable to a variety of different contexts, and provide fast and incremental feedback; whereas expert methods are typically specialised, applicable to only one specific type of task. This makes sense considering how GUIs can be easily learned with the knowledge of only a few standard graphical widgets and how a mouse works, whereas every single keyboard shortcut or stroke gesture need to be intentionally learned [2]. As a result, the novice methods become increasingly rehearsed and reinforced through practice. The excessive guidance provided by novice user interfaces might also become relied upon by users so as to hinder the learning of expert methods [1, 81].

Kurtenbach and colleagues [50] realised that supporting users to transition to expert performance is especially hard when the novice and expert methods are drastically different behaviours, or as Scarr et al. [80] characterised, in two different modalities. Scarr et al. suggested that in order to learn more effective strategies in a different modality, users need to first go through a performance dip that yields lower throughput than reusing the learned novice methods. Kurtenbach and colleagues’ research on Marking Menus [48, 50, 51, 52] made substantial contributions to overcoming this performance dip. A Marking Menu provides two different modes for novice and expert users. A novice user presses and holds at an activation point for one-third seconds to reveal a hierarchical pie menu, which visually guides them to select a menu item. An expert user, without waiting for the menu display, directly approximates the stroke used for selecting the same item on the pie menu to issue the same command. Marking Menus are expected to afford smooth transition to expert behaviour because the novice execution of the user interface is the physical rehearsal of the expert behaviour – this is also called the “*rehearsal hypothesis*”.

Following the rehearsal hypothesis, a large body of research has been conducted to improve user interface learning. These interfaces are also referred to as “*rehearsal interfaces*” [33]. For example, ShapeWriter [47, 101, 102] enables users to connect a sequence of key taps into a stroke gesture on a soft keyboard to input complete words. OctoPocus [4] provides dynamic, contextual guidance showing matching pen-stroke gestures if users slow down their drawing movement. FastTap [33, 34] let novice users hold an activation button with one finger to display a grid of buttons and use a second finger to tap on a button to activate

commands; expert users can directly tap on the same locations with a chord posture.

In addition to the rehearsal hypothesis, evidence also shows the following factors may facilitate novice to expert performance transition:

- *Cost* – a cost added to the novice execution, such as a delay in revealing the visual interface, can encourage mental elaboration and active retrieval of expert behaviour [15, 18, 30]; too much cost, may impede learning [15].
- *Guidance* – an appropriate amount of guidance can help users specify and complete their actions [4, 50, 82]; too much guidance, however, might hinder learning [1, 82].
- *Spatial stability* – a user interface being spatially stable helps users leverage the power of spatial memory, reduce visual search time, and retrieve visited items faster [77, 78, 79, 34, 53, 83, 90, 89, 19, 38].

Of all the rehearsal interfaces, our work is most relevant to ExposeHK [56], which allows users to press a modifier key to display shortcut information overlaid on their associated commands. ExposeHK can be used for both toolbars and menus. With menus, when the modifier key is pressed, all the menu items will be displayed in a modified visual layout so that users can browse and access the commands using shortcut keys in one display. While flattening the command hierarchy can maximise expert performance (e.g. [77]), in practice, it can be hard to adapt all user interface commands into one single display. Menu hierarchies are regularly employed to convey complex command constraints and sometimes there might simply be too many commands in one application.

In contrast, FingerArc and FingerChord do not seek to enable shortcut access to all commands. They are complementary to existing interfaces – no modification to the menu layout is required. FingerArc and FingerChord shortcuts can be displayed next to menu items just like regular shortcuts. Users then start exploring a shortcut key when they frequently access the same menu item. A family of related commands can be bound to the same key to increase their discoverability.

As we discussed earlier, FingerArc and FingerChord are the combinations of rehearsal- and cost-based strategies to promote keyboard shortcuts. Once users decide to activate a FingerArc or FingerChord shortcut, they are presented with

an intermediate mode that provides dynamic visual guidance in a display after holding a key for a predesignated delay time. More experienced users can simply use the same hand posture to activate the same command. These designs make FingerArc and FingerChord a rehearsal interface, with a cost added to the novice execution, guidance provided to help users to specify their actions, and spatial stability from the nature of keyboard keys.

Chapter 3

Finger-Aware Shortcuts

In this chapter, we propose, evaluate, and demonstrate Finger-Aware Shortcuts. We first introduce their basic concept. Then, a formative study measures the performance and preference for using different fingers, hands, and hand postures to press a key. Based on the result of the formative study, we present a guideline that recommends which postures to use and which ones to avoid. We then detail how Finger-Aware Shortcuts can be implemented by adding only a few additional components to a laptop. Finally, we compare Finger-Aware Shortcuts with conventional shortcut invocation methods, and demonstrate how they can be mapped with real-world commands.

3.1 Concept

We provide details of standard shortcut keys and discuss how Finger-Aware Shortcuts can replace or augment them.

3.1.1 Keyboard Shortcuts

Our focus is on two primary types of shortcuts: single keys (e.g. **B** for *Brush*) and keys pressed while one or more modifier keys are held down (e.g. **⌘+C** for *Copy*). Other more complex variations exist such as keys pressed after a special command mode is entered (e.g. **ctrl+D**, then **K** for *Tasks*), but this is not our focus.

Single “unmodified” shortcuts are practical only when keys are not used for text entry, so this approach is not suited to word processors or text editors. However, they are common in graphical direct manipulation applications like photo editors. A common mapping strategy is to choose keys that correspond to the first letter of the command for a semantic connection. Even with a handful of commands, the first letter key will already be taken leading to more arbitrary mappings (e.g. **R** for *Blur*). The number of available shortcut commands is also limited by the number of keyboard keys.

A more generalised method is pressing a single key while holding one or more modifier keys. A single modifier can be used, like **⌘**+**P** for *Print*. The ideal mapping uses the first letter of the command, but collisions are unavoidable. Consider **⌘**+**V** for *Paste*. The semantic link could be how “V” resembles an insertion mark, but more likely it was chosen for non-semantic reasons. Since **⌘**+**C** is used for *Copy* and *Paste* is often used immediately after, **V** suggests a proximity mapping since the two shortcut keys are side-by-side. To expand the input space and enable more meaningful mappings, different modifiers may be used like **⌘**+**F** for *Find* and **ctrl**+**F** for *Format*. Or, multiple modifiers can be used together like **⌘**+**↑**+**F** for *Find in Project*, **ctrl**+**⌘**+**F** for *Full Screen Mode*, and **ctrl**+**↑**+**⌘**+**F** for *Distraction Free Mode*.

3.1.2 Finger-Aware Shortcuts

The central concept is to interpret key presses differently depending on which finger is used and whether the remaining fingers on the same hand are spread open or bent closed. In theory, this provides 20 different ways to interpret each key press (10 fingers × 2 hand postures). In actuality, the formative study in the next section shows 16 are actually reasonable with 8 identified as best. This increased expressivity can be exploited as Finger-Aware Shortcuts.

Text-entry applications can have single “unmodified” shortcut keys based on standard touch typing finger-to-key mappings [6]. When the usual finger presses a key while typing, the alphanumeric character is sent (e.g. pressing **G** with the left index finger sends “G”). But, when a key is pressed with a finger not normally used while typing, it sends a command (e.g. pressing **G** with the left middle finger invokes *Goto Line*). This strategy can be relaxed for less-proficient typists by differentiating by open and closed hand postures: typing

with an open hand enters text but pressing keys with one finger and a closed hand activates shortcut commands.

Different fingers can map related commands to the same key. For example, pressing **C** with the right index finger, open hand for *Copy*, pressing **C** with the same finger, closed hand for *Cut*, and pressing **C** with the right middle finger for *Paste*. This enables two first letter mappings (*Cut* and *Copy*) and provides a proximity mapping (*Paste*). New kinds of semantic relations can be created based on fingers and postures. For example, using the right thumb, closed hand to issue global commands like *New Document* and the left thumb, closed hand to issue contextual commands like *New Layer*.

A standard modifier key can be combined with finger-specific actions to augment current shortcuts. As an example, accessing a family of commands by holding **⌘** then pressing a key with different fingers, such as **F** with the left index for *Find*, the left middle for *Find and Replace*, or the left ring for *Find in Project*. Specific fingers can be mapped to common modifier keys used with **⌘**. For example, holding **⌘** then pressing a key with the left index finger could mean **⌘**+**↑**, the left middle **ctrl**+**⌘**, or the left ring **ctrl**+**↑**+**⌘**.

Finger-Aware Shortcuts could potentially benefit expert users using a large number of commands. However, all of these ideas should be based on performance and preference when using different fingers and hand postures for key presses, which is the goal of the following study.

3.2 Experiment 1: Formative Study

The purpose of this study was to identify a set of postures that maximise performance and minimise error and fatigue. We measured time, error, and subjective preference for different postures in different conditions. The results of this study provided data for training and testing our posture recognition algorithm and enabled us to empirically derive guidelines for selecting suitable postures.

3.2.1 Participants

We recruited 21 right-handed participants (9 female, mean age 25.4, $sd=4.9$). All reported extensive experience with desktop or laptop computers. Self-reported

average weekly computer use ranged from 28 to 84 hours ($M=50.7$, $SD=12.6$). A one-minute typing speed test was performed on participants at the beginning of the experiment. Mean typing speed was 47.9 words-per-minute ($SD=12.7$) with a mean error rate of 3.11% ($SD=3.51\%$).

Note initial analysis found Participant 17 had a very high error rate 11.4% for the third task (explained in Task and Stimuli) compared to the mean error rate 2.47% ($SD=2.61\%$) across all participants. This participant was removed, and Participant 21 was recruited to keep the design balanced.

3.2.2 Apparatus

The experiment was performed on an Apple MacBook Pro with a 15-inch display and a QWERTY keyboard. The software was written as a web application in Google Chrome hosted from a local server. The built-in camera recorded the keyboard area with the aid of an Osmo reflector (playosmo.com, see also Figure 3.4a). To simplify background subtraction, a green keyboard cover and green material covered the laptop. A 20-inch square softbox was placed over the laptop at a suitable distance to provide consistent lighting. Hand colours were sampled.

3.2.3 Procedure

Each trial in our experiment consisted of three tasks: a posture task, a reference task, and a repetition of the posture task (Figure 3.1a,b,c). Upon completion of each task, there was a ding sound with different pitches to indicate success.

In a posture task, the participants were required to use a certain hand posture to press a certain key. A hand posture was defined in terms of three independent variables: HAND (LEFTHAND, RIGHTHAND), FORM (OPEN, CLOSED), and FINGER (THUMB, INDEX, MIDDLE, RING, LITTLE). Each finger needed to press three KEYAREAS (Figure 3.1d):

- HOMEAREA refers to the keys a finger normally presses when typing – the home row key of a finger and the two keys above and under that key. For example, the left middle finger’s home row key is **D**, so its HOMEAREA includes **D** (home row key), **E** (above), and **C** (under).

- LEFTAREA refers to the keys at the extreme left: **Q**, **A**, **Z**.
- RIGHTAREA refers to the keys at the extreme right: **P**, **;**, **/**.

The thumb's HOMEAREA contains only one key, **SPACE**. Further, because the home keys for the left little finger is **Q**, **A**, **Z**, same as LEFTAREA, we chose **W**, **S**, **X** as its LEFTAREA instead. For the same reason, we used **O**, **L**, **.** as RIGHTAREA of the right little finger instead of **P**, **;**, **/**.

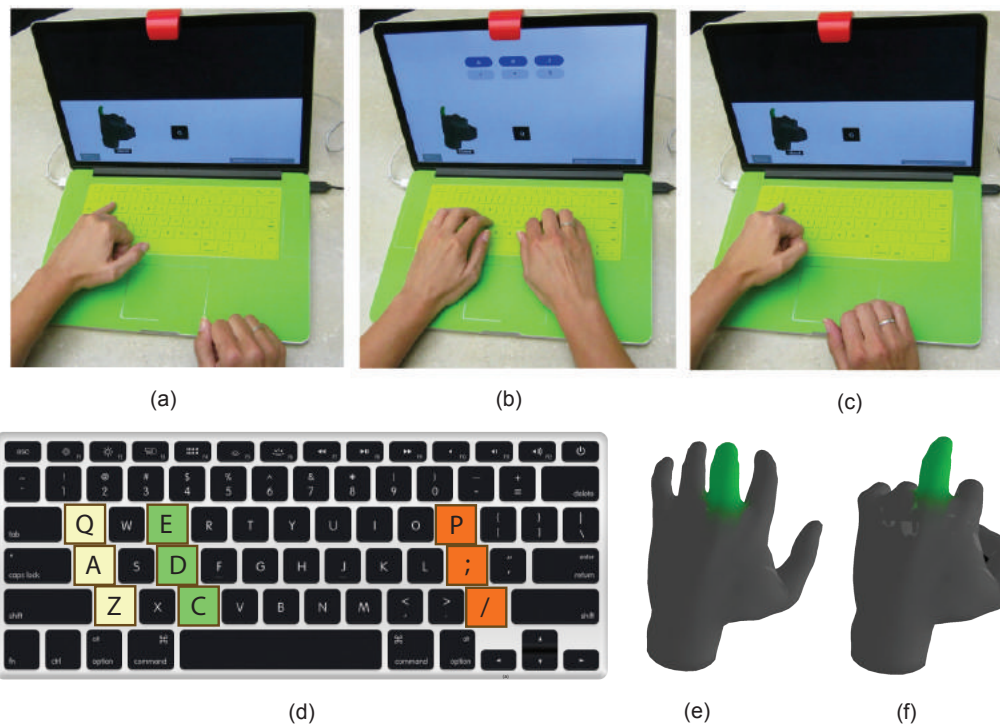


Figure 3.1: Experiment task: (a) prompt for finger and key with initial press; (b) typing reference task; (c) final key press. (d) Key area, example for left middle finger: home area (coloured green in this figure); left area (yellow); right area (red). Hand form examples: (e) left middle, hand open; (f) left middle, hand closed.

In a reference task, participants were required to either type three characters using the keyboard (TYPING) or click a button using the trackpad (POINTING). The purpose was to simulate real-world keyboard interaction. In TYPING, participants were prompted with three characters and they needed to type them sequentially (Figure 3.1b). The characters were randomly selected from the home

row: one from left (A, S, D, F), one from right (J, K, L, ;), and the other randomly chosen from one side. This would force the participants to put both hands onto the keyboard. In POINTING, the participants were asked to use the trackpad to click a button that appeared on the screen. The button could appear either on the left or right side of the display, always different from the last appearance. This forced the participants to place their dominant hands onto the trackpad.

Participants were instructed to memorise the hand posture and the key in the first (posture) task and perform the third (posture) task quickly and accurately. No requirement was imposed to the reference task.

Error and Time

In the experiment, a posture task was considered successful if the participants used the correct hand, form, and finger to press the correct key. When a wrong key was pressed, the participants were immediately informed and would not be able to continue until the correct key was pressed. Posture errors like using a wrong hand, form, or finger were ignored during the experiment and manually classified after it using images captured by the camera. A reference task was considered successful if the correct characters were typed in sequence or the button was clicked. An error occurred when the expected character was not typed or a click did not happen on the button. When an error occurred, a low-frequency basso sound would be played.

Three durations were recorded for each trial, corresponding to the three tasks. Duration 1 started when the start button was pressed in the beginning of a block or the last task was completed. It ended when the correct key was pressed. Duration 2 started immediately after the first task was completed and ended when all three characters were typed or the button was clicked. Duration 3 started immediately after the second task was completed and ended when the correct key was pressed.

We are interested in the time and error when users are cognitively prepared for the key to push and the posture to use after a reference task, so the third task was used for analysis. Note the first task allowed participants to rehearse the required posture to key, the second task simulated keyboard or touchpad use, and the third task provided the best estimation for the time and error measures of practised performance.

3.2.4 Design

The experiment design was within-subject, repeated measures, and full factorial. All trials for each type of FORM were presented together in counterbalanced order. For each FORM, participants performed 3 blocks of measured trials. Each block presented all trials for each combination of HAND, FINGER, KEYAREA, and REFTASK in random order. A short training containing 8 randomly selected trials for each FORM was performed prior to the formal experiment. Short rest breaks were enforced at the end of each block.

In summary: 3 BLOCKS \times 2 FORMS \times 2 HANDS \times 5 FINGERS \times 2 REFTASKS \times 3 KEYAREAS = 360 trials per participant.

3.2.5 Results

Repeated measures ANOVA and pairwise t-tests with Holm correction were used for all measures¹. Because the measures for preference exhibited non-normality, they were transformed using Aligned Rank Transform [97]. Trials were aggregated by participant and the factors being analysed. Time data were aggregated using the median.

Learning Effect

Overall, BLOCK had a main effect on time ($F_{1,35,25.71} = 30.91$, $p < .0001$, $\eta^2 = .619$), but not on error. Post hoc tests found block 1 significantly slower than blocks 2 and 3 (both $p < .0001$) and block 2 significantly slower than block 3 ($p < .005$), suggesting a learning effect across all blocks. In all subsequent analysis, we used only block 3 for the best estimation of practised performance.

Error Rate

No main effect was found for REFTASK, FORM, HAND, FINGER, or KEYAREA on error rate and the overall mean error rate was a very good 1.88% (SD=1.83%). The

¹When the assumption of sphericity was violated, we corrected the degrees of freedom using Greenhouse-Geisser (Greenhouse-Geisser's $\epsilon < 0.75$) or Huynh-Feldt (Greenhouse-Geisser's $\epsilon \geq 0.75$).

total error rate breakdown was 1.08% (SD=1.24%) for pressing a wrong key and 0.79% (SD=1.06%) for using a wrong posture when the correct key was pressed.

Time

The mean time across all the participants was 1060.85ms (SD=299.52). There was a main effect of REFTASK on time ($F_{1,19} = 59.769$, $p < .0001$, $\eta^2 = .759$). Time with POINTING as the reference task was significantly faster than TYPING by 346ms. There was a main effect of FINGER on time ($F_{4,76} = 10.853$, $p < .0001$, $\eta^2 = .364$) (Figure 3.2 and Figure 3.3). Post hoc tests showed THUMB was significantly faster than INDEX and LITTLE (both $p < .05$) and MIDDLE and RING (both $p < .001$). As would be expected, there was a main effect of KEYAREA on time ($F_{2,38} = 23.892$, $p < .0001$, $\eta^2 = .557$). Post hoc tests showed pressing keys in HOMEAREA was significantly faster than LEFTAREA by 100ms ($p < .0005$) and RIGHTAREA by 153ms ($p < .0001$), LEFTAREA was significantly faster than RIGHTAREA by 53ms ($p < .05$). No main effect was found for hand or form on time suggesting similar time performance for left and right hands and open and closed forms.

A further examination found an interaction between KEYAREA and HAND on time ($F_{1,39,16,40} = 5.527$, $p < .05$, $\eta^2 = .155$). Post hoc tests found LEFTHAND was significantly slower when pressing keys in RIGHTAREA compared to LEFTAREA by 145ms ($p < .05$). No pairwise difference was found for comparisons involving RIGHTHAND OR HOMEAREA. Regarding different fingers, there was an interaction between KEYAREA and FINGER on time ($F_{8,152} = 7.132$, $p < .0001$, $\eta^2 = .273$). Post hoc tests did not find any difference between pressing LEFTAREA and RIGHTAREA with any finger, but there were differences involving HOMEAREA for all fingers except INDEX. THUMB had the most pronounced difference where HOMEAREA was significantly faster than LEFTAREA by 273ms and RIGHTAREA by 340ms (both $p < .0001$). MIDDLE, RING, and LITTLE had more moderate differences with HOMEAREA significantly faster than RIGHTAREA and/or LEFTAREA by 130ms to 150ms (all $p < .05$).

There was an interaction between REFTASK and FINGER on time ($F_{2,78,52,73} = 5.877$, $p < .005$, $\eta^2 = .236$). Post hoc tests showed that for each finger, durations after the POINTING reference task were significantly faster than TYPING ($p < .0005$ for THUMB; $p < .0001$ for others). There was no interaction between REFTASK and HAND, HAND and FINGER, OR FORM and FINGER on time. An examination of the data also indicates a symmetric pattern for fingers across hands for these factors (Figure 3.2 and Figure 3.3).

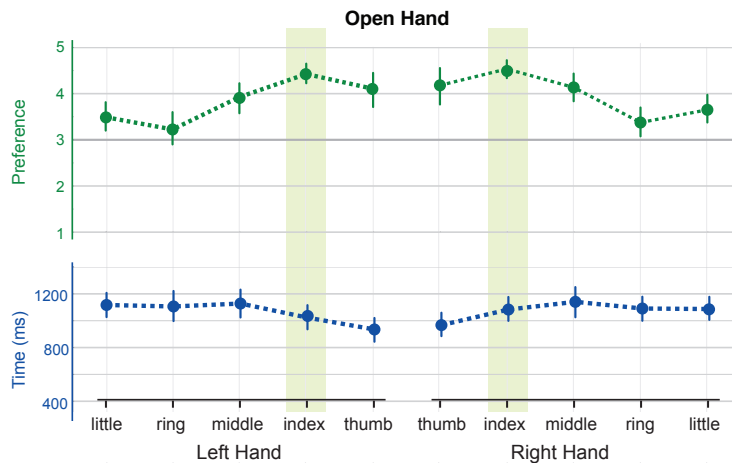


Figure 3.2: Time and preference by HAND and FINGER for open hand form. Green shading indicates Tier 1 fingers (best). Note FINGER is a categorical variable, dashed lines connecting fingers used for readability only. Error bars are 95% confidence intervals.

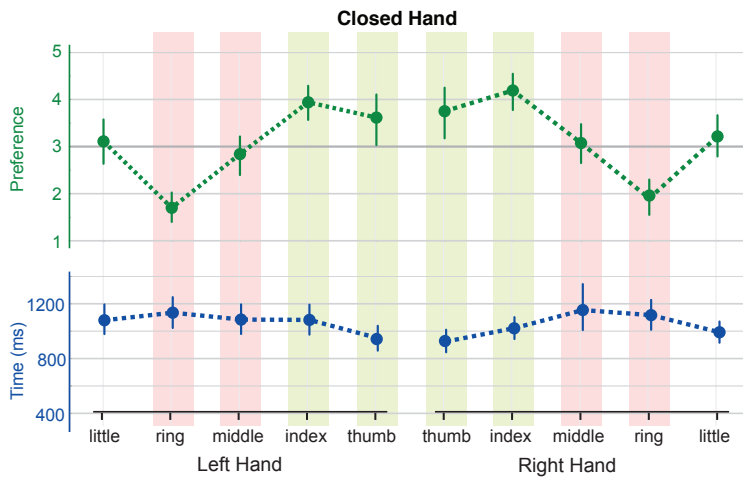


Figure 3.3: Time and preference by HAND and FINGER for closed hand form. Green shading indicates Tier 1 fingers (best), red shading indicates Tier 3 fingers (to avoid). Note FINGER is a categorical variable, dashed lines connecting fingers used for readability only. Error bars are 95% confidence intervals.

Preference

At the end of the experiment, participants were asked to consider accuracy, speed, and fatigue and provide a numerical preference score for each combi-

nation of HAND, FINGER, and FORM. Preference scores used a real numbered, continuous scale from 1 (least preferred) to 5 (most preferred). Decimal ratings such as 3.5 were permitted.

There was a main effect of FINGER on preference ($F_{2.39,45.47} = 21.962$, $p < .0001$, $\eta^2 = .536$). Post hoc tests found RING (2.55) was less preferred than all other fingers (all $p < .005$). Differences between other fingers suggest a partial ranking with INDEX (4.27) preferred over MIDDLE (3.49) and LITTLE (3.37) (both $p < .0005$), and THUMB (3.91) preferred over LITTLE ($p < .05$). There was a main effect of FORM on preference ($F_{1,19} = 33.675$, $p < .0001$, $\eta^2 = .639$). Hand form OPEN (3.90) was preferred over CLOSED (3.13). There was also a main effect of HAND on preference ($F_{1,19} = 18.166$, $p < .0005$, $\eta^2 = .489$). RIGHTHAND (3.60) was preferred over LEFTHAND (3.44). This may be attributed to all participants being right-handed.

There was a large interaction of FINGER and FORM on preference ($F_{4,76} = 17.249$, $p < .0001$, $\eta^2 = .476$). Most interesting is that post hoc tests found RING OPEN (3.3) was more preferred than RING CLOSED (1.8) ($p < .01$). No interaction was found between FINGER and HAND, FORM and HAND, or FINGER, FORM, and HAND on preference.

In the interview, three participants indicated their dislike of THUMB OPEN, explaining the hand could hit the screen when pressing the upper keys; two participants expressed their aversion to MIDDLE CLOSED due to bad social implications.

3.2.6 Discussion

Our study results show that Finger-Aware Shortcuts, performed with any finger with either open or closed hand forms, are acceptable in terms of speed. A consistent learning effect and low error rate suggest the technique is easy to learn and perform. Pressing left and right areas of the keyboard have comparable time for all fingers, regardless of their home areas. As expected, pressing keys in the home area is significantly faster than left or right area, but the difference is reasonably small. The significant difference between reference tasks suggests that Finger-Aware Shortcuts will be faster in pointing-intense applications compared to typing-intense applications. However, both times are within a reasonable and practical range. Overall, this study shows that pressing a single key in 16 different ways is feasible, supporting our goal of increasing keyboard expressivity and availability.

Guideline for Selecting Hand Postures

We use the time and preference results to provide a three-tier guideline for selecting postures. Tier 1 are most recommended postures and Tier 3 are least recommended (see green and red shading in Figure 3.2 and Figure 3.3). Since time and preference are symmetric between hands, we simplify our discussion to 5 fingers \times 2 forms regardless of hand.

The index finger has consistently high performance and preference for open and closed forms, so it is Tier 1. The thumb is comparable, but people with larger hands hit the screen when using the open form, so we place thumb closed in Tier 1 and thumb open in Tier 2. Although the performance of the middle, ring, and little fingers with an open hand are good, they are less preferred than the index finger and thumb. We place them in Tier 2. When the hand is closed, the middle finger has similar measures compared to the little, but this posture has negative social implications, so it is Tier 3. The ring finger with closed form is least preferred, so it is also Tier 3.

In summary:

- Tier 1 postures are the index finger (hand open/closed) and thumb (hand closed) – they should be mapped to most frequently used commands.
- Tier 2 postures are the thumb (hand open), middle finger (hand open), ring finger (hand open), and little finger (hand open/closed) – they should be mapped to less frequent, secondary functions.
- Tier 3 postures are the middle finger (hand closed) and ring finger (hand closed) – they should be avoided.

3.3 Implementation

In this section, we describe an algorithm that identifies the hand and finger on the keyboard when a key is pressed. The algorithm recognises all postures that are in Tier 1 and 2 of our guideline. We used the same apparatus as Experiment 1, including the web camera reflector (Figure 3.4a) and the green material covering the keyboard face that facilitates keyboard localisation (Figure 3.4b) and background subtraction (Figure 3.4c). Our simple prototype can be easily

reproduced and might be further improved with dedicated sensors and improved background models.

3.3.1 Keyboard Localisation

To reliably determine which finger is pressing a key, we used a keyboard cover with a slightly different shade than the green laptop skin. The cover colour was sampled in advance. We used this information to extract the keyboard area and automatically localise the four corners of the keyboard by approximating a quadrilateral. The perspective of the image frame was corrected by rectifying the quadrilateral to a rectangle. Key positions were mapped from the physical keyboard to coordinates in this rectangle.

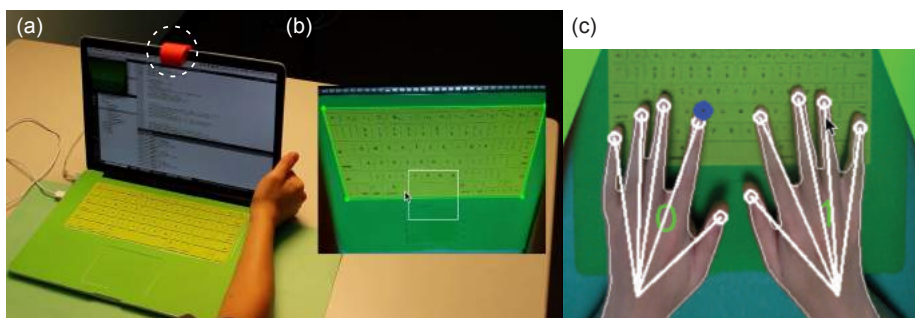


Figure 3.4: Apparatus: (a) A reflector directs the web camera to watch the keyboard; (b) a slight difference in colour between the keyboard cover and laptop skin helps extract the keyboard location; (c) aided by a green keyboard cover and laptop skin, the hands are isolated and tracked in a rectified frame.

3.3.2 Hand Feature Extraction

Hand skin and background samples from Experiment 1 were converted to the Hue-Saturation-Value (HSV) colour space to train a Gaussian naive Bayes classifier, which identifies which pixel is likely to be part of the skin. The identified skin pixels are often disconnected or contain false positives. Performing a morphological close followed by an open reduces noise and brings each hand contour together. The contours are then smoothed by sequentially applying blurring, dilating, thresholding, and eroding. To eliminate dark areas between

two fingers, Sobel operators are applied to the contour areas to identify sharp changes in derivatives of brightness. The contours are smoothed one more time.

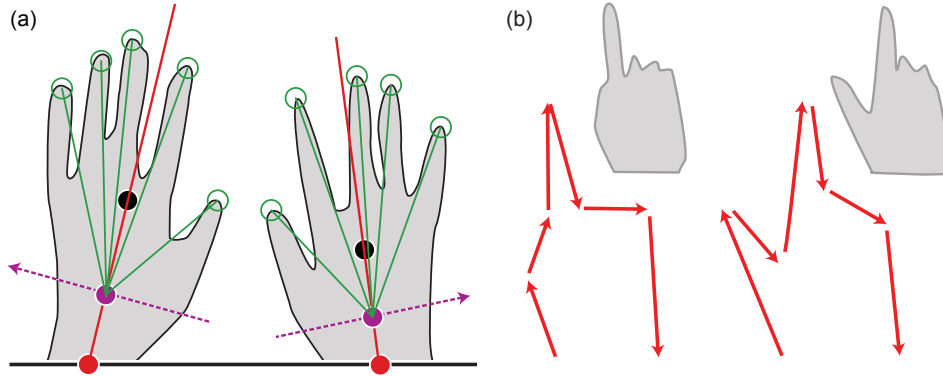


Figure 3.5: Algorithm: (a) hand feature extraction showing wrist reference points as red dots, palm centre as purple dot, principle axis as red line, and perpendicular axis as dashed purple line; (b) stroke recognition approach for hand and finger identification.

Two points and two axes are calculated from each hand contour to facilitate identifying fingers and hand shape. With the assumption that computer users always interact with the keyboard by reaching their hands forward, we calculate a *wrist reference* by simply taking the intersection midpoint between the forearm and the bottom of the camera’s view (Figure 3.5a). We also estimate the *palm centre* by calculating a weighted average of the wrist reference and the centroid of the hand contour. A *principal axis* is formed with the vector from the wrist reference to the palm centre, which provides an estimation of the orientation of the hand. A *perpendicular axis* is perpendicular to the principal axis at the palm centre pointing outwards. This axis is used to reduce false positives during fingertip detection and provide a reference for sorting vectors of the wrist reference and fingertips.

3.3.3 Fingertip Localisation

A similar approach to Yoruk et al. [100] was used for localising fingertips. We compute distances from the wrist reference to each point on the contour in sequence, then local maxima are candidates for fingertips. False positives on the wrist reference side of the perpendicular axis are eliminated. Remaining candidates are mostly extended fingers. Bent fingers can be falsely detected

due to the slight curvature at the knuckle, but we minimise this by tuning the curvature threshold.

3.3.4 Hand and Finger Identification

Using the extracted hand features and localised fingertips, we applied two approaches for hand identification: stroke recognition and heuristic rules. Both distinguish between left and right hands and open or closed forms. We used stroke recognition in Experiment 2 and heuristic rules in Demonstrations.

Stroke Recognition Approach — A hand contour can be normalised as a single “stroke” from the bottom left corner to the bottom right corner, ignoring the intersection between the forearm and the bottom of the frame. Therefore, techniques used for processing and recognising strokes can be applied to classify hand shapes (Figure 3.5b). We used 1€ Filter [13] to smooth and \$1 Recogniser [98] to recognise the stroke. The 1€ Filter is a first-order low-pass filter for reducing noisy signals. Applying it produces a more normalised stroke which aids the \$1 Recogniser algorithm. The \$1 Recogniser is a 2D single stroke recogniser. Each posture, hand side, and form is labelled with a code (such as “R01000” if right hand with the index finger extended and other fingers closed). We train the \$1 Recogniser with the filtered strokes and corresponding labels. When a single finger is extended, the finger used for pressing the key is immediately identified.

Heuristic Rule Approach — As a simple and practical alternative, we use heuristics to identify hands. Hand side is jointly determined with the relative positions of the wrist reference and the direction of the principal axis. The frame is equally divided into three parts horizontally. For example, if the wrist reference falls into the left part, it is a left hand. If the wrist reference falls into the middle part, the opposite side of the principal axis direction is the hand side. With simple heuristics, hand form is determined using the distance of each fingertip to the palm centre and their vertical differences. When the maximum of the distances is greater than all other distance by a threshold, either index or little is extended. When the difference between the maximum and minimum of the vertical differences is smaller than a threshold, the thumb is extended. In all other conditions, we consider the hand open.

If an open hand is recognised by either approach, fingertips are sorted in descending order based on the angle between the perpendicular axis and the

vector from the palm centre to each fingertip. A larger angle indicates the finger is closer to the thumb. We compare the distance from each finger to the key and identify the finger using its order.

3.3.5 Hand and Finger Tracking

We used a Kalman Filter to track each identified fingertip. The filter models fingertip movement as uniform motion. When a new observation is not presented, a fingertip position is predicted. We used dynamic programming to match new fingertip positions with predicted positions. Cartesian distances were employed to measure the difference between two positions. The dynamic programming reduces the computational cost by assuming the mapping from one sorted fingertip to another is monotonic. It also yields the best matching score by calculating the minimum distance sum of two hands. With the best matching scores, we could also trivially track hands.

3.3.6 System-Wide Background Service

We built our algorithm as a background service in Apple OS X. The service has two parts: an OpenCV posture recogniser and a Cocoa key event interceptor. The interceptor traps key events from the operating system using Quartz Event Services. When a key down is received, it sends the key code and timestamp to an intermediate in-memory event queue. The recogniser receives the event and searches in a 3s cache for an unprocessed frame with the closest timestamp. The posture is then recognised and the interceptor is notified via another queue. The interceptor rewrites the event according to a configurable mapping between Finger-Aware Shortcuts and equivalent shortcut keys. The event is fired immediately after receiving the posture or releasing the key. Processing a shortcut using our background service takes approximately 77ms, empirically shorter than the duration of a key press and hardly perceivable. With accessibility features in OS X, we can also detect which application is active and interpret key events as application-specific commands.

3.4 Experiment 2: Comparing with Shortcut Keys

The goal of this study is to compare Finger-Aware Shortcuts with current keyboard shortcut approaches. Time and error were used as primary performance metrics, but we also examined cognitive differences using perceived workload and frequency of viewing the command mappings. The general format of the experiment was similar to Experiment 1.

3.4.1 Participants

A subset of 8 participants (3 female) from Experiment 1 were recruited, age ranging from 23 to 37 ($M=25.6$, $SD=4.7$).

3.4.2 Apparatus

The same apparatus as the previous experiment were employed. The stroke-based recognition algorithm (described above) was used in real-time.

3.4.3 Procedure

In each trial, participants completed a reference task followed by a shortcut task. The reference task simulates real-world scenarios of keyboard interactions with the same TYPING and POINTING reference tasks from Experiment 1.

Our main focus is the shortcut task where participants issue one of 6 “commands” using a shortcut technique: either Finger-Aware Shortcuts or a standard keyboard shortcut method. The commands were represented as two sets of three four-letter English words. All words in each set begin with the same letter (e.g. W: *Wack, Wool, Whim*, P: *Peek, Pill, Pump*).

We tested for 3 variations representing how commands are typically mapped to standard keyboard-only shortcuts:

3KEY: 3 Keys Without Modifier — The set of commands are mapped to different keys without any modifier key. Only one key can correspond to the first letter of the command, other keys are chosen arbitrarily. Using the ‘W’ example

set, this could create the following mapping: **W** for *Wack*; **S** for *Wool*; and **R** for *Whim*. This variation is based on a common method for mapping shortcut keys in drawing applications where text entry is modal, for example in Adobe Illustrator, **S** means *Scale*, **V** means *Select*, and **K** means *Slice*. We expect **3KEY** will be fast since it is simply pressing a key, but there may be cognitive overhead from the conflicting key and the first letter of the word.

3KEY1MOD: 3 Keys using 1 Modifier — The set of commands are mapped to different keys using a single modifier key. As above, only one key can correspond to the first letter of the command. We use the **⌘** (command) modifier. Using the ‘W’ example set, this could create the following mapping: **⌘+W** for *Wack*; **⌘+D** for *Wool*; and **⌘+A** for *Whim*. This variation is based on an analogous common shortcut key mapping: **⌘+C** for *Copy*; **⌘+X** for *Cut*; and **⌘+W** for *Close*. We expect **3KEY1MOD** to be slower than **3KEY** due to the extra modifier press, but the same conflict between word and key letter could introduce cognitive overhead.

1KEY3MOD: 1 Key using 3 Modifiers — All commands in a set are mapped to the same key using three different modifier key combinations. The key corresponds to the common first letter in the command set, for example *Wack*, *Wool*, *Whim* are all mapped to **W**. Three modifier key combinations were used: **⌘** (command), **⌘+⇧** (command + shift), and **⌘** (alternate). For the example set, this creates the following mapping: **⌘+W** for *Wack*; **⌘+⇧+W** for *Wool*; and **⌘**+**W** for *Whim*. This variation is based on applications that use different modifier keys to map related commands to the same key, such as **⌘+S** for *Save*; **⌘+⇧+S** for *Save As ...*; and **⌘**+**S** for *Save Copy*. We expect **1KEY3MOD** to be slower than **3KEY**, but it may have less cognitive overhead without a conflict between word and key letter.

We compare these to Finger-Aware Shortcuts:

FINGERAWARE: 1 Key with 3 Finger Aware Postures — All commands in a set are mapped to the same key using three different postures. The key corresponds to the common first letter in the command set, for example, *Wack*, *Wool*, *Whim* are all mapped to **W**. Three postures were used: **INDEX OPEN**, **INDEX CLOSED**, and **MIDDLE OPEN**. All postures were performed with the right hand. These postures were selected according to the guideline derived from the formative study: two postures are from Tier 1 (**INDEX OPEN**, **INDEX CLOSED**) and one Tier 2 (**MIDDLE OPEN**). Like **1KEY3MOD**, there is no conflict between word and key letter. However, we expect it to be slower than **3KEY** given overhead for using a specific

finger and posture. The question is whether it is comparable to 1KEY3MOD or 3KEY1MOD since it could be an alternative to these common techniques.

Common Task Details

With every SHORTCUT, the two sets of 3 commands are mapped to a subset of 12 keys: 6 keys on the left (W, E, R, S, D, F) and 6 on the right (P, O, I, L, K, J). The two command sets were chosen such that the first letter of one set matched one of the left keys and the other matched one of the right keys. Parts of the keyboard are denoted with KEYPART (LEFTPART, RIGHTPART).

For single key mappings (1KEY3MOD, FINGERAWARE), all commands in a set were assigned to the key matching the first letter. For multiple key mappings (3KEY, 3KEY1MOD), one command from each set was assigned to the key matching the first letter. The remaining key mappings were randomly selected without duplicates from the remaining 10 keys.

All shortcut mappings were displayed on a cue card, rendered as two columns of 3 commands. The cue card could be viewed anytime by pressing SPACE, but participants were encouraged to memorise the mappings. The current trial was restarted if the cue card was viewed. We logged the number of view (CC Count) and viewing duration (CC Duration) as additional factors to test learning effects.

Error and Time

We are only concerned with errors during shortcut tasks. In the experiment, an error occurred when wrong shortcut keys were used for the prompted command word. Participants immediately repeated the same trial beginning from the reference task for a maximum of 3 tries. This repeated trial design maximises error-free trials for analysis. With FINGERAWARE, errors might also be caused by incorrect posture recognition. These errors appeared as trial errors to participants, but they were informed that FINGERAWARE could have false negatives prior to the experiment. We manually corrected for these false errors (accounting for 10.9% overall) before analysis. Most were due to capture quality issues such as fingers cropped by capture area, shadows, and motion blur, which could be addressed with a wider angle, faster, and more sensitive camera. Others were caused by one hand occluding the other, which could be addressed with a depth sensor.

3.4.4 Design

The experiment design was within-subject, repeated measures, and full factorial. All trials for each variation of SHORTCUT were presented together, with their order counterbalanced using a balanced Latin Square. For each SHORTCUT, participants performed 5 blocks of measured trials. Each block presented all trials for all combinations of COMMANDS and REFTASKS in random order for 3 REPETITIONS.

In summary: 4 SHORTCUTS \times 5 BLOCKS \times 6 COMMANDS \times 2 REFTASKS \times 3 REPETITIONS = 720 trials per participant.

3.4.5 Results

The same analysis method was used as Experiment 1. CC Count and CC Duration were aggregated using sum. All subjective data were transformed with Aligned Rank Transform.

Learning Effect

BLOCK had a main effect on time ($F_{4,28} = 14.756$, $p < .0001$, $\eta^2 = .678$). Post hoc tests found block 1 significantly slower than blocks 4 and 5, and block 2 significantly slower than block 4 (all $p < .05$). BLOCK had a main effect on error ($F_{4,28} = 5.298$, $p < .005$, $\eta^2 = .431$), but no post hoc differences. BLOCK had a main effect on CC Duration ($F_{1,21,8.46} = 54.196$, $p < .0001$, $\eta^2 = .886$). Post hoc tests show block 1 had a longer duration than all other blocks (all $p < .005$). Given these learning effects, only blocks 4 and 5 are used in subsequent analysis.

Error Rate

Excluding false negatives in FINGERAWARE, the mean error rate across all participants was 8.7% with $SD=5.53\%$ (3KEY 8.9%, 3KEY1MOD 9.0%, 1KEY3MOD 9.9%, FINGERAWARE 6.9%). There was no main effect of SHORTCUT or REFTASK on error, nor was there a main effect of SHORTCUT on error for REFTASK. Analysing each SHORTCUT separately found a main effect of REFTASK on error in 3KEY ($F_{1,7} = 40.111$, $p < .0005$, $\eta^2 = .851$): TYPING ($M=12.15\%$, $SD=5.34\%$) had a significantly higher error rate than POINTING ($M=5.56\%$, $SD=6.47\%$).

Time

The mean time across all participants was 629ms (SD=190). Overall, there was a main effect of SHORTCUT on time ($F_{1.54,10.81} = 9.809$, $p < .0005$, $\eta^2 = .584$). Post hoc tests showed 3KEY faster than 1KEY3MOD and FINGERAWARE both by 421ms ($p < .005$), and 3KEY1MOD faster than 1KEY3MOD and FINGERAWARE, both by 377ms ($p < .05$).

Analysing each REFTASK separately (Figure 3.6), SHORTCUT had a main effect on time in POINTING ($F_{3,21} = 10.292$, $p < .0005$, $\eta^2 = .595$) and in TYPING ($F_{1.58,11.05} = 7.676$, $p < .05$, $\eta^2 = .523$). Post hoc tests revealed that in POINTING, 3KEY was significantly faster than 1KEY3MOD by 303ms and FINGERAWARE by 329ms; 3KEY1MOD was significantly faster than 1KEY3MOD by 238ms and FINGERAWARE by 264ms (all $p < .05$). In TYPING, 1KEY3MOD was significantly slower than 3KEY by 539ms and 3KEY1MOD by 515ms (both $p < .05$).

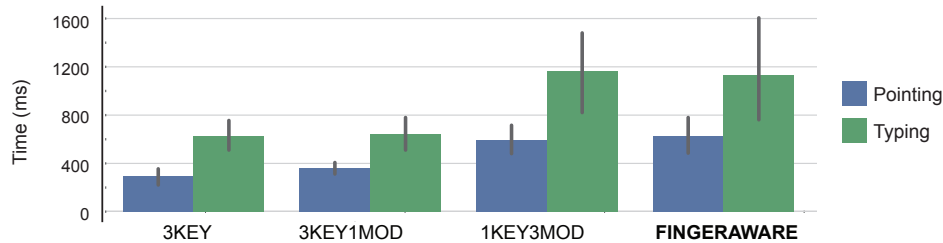


Figure 3.6: Time by SHORTCUT technique by reference task.

Perceived Workload

At the end of the experiment, NASA-TLX was used to elicit participants' perceived workload. Each SHORTCUT technique was measured in six dimensions on a scale from 0 (very low) to 20 (very high). Mental demand, physical demand, performance, effort, and frustration for each SHORTCUT were: 3KEY 8.1, 4.4, 8.6, 7.3, and 6.6; 3KEY1MOD 12.6, 10.3, 9.8, 10.9, and 10.8; 1KEY3MOD 13.9, 13.1, 11.4, 13.4, and 12.9; FINGERAWARE 14.0, 12.7, 13.0, 12.7, and 13.9, respectively.

SHORTCUT had a main effect on mental demand ($F_{3,21} = 3.771$, $p < .05$, $\eta^2 = .350$), with post hoc tests showing 3KEY had lower demand than 1KEY3MOD and FINGERAWARE (both $p < .05$). SHORTCUT had a main effect on physical demand

($F_{3,21} = 10.595$, $p < .0005$, $\eta^2 = .602$), with post hoc tests showing 3KEY had lower demand compared to all others (all $p < .05$). SHORTCUT had a main effect on effort ($F_{3,21} = 6.785$, $p < .005$, $\eta^2 = .492$), with post hoc tests showing 3KEY had lower effort than 1KEY3MOD ($p < .005$) and FINGERAWARE ($p < .01$). SHORTCUT had a main effect on frustration ($F_{3,21} = 4.152$, $p < .05$, $\eta^2 = .372$), with post hoc tests showing 3KEY had lower frustration than FINGERAWARE ($p < .05$). There was no main effect of SHORTCUT on performance.

Memorisation

A real number continuous scale was provided for each SHORTCUT, describing the ease of memorising the command-shortcut mappings. The scale is from 1 (hardest to remember) to 5 (easiest to remember). The results were 3KEY 4.0, 3KEY1MOD 2.2, 1KEY3MOD 2.8, and FINGERAWARE 3.0. There was a main effect of SHORTCUT on ease of memorisation ($F_{3,21} = 3.423$, $p < .05$, $\eta^2 = .328$), with post hoc tests showing 3KEY easier to remember than 3KEY1MOD ($p < .05$).

Participants were asked about their memorisation strategy for each SHORTCUT. Their descriptions fall into three categories: *a*) memorising the order of shortcuts and commands independently, then using orders to establish a mapping; *b*) using mnemonics; and *c*) using no special technique. In 3KEY, six participants used *b*, one used *a*, and one used *c*. In 3KEY1MOD, four used *b*, three used *c*, and one used *a*. 1KEY3MOD and FINGERAWARE had the same result: six used only *a*, one only used *b*, and one used both *a* and *b*.

3.4.6 Discussion

Our finding that single key shortcuts are fast was expected – this is simply pressing a key. Despite the conflicts between command letters and key letters, such strong performance when mapping single keys with or without one modifier is surprising. We expected a cognitive processing overhead, but the result suggests this was less of a concern. However, this benefit may be due to the relatively small number of commands and short training time. It is possible that increasing the number of commands or performing a longer study could reveal different patterns. Regardless, in real world usage, single key shortcuts are limited to non-text modes and both single key techniques are limited in the number of available keys.

It is encouraging that Finger-Aware Shortcuts achieved similar performance with one key with multiple modifiers. As reflected by the memorisation patterns used by participants, the two techniques are analogous in that different postures act like different modifier keys. It is important to recognise that using specific fingers and postures to press keys is not currently a well-practised task. With more time, advantages for motor memory are likely to develop, further decreasing performance time. In addition, participants explained that higher frustration scores with Finger-Aware Shortcuts were due to occasional false detection in the computer vision system. This likely negatively impacted performance. Since participants are already familiar with multiple modifier shortcuts, the comparable performance of Finger-Aware Shortcuts is notable considering the short exposure and “research-quality” posture recognition.

Overall, our study confirmed that Finger-Aware Shortcuts have similar performance to a common and necessary class of shortcut keys with multiple modifier keys and a single alphanumeric key. With further improvements to recognition robustness and more practice, our technique could be a viable alternative to multiple modifier keys by substituting modifiers with postures. It could also enable single shortcut keys in text-focused applications by simply using the thumb to press keys while typing. It could even be combined with a single modifier key to double the number of single key mappings.

3.5 Demonstrations

We demonstrate Finger-Aware Shortcuts for complementary commands, up/down controls, and alternative modifiers in *Adobe Photoshop* and *Google Docs*. We also extend for parameter control during a shortcut press and two-handed shortcuts with a single key.

3.5.1 Command Mappings

Complementary Command

Finger-Aware Shortcuts enable the same key to overload a family of related commands. In Google Docs, pressing H with the left index finger, open hand to change the style of the current line to Heading 1, the middle to Heading 2, and

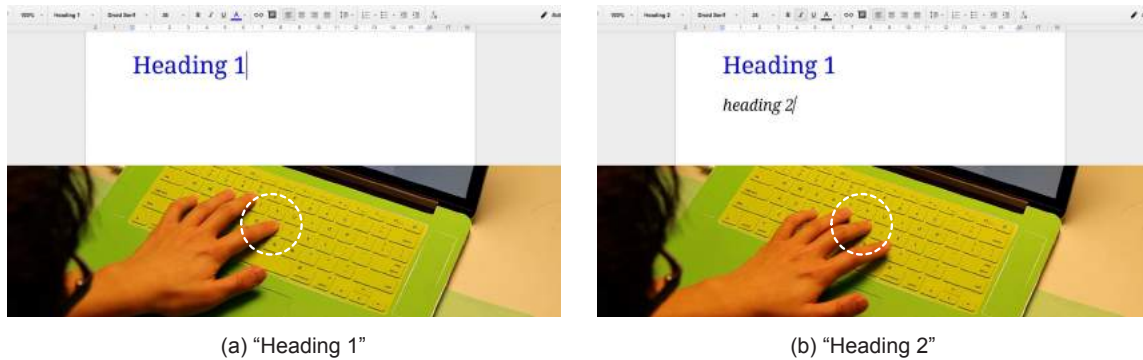


Figure 3.7: Complementary command – in Google Docs, (a) pressing **[H]** with the left index finger, open hand changes the style to “Heading 1”; (b) left middle finger changes the style to “Heading 2”.

the ring to Heading 3 (Figure 3.7). The text is set to normal with the left thumb, closed hand.

Up/Down Control

By mapping two fingers to up and down, then pressing related keys, numerical values of different properties can be adjusted. In Photoshop, pressing **[B]** with the right index, open hand decreases the brightness of an image, while the right middle increases the brightness (Figure 3.8). Similarly, pressing **[C]** with different fingers adjusts contrast and pressing **[O]** adjusts opacity. Up/down controls can be mapped to a non-dominant hand when pointing is intense. When drawing with the brush tool in Photoshop using the right hand, pressing **[H]** with the left index or middle finger using an open hand decreases or increases the hue of the brush colour.

Alternate Modifier

Another way is to map keyboard shortcut modifier combinations to different postures. For example, mapping the **[⌘]** modifier to the right thumb, closed hand. All existing keyboard shortcuts using only **[⌘]** can then be triggered by pressing the same key with this thumb posture. In Photoshop, users can *Open File* by pressing **[O]** and *Save* by pressing **[S]**, both with the thumb posture.

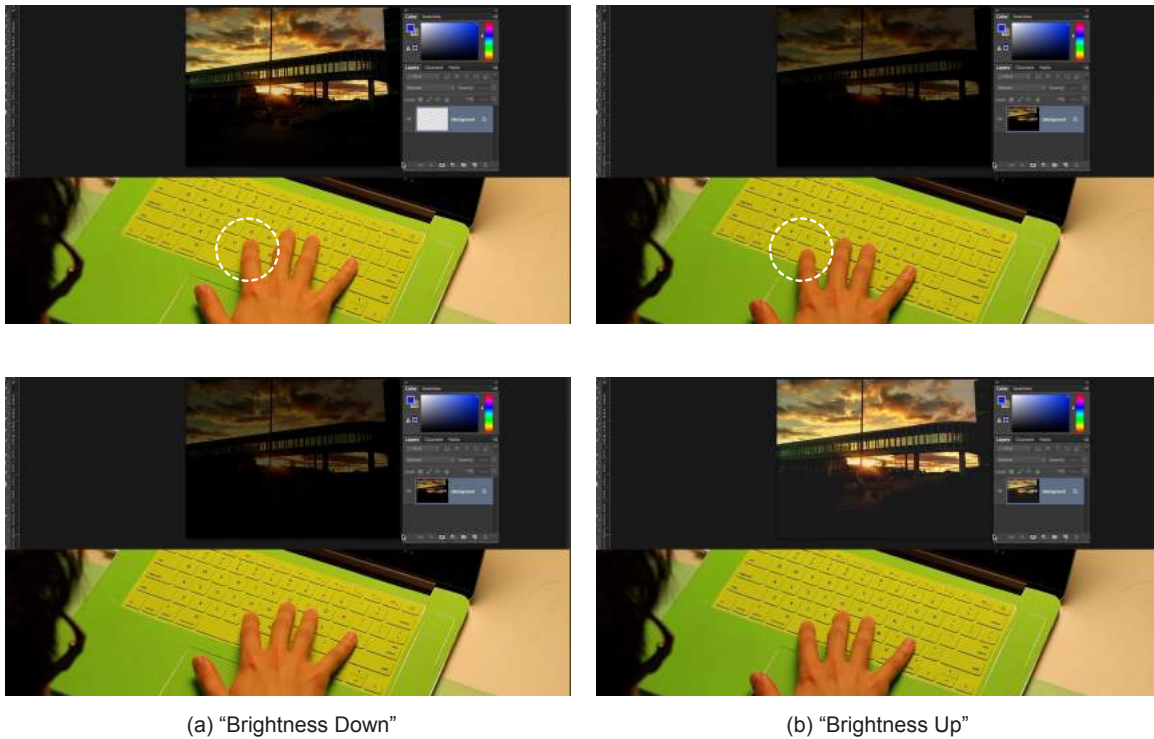


Figure 3.8: Up/down control – in Photoshop, (a) pressing **B** with the right index finger, open hand decreases the brightness; (b) whereas the right ring finger increases the brightness.

Similarly, **⌘**+**↑** can be mapped to the right index and closed hand, so pressing **N** with that posture invokes *Create a New Layer*. In Google Docs, *Bold*, *Italic*, and *Underline* can be triggered in the same way using the right thumb (Figure 3.9).

3.5.2 Extensions

Simultaneous Parameter Control

It is possible to track finger movement to control parameters during a key press. For example, pressing and holding a key with the index finger with an extended thumb, then moving the thumb to adjust a continuous parameter. For example, adjusting an RGB colour by pressing and holding **R** to adjust red, **G** to adjust green, or **B** to adjust the blue (Figure 3.10).

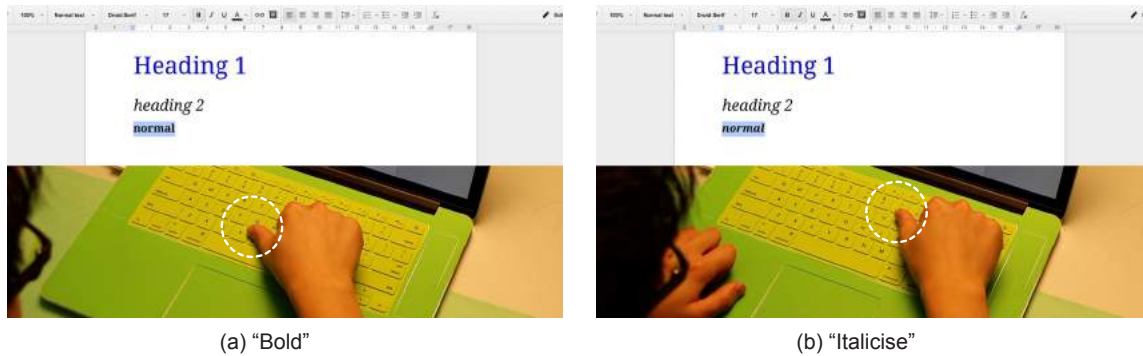


Figure 3.9: Alternate modifier – in Google Docs, the right thumb, closed hand can be treated as an additional ⌘ modifier key: (a) pressing B triggers $\text{⌘} + \text{B}$ “Bold”; (b) pressing I triggers $\text{⌘} + \text{I}$ “Italicise”.

Thumb movement can be discretised into menu options for variations of a command. For example, holding P with the index finger displays a partial pie menu with items selected with the thumb, like *Print to Printer* and *Print to PDF*. We will further explore such interaction in the next chapter.



Figure 3.10: Simultaneous parameter control – pressing R , G , B with the right index finger and the thumb extended could achieve simultaneous parameter control of RGB colour values.

Two-Handed Key Press

Special or uncommon commands could even be triggered with two-handed key presses. For example, using both index fingers to simultaneously press L to *Log Out* (Figure 3.11).



Figure 3.11: Two-Handed Key Press - pressing `L` with two hands could trigger the uncommon action “Log Out”.

3.6 Summary

This chapter has introduced the idea of Finger-Aware Shortcuts and showed it is a viable method for augmenting current shortcut keys, and perhaps used as an alternative to shortcut keys requiring multiple modifiers.

The recognition algorithm is robust enough for conducting an experiment and demonstrating the technique, but infallible hand tracking in all environmental conditions remains a barrier for full deployment. If we use an alternate capture set up with the assumption that it could be built into future laptops, hand tracking could be improved further. For example, pairing stereo cameras with infrared illumination and a laptop and keyboard with a retro reflective coating [43] would dramatically increase the reliability of hand and finger extraction.

In addition to technical improvements, our findings regarding shortcut mappings and cognitive load require more targeted studies to control and fully understand. A direct comparison between mappings based on fingers and keys under many different variations of command names, keys, and cue card access would be an informative next step. Besides, we can also actively improve our technique to facilitate the learning of finger-aware shortcut keys, which we will discuss in more detail in the next chapter.

Chapter 4

FingerArc and FingerChord

In the previous chapter, we have demonstrated how Finger-Aware Shortcuts can enable more expressive keyboard interaction. The experiment that compared Finger-Aware Shortcuts with conventional shortcut key mappings reveals good performance in learning given a small number of commands. Yet, it remains challenging for users to memorise a substantial number of commands to make effective use of Finger-Aware Shortcuts.

As an extension, we present FingerArc and FingerChord, both providing an intermediate step between using menus and directly activating hand posture based keyboard shortcuts. The intermediate step provides contextual, dynamic visual guidance to users to help them specify and complete their actions. In addition, our techniques reduce the need for modifier keys and alleviate the risk of making errors with a common mechanism for cancellation.

In this chapter, we first describe an interview study with expert computer users to identify the key issues with conventional keyboard shortcuts. Then, based on the findings from the interview, we present the design goals, concept, and proof-of-concept implementations of FingerArc and FingerChord. With these preliminary investigations, we hope to start making finger-aware keyboard shortcuts more usable, learnable, and explorable.

4.1 Interview with Expert Computer Users

In this section, we report on a qualitative interview study with expert computer users to identify what hinders them from learning, using, and exploring keyboard shortcuts. We are most interested in why users learn and use one type of keyboard shortcuts rather than another. The results of this study provide us with a better understanding of current issues of keyboard shortcuts that can be addressed with finger-aware techniques.

4.1.1 Participants

We recruited 12 participants (4 female, 9 right-handed), ages ranging from 21 to 39 years ($M=26.9$, $SD=5.9$). Each participant is associated with a unique identifier from [P1] to [P12]. Participants reported extensive use of computers, weekly usage ranging from 42 to 84 hours ($M=61.3$, $SD=10.6$). In addition, they all had formal education or industry experience in computer science, therefore we consider them to be expert computer users.

Regarding operating systems, participants reported regular use of Windows (7 participants), macOS (7 participants), and Linux (3 participants): all had experience with at least two; nine used all three; and five participants frequently switched between two different operating systems.

Participants also reported experience with a variety of desktop, web, and terminal programs, including text-intensive applications (e.g. Google Docs, Microsoft Word, Sublime Text, Vim), graphics-intensive applications (e.g. Adobe Photoshop, GIMP, Blender, Autodesk 3ds Max), integrated development environments (e.g. Visual Studio, Xcode, Eclipse, Unity), and other common software (e.g. Google Chrome, Safari, Adobe Acrobat Reader, Skype).

4.1.2 Procedure

Participants were first asked to complete a questionnaire eliciting demographic data and computer experience. Semi-structured interviews were then conducted to probe for different aspects of keyboard shortcut interaction, including basic usage, learning, physical articulation, and feedback. The interview questions were formulated based on an analysis to keyboard shortcuts with

Norman's seven stages of action framework [67]. Of particular importance, we focused on eliciting the types of keyboard shortcuts that participants tend to learn, use, or forget.

Upon completion of the interviews with 12 participants, we reviewed the data to ensure saturation had occurred – the same patterns were recurring, and it was not likely to obtain additional information by recruiting more participants [21]. Thematic analysis [11] was then employed to identify the themes within our data.

4.1.3 Results

We identified nine factors that impact users' learning, use, and exploration of keyboard shortcuts:

1. **Frequency** of triggering the same command.
2. **Perceived cost** of graphical input versus keyboard shortcuts.
3. **Visibility** of keyboard shortcuts.
4. **Semantic alignment** to the associated command.
5. **Difficulty** of pressing multiple keys, together or in multiple steps.
6. **Consistency** across applications and operating systems.
7. **Perceived risk** of making errors.
8. **Feedback** to keyboard shortcuts.
9. **Customisation** to keyboard shortcuts.

We present the analysis of each factor in detail and report our findings with the support of quotes from the participants.

Factor 1: Frequency of Triggering the Same Command

Comments from participants suggest that frequent activation of a command motivates the learning of a keyboard shortcut: “for functions like Bold and Italicise – just as I write documents, I often need to emphasise stuff – they are common enough that I feel I need to learn them” [P5]; and the use of a keyboard shortcut: “for me to do something a lot of times, I would always prefer hitting the keyboard shortcut” [P9].

Also, learning keyboard shortcuts typically occurs naturally with repeated usage: “it’s not like a dedicated learning process where I spend ten minutes just for memorising keyboard shortcuts” [P3]. Because of the gradual nature of learning, infrequent commands are more likely to be forgotten: “I’d always pay attention to the keyboard shortcut while clicking on the menu, and realise [that keyboard shortcuts would have been faster]; but by the time I reuse the same command, I would already forget the shortcut” [P11].

Some participants suggested they are consciously biased towards using keyboard shortcuts if they intend to learn: “I’d make a conscious effort trying to use the keyboard shortcut once I’m aware of it” [P4]. Some would go even further: “if I find myself repeatedly clicking on the menu, I would either find a way to hide the menu item or completely disable it [to force myself to use its keyboard shortcut]” [P5].

Frequency of use also affects the development of automaticity in activating keyboard shortcuts. For very common actions such as *Copy*, *Paste*, and *Save*, all participants acknowledged that they have developed muscle memory – keyboard shortcuts can be articulated without much cognitive effort. In contrast, less frequent keyboard shortcuts require a conscious association to the keys being pressed: “for the things that I use less often like Bold in Google Docs, I often have myself being like ... OK, I have to use Ctrl+B, so it’s a little bit more thoughts into it, and it’s just because it’s a less frequent action” [P5].

Factor 2: Perceived Cost of Graphical Input versus Keyboard Shortcuts

Switching to keyboard shortcuts is often motivated by the inconvenience of graphical input: “using your fingers to trigger keyboard shortcuts is way more efficient than just clicking everywhere with a mouse” [P3]. Indeed, users may

unconsciously compare the perceived costs between using a mouse and a keyboard. The perceived costs, as an estimation of each action's physical and cognitive effort, drive users to employ one method from another. While the costs are certainly relative to users' experience, they are also relevant to the specific interaction context.

By nature, graphical widgets, including context menus, toolbars, and menus, require different extents of effort. In many situations, especially for infrequent actions, participants found it convenient enough to use context menus: "if I can do the task with a right click [and a context menu], I might not bother to remember the keyboard shortcut" [P7]; or toolbars: "if it's just on the toolbar, I probably will have to use it a lot of times before I search for the shortcut" [P10]. In these cases, the perceived cost of graphical input is relatively low.

In contrast, menus are considered less efficient, therefore a higher perceived cost: "I feel it's more likely for me to use keyboard shortcuts if the commands are hidden in a menu" [P9]. Further, if an application is designed in favour of keyboard access, the cost of graphical input becomes very high: "in Vim, there's no other choice than using keyboard shortcuts, so I have to learn them" [P7].

The perceived costs are also related to the hand positions *before* and *after* the command action. When two hands are both on the keyboard, providing graphical input is costly: "... you have to move your hand back to the mouse, which is much more work than just using the shortcut" [P3]. However, when one hand is already on the mouse, accessing graphical widgets becomes easier, which discourages the learning and use of keyboard shortcuts [P2,8,12]. If subsequent interaction requires text entry, keyboard shortcuts are more likely to be used [P11]; whereas if requiring a mouse, graphical input is more likely to be used: "keyboard shortcuts like Ctrl+P pops up a print settings dialogue - I have to use my mouse anyway" [P7].

Factor 3: Visibility of Keyboard Shortcuts

The lack of visibility is one major hindrance to learning keyboard shortcuts: "there's no real good way of discovering them" [P5]. Because of this, participants adopted different strategies to find out the keys to press when learning. The strategies include navigating menus [P1 - 12], hovering on toolbar icons [P1 - 12], performing menu search [P10], searching online [P1 - 12], studying crib sheets [P1,2,7,10 - 12], and watching tutorials [P2,5,6]. Although the extent may vary,

these strategies all require a certain amount of effort, therefore various levels of visibility.

The most common way of finding a keyboard shortcut is either navigating menus or hovering on toolbar icons [P1 – 12]. Given a certain command, users need to first locate the corresponding graphical element from the user interface. This is relatively easy if the learning is prompted by frequent menu or toolbar access [P10]. Yet, it might not be immediately clear if they are unfamiliar with the user interface, despite having a clear goal in mind: “for something like Ctrl+K Ctrl+C for Comment in Visual Studio, I know the command must exist [from other code editors], but I’m not sure where to find it in the menu” [P10].

For the latter, some operating systems (e.g. macOS) and applications (e.g. Google Docs) feature the ability to search commands inside the menu [P10], but this was not well known among our participants. More participants resorted to online search, often with keywords containing the application name and a description of the action that the command performs [P1 – 12]. Online search becomes more valuable for terminal applications, which heavily rely on keyboard shortcuts (e.g. Vim [P6]) or complex software with many functions (e.g. SolidWorks [P6]). In both cases, the visibility of keyboard shortcuts from the user interface alone is so low that users have to seek for external support.

Another common strategy is to use cheat sheets [P1,2,7,10 – 12]. Participants indicated that cheat sheets are most suitable for learning keyboard shortcuts they have used before: “if I’ve learned the shortcuts in the past, and I just need to quickly glance over them, then cheat sheets are the best option” [P10]. Regarding the source of cheat sheets, some participants made their own [P6,12], some searched online [P1,2,7,10,11], and others used the ones provided by the software [P9]. Many applications include the feature of quickly viewing a cheat sheet with a keyboard shortcut, for example, “in Google Docs, you can use Command+/ to see all the shortcuts available” [P10]. In addition to cheat sheets, participants also suggested watching online tutorials helps learning keyboard shortcuts, especially for feature-rich software like Visual Studio Code [P5], SolidWorks [P6], and Blender [P11].

Factor 4: Semantic Alignment to the Associated Command

Participants commonly acknowledged the importance of semantic alignment between the shortcut key and its associated command [P1 – 12]. Many suggested

the most pronounced issue with keyboard shortcuts are that they “do not make sense” [P1,2,4,6–12]. The misalignment with the associated command not only impedes learning, but also hinders the exploration to new keyboard shortcuts.

Participants indicated that keyboard shortcuts are most easily memorised and retained when the key character is the same as the first letter of a major word in the command (e.g. `Ctrl+C` for *Copy*, `Ctrl+S` for *Save*, and `Ctrl+A` for *Select All*) [P1–12]. They might even carry this expectation to unknown shortcuts of common actions, for example: “sometimes I think Ctrl+D should be delete, and I’ll do a Ctrl+D when I want to delete a line of code, but only to find out it actually does something else” [P3].

When another alphabetical key is selected instead of the first letter of a command word (e.g. `Ctrl+G` for *Find Next*), it is more likely to be confusing [P1–12], especially when using the first letter is a common expectation: “Ctrl+S to save makes sense, but to strikethrough [text], for example, you have to try to figure out what the keyboard shortcut for that is, because the one that first comes to your mind is Ctrl+S” [P6].

Even worse than arbitrary alphabetical key is when using symbol keys [P10], number keys [P11], function keys [P12], and most frequently, modifier keys [P1–12]: “anything that has to do with Ctrl, Alt, and Shift [is] very confusing: they are all in the same area and the combinations you need to hit in connection with another key is very confusing” [P6]. In addition, participants suggested the symbol representations of modifier keys on macOS further aggravates the issue: “(pointing to a modifier key symbol) is this control [or] option? I’m always confused about these things” [P8].

Factor 5: Difficulty of Pressing Multiple Keys

Participants indicated that the difficulty of pressing multiple keys also affects learning and use of keyboard shortcuts [P1–12]. This difficulty can be attributed to both the cognitive complexity of memorising multiple keys [P2–5,7–12] and the physical complexity of articulating multiple keys [P1,5–7,10]. Most participants suggested that pressing three keys simultaneously with one hand is acceptable, but more keys would be hard to memorise or articulate [P3–5,7–12].

Even though pressing more keys with both hands is possible, participants preferred single-handed shortcuts [P1,3–7,9,11,12]. Having one hand on the

mouse and the other activating keyboard shortcuts may be of particular advantage when frequently switching modes to manipulate graphical objects, for example: “when I’m using Blender, I would typically have one hand on the keyboard to switch between different tools and the other hand on the mouse to manipulate objects” [P10].

Some keyboard shortcuts require pressing keys in multiple steps (e.g. **Ctrl**+**K**, **Ctrl**+**C** for *Comment* in Visual Studio), but participants found them hard to learn [P4,5,11], for example: “there’re already too many things to remember in a keyboard shortcut, yet with these shortcuts, I’ll also have to remember their ordering” [P5].

Factor 6: Consistency across Applications and Operating Systems

Comments from participants suggest the consistency of shortcut mappings across applications and operating systems helps them learn keyboard shortcuts. Participants described those that are retained to be “system-wide” [P3,5], “ubiquitous” [P1,6,8], “universal” [P7,9], “useful” [P0,4,8,10], or “shared across different applications” [P11]. These also include the ones with obscure command mappings like **Command**+**V** for *Paste* [P1,2,3,5,9], **Ctrl**+**Alt**+**Del** for *Windows Task Manager* [P4], and **F2** for *Rename* [P12].

In contrast, keyboard shortcuts that participants tend to forget are typically specific to a certain application [P1,2,4,6,7,8,9,10,12], especially the ones that contain many shortcut commands like Adobe Photoshop [P1], Adobe Illustrator [P8], Inkscape [P4], SolidWorks [P6], Blender [P11], and Vim [P2,3]. Learning or relearning happens on an ad hoc basis based on the need to certain functions: “I’d remember a keyboard shortcut for the day that I am using it, but the next day I might already forget it” [P7]; and to certain applications: “when I stop using Photoshop on [a] day to day basis, I start forgetting its keyboard shortcuts, but if I move back to using the application again more frequently, I’d actively relearn the keyboard shortcuts, because it just saves me more time” [P11].

The consistency of keyboard shortcuts in different contexts motivates the exploration of the same shortcut in different applications: “sometimes I try the shortcut I know for other applications – I think ‘well, this seems to be pretty standard, so maybe it works in this application too’ ” [P4]. On the other hand, inconsistency impedes exploration: “I’d get confused when some standard shortcuts

are mapped differently in some other applications” [P4]. Participants raised different examples to illustrate this inconsistency: **Ctrl**+**T** does not open a new tab in Notepad++ [P4], **Command**+**Z** does not perform an undo in Vim [P8], and **Ctrl**+**/** does not comment a line in Visual Studio [P4,11]. In addition, participants also reported making errors when switching operating systems due to the inconsistency of the default modifier keys (e.g. **Ctrl** in Windows versus **Command** in macOS) [P3 – 7,10,11].

Factor 7: Perceived Risk of Making Errors

Triggering a wrong command may happen by accident when articulating keyboard shortcuts [P1,6,7,10], but it is also possible when exploring new keyboard shortcuts. Nevertheless, many participants suggested that they will use a keyboard shortcut only if they are very confident about its associated action [P1,3,5,7,8,10,11,12], for example: “I don’t really use a keyboard shortcut unless I’m 90% certain of what it does” [P5]. Indeed, the perceived risk of making errors hinders the exploration to new shortcuts, especially when the task is important: “if I already have a three-page document full of important text, then I wouldn’t try a keyboard shortcut that I’m not sure at all” [P12].

Some participants felt that making an error is undesirable even if the action is nondestructive and correctable with an undo command [P5,7,9,11]. This may be because the perceived cost of correcting the error is high: “there’s a lot of fatigue in correcting mistakes” [P6]. Yet, others suggested that they are willing to experiment with a group of shortcuts if these shortcuts simply switch between different modes [P2,11], for example: “in 3ds Max, pressing number keys switches between editing vertices, edges, and faces; I know that the numbers switch between them, but I’m not quite sure, for example, whether two is the right number; in that case, I’d try pressing the keys until I see the one I want” [P2].

Factor 8: Feedback to Keyboard Shortcuts

Participants suggested that feedback to the actions triggered by keyboard shortcuts also affects their adoption [P1,2,5,6,8 – 12]. Appropriate feedback helps them confirm their action when it is correct: “... the interaction in SolidWorks is very effective, because you have immediate feedback on what’s happened” [P6];

and indicate what command is triggered instead when making an error: “it would be easy to tell [what error I made] in Lightroom, because it pops up a toast whenever I trigger a shortcut” [P10]. Explicit feedback also encourages users to explore new keyboard shortcuts: “the toast notification in Lightroom also makes me more willing to experiment with keyboard shortcuts ... if I don’t know what the error was, then I won’t be able to learn from mistakes” [P10].

Despite this, many participants acknowledged that shortcut feedback in most applications is very poor [P1,2,5,6,8–12], especially when accidentally triggering a different action [P1,5,6,9–12]. Keyboard shortcuts may trigger actions anywhere in the user interface, but users’ focus is often on their object of interest: “knowing what command a keyboard shortcut has triggered is often very difficult unless it involves apparent UI changes; you won’t know if you accidentally triggered a keyboard shortcut which does something in the background or toggles a small button on the toolbar” [P11].

Factor 9: Customisation to Keyboard Shortcuts

A few participants suggested customised shortcut mappings compensate the drawbacks of existing keyboard shortcuts and may be more easily memorised [P4,6,10,12], for example: “I typically remember quite well shortcuts I bound myself, so if I come up with a custom shortcut, I generally remember that much better than shortcuts that are given to me” [P6].

Yet, participants also indicated that customised shortcuts may be hard to communicate with other people [P10] and difficult to find the mappings once forgotten [P1]. In addition, customisation may require more modifier keys to avoid conflicts with existing shortcuts: “when you customise a shortcut, you don’t want it to conflict with existing shortcuts; and because of that, you have to use multiple modifier keys and try different combinations of modifier keys to see which one hasn’t been used” [P10].

4.1.4 Discussion

Our interview identified nine factors that impact the adoption of keyboard shortcuts. Of these factors, the frequency of activating a command dominates learning and use, but it is primarily driven by users’ need for certain actions (factor 1).

Other factors help us pinpoint the current issues and identify implications for better shortcut design.

Overall, our findings suggest that promoting keyboard shortcuts can be achieved by lowering the cost, increasing visibility, improving semantic alignment, reducing the number of keys, advocating single-handed shortcuts, being consistent with standard mappings, decreasing the risk of making errors, providing explicit feedback, and facilitating customisation.

Further interpretation of the results indicates that consistency (factor 6) and customisation (factor 9) are more a matter of better design decisions such as conforming to the conventional shortcut mappings used in other applications and providing utilities for creating customised shortcuts. On a deeper level, more profound issues with keyboard shortcuts are the following:

1. The use of modifier keys increases the complexity of keyboard shortcuts (factor 2) and induces cumbersome hand postures (factor 5), which limits the expressivity or the number of commands can be practically bound to an action key.
2. The use of an action key that is not the first letter of a major word in the command leads to semantic misalignment and poor memorability (factor 4).
3. The drastically different acts of providing graphical input and activating keyboard shortcuts reduces the visibility (factor 3) and hinders the effective transition to expert behaviour.
4. The lack of explicit feedback (factor 8) and mechanism for cancelling a keyboard shortcut increases the perceived risk of making errors (factor 7), therefore hindering the exploration of keyboard shortcuts.

Some of our observations are consistent with other work at a high level. For example, Kim and Ritter [44] also acknowledged frequency of use as a dominant factor in learning keyboard shortcuts (factor 1); Grossman et al. [30] and Krisler and Alterman [46] found perceived cost (factor 2), visibility (factor 3), semantic alignment (factor 4), and feedback (factor 8) impact shortcut learning; Peres et al. [71] suggested perceived risk (factor 7) also contributes to the underuse of keyboard shortcuts; Malacria [56] promoted keyboard shortcuts by increasing

visibility (factor 3) and decreasing the cost (factor 2); Pietrzak et al. [73] recognised the difficulty of pressing multiple modifier keys (factor 5) and duplicated modifier keys on a mouse; and Giannidakis et al. [23] encouraged shortcut usage by increasing the visibility on toolbar icons (factor 3).

Despite this, our study adds depth to the understanding of keyboard shortcuts at a more nuanced level and recognises other factors such as consistency and customisation that are not previously identified. In addition, we synthesised our findings into a clear outline for designers and researchers to address the key issues of keyboard shortcuts. Further, the study provides us with a solid basis to formulate design goals of FingerArc and FingerChord, which we present in detail in the next section.

4.2 Concept

In this section, we describe our design goals based on the findings of the interview study, then explain in detail the concepts of FingerArc and FingerChord.

4.2.1 Design Goals

The motivation of designing FingerArc and FingerChord is to leverage the ability to detect hand postures to alleviate the existing issues with conventional keyboard shortcuts. We expect that with our techniques, keyboard shortcuts can be more usable, learnable, and explorable. Based on the results of the interview study, we present four design goals as follows.

Design Goal 1: Increase Expressivity to Keyboard Shortcuts

Pressing more than two modifier keys simultaneously is very hard to articulate with a single hand. Yet, single-handed shortcuts are of particular importance because they free the other hand for users to interact with active objects. This practically limits the number of commands feasible of being associated with one action key. To bring more expressivity to keyboard shortcuts, FingerArc and FingerChord complement conventional keyboard shortcuts that require pressing one or more modifier keys. Besides, the two compatible techniques also

increase the practical number of commands associated with one action key by eight.

Design Goal 2: Improve Semantic Alignment to Associated Commands

The poor memorability of keyboard shortcuts often stems from the misalignment between the shortcut keys and the associated actions. A typical keyboard shortcut (e.g. **Ctrl**+**V**) consists of one action key (**V**) and one or more modifier keys (**Ctrl**). Ideally, an easily memorised shortcut should use the same letter that a major word in the command starts and only one standard modifier key (e.g. **Ctrl**+**C** for *Copy*). However, when multiple commands start with the same letter, designers have to assign the new command to a different, unused action key, or add an extra modifier key. When a growing number of commands are assigned with keyboard shortcuts, the mappings become increasingly obscure. To alleviate this semantic misalignment, FingerArc and FingerChord reduce the need for pressing modifier keys by detecting whether a special hand posture is formed. Designers can group the commands starting with the same letter under the same action key to minimise this misalignment. Further, the actions associated with pressing a certain key become more predictable.

Design Goal 3: Support Physical Rehearsal of Expert Behaviour

Undoubtedly, learning keyboard shortcuts requires a considerable amount of physical and mental effort. This is largely because the current interaction paradigm does not support users to effectively rehearse them. There is a huge gulf between the physical movement of navigating menus or clicking toolbar icons and the one of activating shortcut keys (Figure 4.1a). When learning keyboard shortcuts, users need to point their cursor to the graphical widget, where the action can be completed with a simple click. Yet, they need to consciously avoid using graphical input and trigger the action with the displayed shortcut. In addition, the lack of visual guidance when articulating shortcut keys further exacerbates the issue. To support physical rehearsal of keyboard shortcuts, FingerArc and FingerChord provide an intermediate step between menu navigation and direct activation of shortcut keys (Figure 4.1b). By providing dynamic visual guidance when pressing and holding an action key, users practise the same expert behaviour with the feedback from the shortcut interface. This facilitates the smooth transition to expert behaviour.

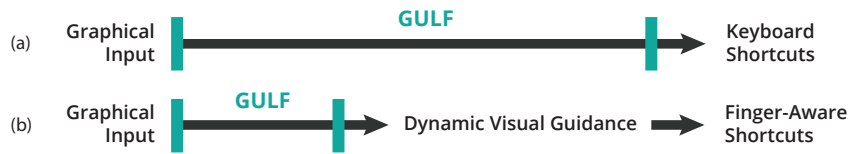


Figure 4.1: Illustration of FingerArc and FingerChord interaction model compared to traditional keyboard shortcuts: (a) a huge gulf exists between graphical input and keyboard shortcuts; (b) FingerArc and FingerChord reduces such gulf by providing dynamic visual guidance when pressing and holding an action key.

Design Goal 4: Facilitate Shortcut Command Exploration

Keyboard shortcuts are rarely learned directly through the interaction with a keyboard due to the unpredictable behaviour of activating a different action. An exception is when users expect a shortcut to trigger the same command learned from a different context or the same combination of modifier keys to modify the primary command the same way. Regardless, the explorability of keyboard shortcuts is very limited. To facilitate active shortcut command exploration, FingerArc and FingerChord provide a common mechanism for cancellation to reduce the risk of making an error. Besides, when pressing one action key, one can also view other commands bound to the same key. This prompts the discovery of new command shortcuts from the keyboard itself, rather than graphical widgets.

4.2.2 FingerArc and FingerChord

FingerArc and FingerChord are complementary to menu interaction for novice users. A novice user starts interacting with an interface by navigating through menu hierarchies. Hand posture based keyboard shortcuts are displayed next to their associated commands as plain text or as symbols. Users can activate the shortcuts in two different modes:

- **Guidance mode** is triggered when users pause on the action key for a pre-designated delay time with a special hand posture. A shortcut interface pops up after the delay. Users then interactively adjust their hand postures to select the desired command. The shortcut interface dynamically provides feedback on which option is selected.

- **Shortcut mode** is triggered when users directly press the action key with the posture that selects the command.

FingerArc and FingerChord both expect a special hand posture to differentiate between entering text or activating a keyboard shortcut. FingerArc expects the action key to be pressed using the index finger with the little, ring, and middle fingers tucked in (posture illustrated in Figure 4.2a). FingerChord expects the middle finger to press the action key and the little and ring fingers tucked in (posture illustrated in Figure 4.3a). Because the two techniques use different fingers, they are compatible with each other. In addition, they are also compatible with the conventional keyboard shortcuts with modifier keys.

FingerArc triggers the default action when the thumb is hidden under the index finger (Figure 4.2b). To select other options under the action key, FingerArc detects the relative angle between the thumb and the index finger. If the angle is greater or equal to 60° , the second command is selected (Figure 4.2c). If the angle is greater or equal to 30° and smaller than 60° , the third command is selected (Figure 4.2d). If the angle is smaller than 30° , the fourth command is selected (Figure 4.2e).

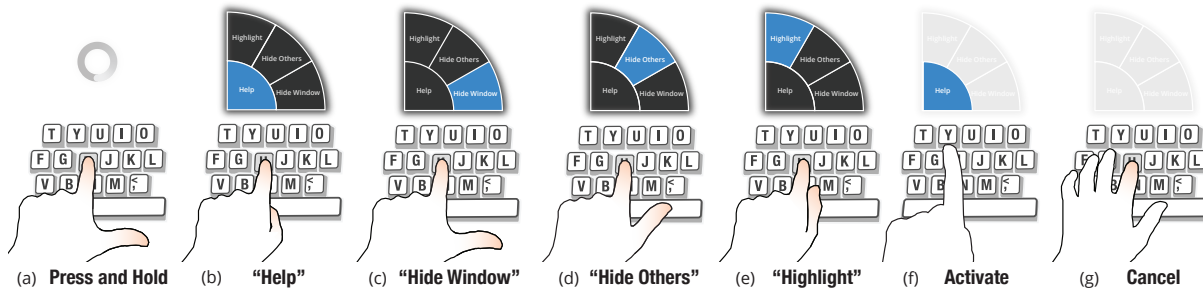


Figure 4.2: FingerArc: (a) for users who are not familiar with the shortcut, they can press and hold the action key using the index finger with the little, ring, and middle fingers tucked in for a predesignated delay time to show the shortcut interface; (b) hiding the thumb underneath selects the primary command; (c-e) posing the thumb relative to the index finger to form different angles selects other options; (f) releasing the key while maintaining the hand posture activates the selected command; (g) revealing all the hidden fingers cancels the operation. For users who are already familiar with the hand posture, they can simply press the key posing the same hand posture to trigger its corresponding command.

FingerChord allows users to select different commands by optionally pressing different key areas using the index finger. Figure 4.3 illustrates an example

that demonstrates this concept. The default action of pressing with only the middle finger selects the first command (Figure 4.3b). Subsequently pressing a key in the lower row relative to the action key selects the second command (Figure 4.3c). Similarly, pressing a key in the middle row selects the third command (Figure 4.3d) and pressing a key in the upper row selects the fourth command (Figure 4.3e).

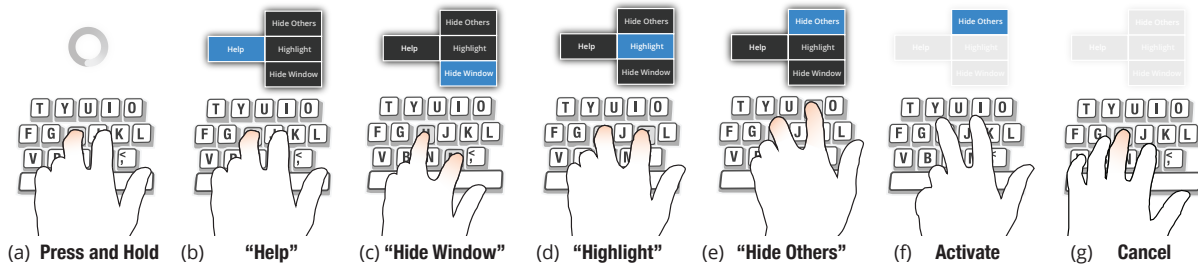


Figure 4.3: FingerChord: (a) for users who are not familiar with the shortcut, they can press and hold the action key using the middle finger with the little and ring fingers tucked in for a predesignated delay time to show the shortcut interface; (b) maintaining the hand posture selects the primary command; (c-e) using the index finger to press different key areas selects other options; (f) releasing the key while maintaining the hand posture activates the selected command; (g) revealing the little and ring fingers cancels the operation. For users who are already familiar with the hand posture, they can simply press the key(s) using the same hand posture to trigger its corresponding command.

To activate the selected command, both FingerArc and FingerChord require releasing the key while maintaining the posture (Figure 4.2f and Figure 4.3f). Additionally, if two keys are being pressed in FingerChord, they need to be released at the same time. In both techniques, cancellation of the current operation is achieved with a common mechanism that requires revealing all the fingers regardless of the current mode (Figure 4.2e and Figure 4.3e).

Compared to traditional keyboard shortcuts, we hypothesise FingerArc and FingerChord to have the following advantages, which correspond to the four design goals that we have discussed earlier:

- **Increased expressivity** – by complementing the conventional shortcuts that involve pressing one or more modifier keys, the two compatible techniques increase the number of commands under the same key by eight.
- **Increased memorability** – by associating commands with the letter key that the commands start with, and prompting users to take advantage of

more expressive hand postures, keyboard shortcuts become more easily remembered.

- **Increased learnability** – by providing an intermediate guidance mode, users can physically rehearse the expert behaviour with dynamic visual feedback, therefore transitioning to experts more effectively.
- **Increased explorability** – by affording a common mechanism to cancel an operation, these techniques reduce the risk of making an error and encourage users to explore hand posture based keyboard shortcuts as if navigating menus.

4.3 Proof-of-Concept Implementation

We developed a proof-of-concept implementation of FingerArc and FingerChord on a MacBook Pro laptop computer with a 15-inch display and a QWERTY keyboard (Figure 4.4a). A downward facing camera fixed in a 3D printed case was mounted on top of the laptop screen, recording the keyboard area. The camera was equipped with an illumination board and an infrared filter to precisely control the lighting environment. The built-in camera, reflector, and green keyboard cover in Finger-Aware Shortcuts was not applied here because the tracking accuracy was sensitive to the environment lighting change.

Three 5mm hemisphere reflective markers were attached to the fingertips of the little finger, index finger, and the thumb, respectively. A fourth marker was attached to a point on the back of the hand that forms a right angle with the markers on the index finger and the thumb. A native macOS application ran as a system-wide service, tracking the marker positions while intercepting keyboard events. Keyboard area was rectified by manually labelling the four corners of the keyboard in the software (Figure 4.4b). Heuristic rules were applied to identify hand postures in real-time (Figure 4.4c).

Our preliminary test with a number of users showed that FingerArc and FingerChord are feasible of complementing standard shortcuts to enable more expressive keyboard interaction. We expect future work to further understand the performance and learning of hand posture based keyboard shortcuts compared to standard shortcuts.

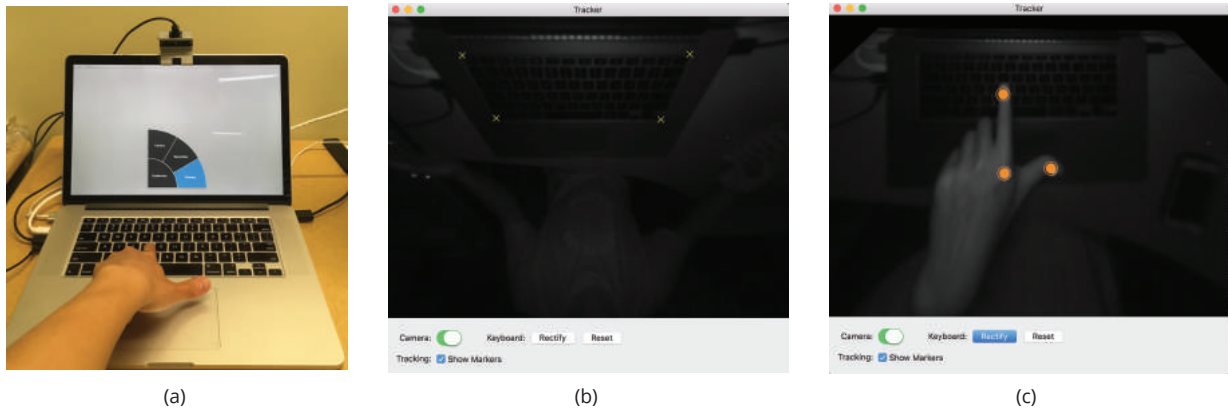


Figure 4.4: Implementation: (a) apparatus and a demonstration of triggering a FingerArc shortcut; (b) keyboard rectification with manual labels; (c) marker tracking for hand posture identification.

4.4 Summary

This chapter has presented a qualitative interview study that provides us with a better understanding of the current issues with conventional keyboard shortcuts. Based on the findings, we presented the concept and implementation of FingerArc and FingerChord, which extends Finger-Aware Shortcuts to be more usable, learnable, and explorable. By providing visual guidance as an intermediate step between graphical input and direct shortcut activation, FingerArc and FingerChord are expected to better facilitate users' transition to expert behaviour. In addition, detecting hand posture based keyboard shortcuts further increases the expressivity of keyboard shortcuts.

Chapter 5

Conclusion

This thesis explored three different finger-aware techniques that enable more enriched interaction experience for activating keyboard shortcuts. Finger-Aware Shortcuts allow for activating different commands with the same key by detecting the finger, hand, and hand posture used for pressing. FingerArc and FingerChord let users press a key with one finger and select from different commands with another finger.

Our evaluations to Finger-Aware Shortcuts revealed the performance and preference patterns for pressing keys with different fingers, hands, and hand postures, as well as their performance compared to conventional keyboard shortcut mapping strategies. We showed that Finger-Aware Shortcuts have similar performance to a common, yet necessary class of keyboard shortcuts consisting of one alphanumeric key and multiple modifier keys. Our interview of expert computer users identified the key factors that impede the learning of keyboard shortcuts. Based on the findings, we designed FingerArc and FingerChord that extend Finger-Aware Shortcuts to detect more expressive hand postures, provide dynamic visual feedback, and allow for cancellation to help novice users explore and transition to experts.

Together, our work unveiled a novel design space on conventional keyboards that encode the information from finger, hand, and hand posture identification, which makes keyboard interaction more expressive. While understanding the long-term performance of these techniques is beyond the scope of this thesis, we hope our work may provide valuable insights for researchers to further explore this interaction space.

References

- [1] Fraser Anderson and Walter F. Bischof. Learning and performance with gesture guides. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1109–1118. ACM, 2013. 5, 13, 14
- [2] Caroline Appert and Shumin Zhai. Using Strokes As Command Shortcuts: Cognitive Benefits and Toolkit Support. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 2289–2298, 2009. 13
- [3] Gilles Bailly, Thomas Pietrzak, Jonathan Deber, and Daniel J. Wigdor. Métamorphe: Augmenting Hotkey Usage with Actuated Keys. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 563–572, 2013. 2, 9
- [4] Olivier Bau and Wendy E. Mackay. OctoPocus: a dynamic guide for learning gesture-based command sets. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 37–46. ACM, 2008. 5, 13, 14
- [5] Michel Beaudouin-Lafon. Instrumental interaction: an interaction model for designing post-WIMP user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 446–453. ACM, 2000. 1, 8
- [6] Ruth Ben'ary. *Touch Typing in Ten Lessons*. Perigee Books, New York, NY, USA, revised edition edition, 1989. 17
- [7] Suresh K. Bhavnani and Bonnie E. John. From Sufficient to Efficient Usage: An Analysis of Strategic Knowledge. In *Proceedings of the ACM SIGCHI*

Conference on Human Factors in Computing Systems, CHI '97, pages 91–98, New York, NY, USA, 1997. ACM. 2, 8, 12

- [8] Suresh K. Bhavnani and Bonnie E. John. The Strategic Use of Complex Computer Systems. *Hum.-Comput. Interact.*, 15(2):107–137, September 2000. 2, 8, 12
- [9] Florian Block, Hans Gellersen, and Nicolas Villar. Touch-display Keyboards: Transforming Keyboards into Interactive Surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1145–1154, 2010. 10
- [10] Sebastian Boring, David Ledo, Xiang 'Anthony' Chen, Nicolai Marquardt, Anthony Tang, and Saul Greenberg. The Fat Thumb: Using the Thumb's Contact Size for Single-handed Mobile Interaction. In *Proceedings of the 14th International Conference on Human-computer Interaction with Mobile Devices and Services*, MobileHCI '12, pages 39–48, New York, NY, USA, 2012. ACM. 11
- [11] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101, January 2006. 44
- [12] John M. Carroll and Mary Beth Rosson. *Paradox of the active user*. The MIT Press, 1987. 2, 8, 12
- [13] Géry Casiez, Nicolas Roussel, and Daniel Vogel. 1 € Filter: A Simple Speed-based Low-pass Filter for Noisy Input in Interactive Systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 2527–2530, 2012. 29
- [14] Andy Cockburn, Carl Gutwin, Joey Scarr, and Sylvain Malacria. Supporting Novice to Expert Transitions in User Interfaces. *ACM Comput. Surv.*, 47(2):31:1–31:36, November 2014. 7, 12
- [15] Andy Cockburn, Per Ola Kristensson, Jason Alexander, and Shumin Zhai. Hard Lessons: Effort-inducing Interfaces Benefit Spatial Learning. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 1571–1580, New York, NY, USA, 2007. ACM. 14
- [16] Ashley Colley and Jonna Häkkinä. Exploring finger specific touch screen interaction for mobile phone user interfaces. In *Proceedings of the 26th*

Australian Computer-Human Interaction Conference on Designing Futures: the Future of Design, pages 539–548. ACM, 2014. 11

- [17] Paul H. Dietz, Benjamin Eidelson, Jonathan Westhues, and Steven Bathiche. A Practical Pressure Sensitive Computer Keyboard. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*, UIST '09, pages 55–58, 2009. 10
- [18] Brian D Ehret. Learning where to look: Location learning in graphical user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 211–218. ACM, 2002. 14
- [19] Bruno Fruchard, Eric Lecolinet, and Olivier Chapuis. MarkPad: Augmenting Touchpads for Command Selection. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 5630–5642. ACM, 2017. 14
- [20] Wai-Tat Fu and Wayne D. Gray. Resolving the paradox of the active user: stable suboptimal performance in interactive tasks. *Cognitive Science*, 28(6):901–935, November 2004. 13
- [21] Patricia I. Fusch and Lawrence R. Ness. Are We There Yet? Data Saturation in Qualitative Research. *The Qualitative Report; Fort Lauderdale*, 20(9):1408–1416, September 2015. 44
- [22] Wilbert O. Galitz. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. Wiley Desktop Editions. John Wiley & Sons, New York, NY, USA, 3, illustrated edition, 2007. 2
- [23] Emmanouil Giannisakis, Gilles Bailly, Sylvain Malacria, and Fanny Chevalier. IconHK: Using Toolbar Button Icons to Communicate Keyboard Shortcuts. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 4715–4726. ACM, 2017. 8, 53
- [24] Hyunjae Gil, DoYoung Lee, Seunggyu Im, and Ian Oakley. TriTap: Identifying Finger Touches on Smartwatches. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 3879–3890, New York, NY, USA, 2017. ACM. 11

- [25] Alix Goguey, Géry Casiez, Thomas Pietrzak, Daniel Vogel, and Nicolas Roussel. Adoiraccourcix: Multi-touch Command Selection Using Finger Identification. In *Proceedings of the 26th Conference on L'Interaction Homme-Machine, IHM '14*, pages 28–37, 2014. 11
- [26] Alix Goguey, Mathieu Nancel, Géry Casiez, and Daniel Vogel. The performance and preference of different fingers and chords for pointing, dragging, and object transformation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '16*, page 12p, 2016. 11
- [27] Alix Goguey, Daniel Vogel, Fanny Chevalier, Thomas Pietrzak, Nicolas Roussel, and Géry Casiez. Leveraging finger identification to integrate multi-touch command selection and parameter manipulation. *International Journal of Human-Computer Studies*, 99:21–36, March 2017. 1, 11
- [28] D. O. Gorodnichy and A. Yogeswaran. Detection and tracking of pianist hands and fingers. In *The 3rd Canadian Conference on Computer and Robot Vision (CRV'06)*, pages 63–63, June 2006. 11
- [29] Wayne D. Gray, Chris R. Sims, Wai-Tat Fu, and Michael J. Schoelles. The soft constraints hypothesis: a rational analysis approach to resource allocation for interactive behavior. *Psychological review*, 113(3):461, 2006. 12
- [30] Tovi Grossman, Pierre Dragicevic, and Ravin Balakrishnan. Strategies for Accelerating On-line Learning of Hotkeys. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '07*, pages 1591–1600, 2007. 1, 2, 8, 9, 14, 52
- [31] Aakar Gupta, Muhammed Anwar, and Ravin Balakrishnan. Porous Interfaces for Small Screen Multitasking Using Finger Identification. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology, UIST '16*, pages 145–156, New York, NY, USA, 2016. ACM. 11
- [32] Aakar Gupta and Ravin Balakrishnan. DualKey: Miniature Screen Text Entry via Finger Identification. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, CHI '16*, pages 59–70, New York, NY, USA, 2016. ACM. 11
- [33] Carl Gutwin, Andy Cockburn, and Benjamin Lafreniere. Testing the Rehearsal Hypothesis with Two FastTap Interfaces. In *Proceedings of the*

41st Graphics Interface Conference, GI '15, pages 223–231, Toronto, Ont., Canada, Canada, 2015. Canadian Information Processing Society. 5, 9, 13

- [34] Carl Gutwin, Andy Cockburn, Joey Scarr, Sylvain Malacria, and Scott C. Olson. Faster Command Selection on Tablets with FastTap. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 2617–2626, New York, NY, USA, 2014. ACM. 5, 13, 14
- [35] Chris Harrison and Scott Hudson. Using Shear As a Supplemental Two-dimensional Input Channel for Rich Touchscreen Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 3149–3152, New York, NY, USA, 2012. ACM. 11
- [36] Chris Harrison, Julia Schwarz, and Scott E. Hudson. TapSense: Enhancing Finger Interaction on Touch Surfaces. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 627–636, New York, NY, USA, 2011. ACM. 11
- [37] Jeff Hendy, Kellogg S. Booth, and Joanna McGrenere. Graphically Enhanced Keyboard Accelerators for GUIs. In *Proceedings of Graphics Interface 2010*, GI '10, pages 3–10, Toronto, Ont., Canada, Canada, 2010. Canadian Information Processing Society. 8
- [38] Seongkook Heo, Jingun Jung, and Geehyuk Lee. MelodicTap: fingering hotkey for touch tablets. In *Proceedings of the 28th Australian Conference on Computer-Human Interaction*, pages 396–400. ACM, 2016. 14
- [39] Da-Yuan Huang, Ming-Chang Tsai, Ying-Chao Tung, Min-Lun Tsai, Yen-Ting Yeh, Liwei Chan, Yi-Ping Hung, and Mike Y. Chen. TouchSense: Expanding Touchscreen Input Vocabulary Using Different Areas of Users' Finger Pads. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 189–192, New York, NY, USA, 2014. ACM. 11
- [40] Robert J. K. Jacob, Audrey Girouard, Leanne M. Hirshfield, Michael S. Horn, Orit Shaer, Erin Treacy Solovey, and Jamie Zigelbaum. Reality-based interaction: A framework for post-wimp interfaces. In *Proceedings*

of the SIGCHI Conference on Human Factors in Computing Systems, CHI '08, pages 201–210, 2008. 2, 8

- [41] John Karat, James E. McDonald, and Matt Anderson. A comparison of menu selection techniques: touch panel, mouse and keyboard. *International Journal of Man-Machine Studies*, 25(1):73–88, July 1986. 1
- [42] John Karat, James E. McDonald, and Matt Anderson. A comparison of menu selection techniques: Touch panel, mouse and keyboard. *International Journal of Man-Machine Studies*, 25(1):73–88, July 1986. 8
- [43] David Kim, Shahram Izadi, Jakub Dostal, Christoph Rhemann, Cem Keskin, Christopher Zach, Jamie Shotton, Timothy Large, Steven Bathiche, Matthias Niessner, D. Alex Butler, Sean Fanello, and Vivek Pradeep. Retrodepth: 3d silhouette sensing for high-precision input on and above physical surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 1377–1386, 2014. 10, 41
- [44] Jong Wook Kim and Frank E. Ritter. Learning, Forgetting, and Relearning for Keystroke- and Mouse-Driven Tasks: Relearning Is Important. *Human-Computer Interaction*, 30(1):1–33, January 2015. 8, 52
- [45] Donald E. Knuth and Andrew Binstock. Interview with donald knuth, Apr 2008. Retrieved Sep 20, 2015 from <http://www.informit.com/articles/article.aspx?p=1193856>. 2, 8
- [46] Brian Krisler and Richard Alterman. Training Towards Mastery: Overcoming the Active User Paradox. In *Proceedings of the 5th Nordic Conference on Human-computer Interaction: Building Bridges*, NordiCHI '08, pages 239–248, 2008. 2, 9, 52
- [47] Per-Ola Kristensson and Shumin Zhai. SHARK2: A Large Vocabulary Shorthand Writing System for Pen-based Computers. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, UIST '04, pages 43–52, New York, NY, USA, 2004. ACM. 13
- [48] Gordon Kurtenbach and William Buxton. User Learning and Performance with Marking Menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, pages 258–264, 1994. 2, 8, 13

- [49] Gordon Kurtenbach, George W. Fitzmaurice, Russell N. Owen, and Thomas Baudel. The Hotbox: Efficient Access to a Large Number of Menu-items. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, pages 231–237, New York, NY, USA, 1999. ACM. 1, 8
- [50] Gordon Kurtenbach, Thomas P. Moran, and William Buxton. Contextual animation of gestural commands. In *Computer Graphics Forum*, volume 13, pages 305–314. Wiley Online Library, 1994. 5, 8, 13, 14
- [51] Gordon P. Kurtenbach, Abigail J. Sellen, and William A. S. Buxton. An Empirical Evaluation of Some Articulatory and Cognitive Aspects of Marking Menus. *Hum.-Comput. Interact.*, 8(1):1–23, March 1993. 8, 13
- [52] Gordon Paul Kurtenbach. *The design and evaluation of marking menus*. PhD thesis, University of Toronto, 1993. 5, 8, 13
- [53] Benjamin Lafreniere, Carl Gutwin, Andy Cockburn, and Tovi Grossman. Faster command selection on touchscreen watches. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 4663–4674. ACM, 2016. 14
- [54] David M. Lane, H. Albert Napier, S. Camille Peres, and Anikó Sándor. Hidden Costs of Graphical User Interfaces: Failure to Make the Transition from Menus and Icon Toolbars to Keyboard Shortcuts. *International Journal of Human-Computer Interaction*, 18(2):133–144, May 2005. 1, 2, 8
- [55] Jinha Lee, Alex Olwal, Hiroshi Ishii, and Cati Boulanger. SpaceTop: Integrating 2d and Spatial 3d Interactions in a See-through Desktop Environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 189–192, 2013. 10
- [56] Sylvain Malacria, Gilles Bailly, Joel Harrison, Andy Cockburn, and Carl Gutwin. Promoting hotkey use through rehearsal with *exposehk*. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 573–582, 2013. 2, 5, 9, 14, 52
- [57] Sylvain Malacria, Joey Scarr, Andy Cockburn, Carl Gutwin, and Tovi Grossman. Skillometers: Reflective Widgets That Motivate and Help Users

to Improve Performance. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 321–330, 2013. 9

- [58] Shahzad Malik and Joe Laszlo. Visual touchpad: A two-handed gestural input device. In *Proceedings of the 6th International Conference on Multimodal Interfaces*, ICMI '04, pages 289–296, 2004. 9, 10
- [59] Damien Masson, Alix Goguey, Sylvain Malacria, and Géry Casiez. Whichfingers: Identifying Fingers on Touch Surfaces and Keyboards using Vibration Sensors. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, 2017. 11
- [60] Hugh McLoone, Ken Hinckley, and Edward Cutrell. Bimanual Interaction on the Microsoft Office Keyboard. In *Proceedings of IFIP TC 13 International Conference on Human-Computer Interaction*, INTERACT '03, pages 49–56, 2003. 1, 2, 8
- [61] Andrew P. McPherson, Adrian Gierakowski, and Adam M. Stark. The Space Between the Notes: Adding Expressive Pitch Control to the Piano Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 2195–2204, New York, NY, USA, 2013. ACM. 11
- [62] Craig S. Miller, Svetlin Denkov, and Richard C. Omanson. Categorization costs for hierarchical keyboard commands. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2765–2768, 2011. 2
- [63] Thomas A. Mysliwicz. FingerMouse: A Freehand Computer Pointing Interface. In *Proceedings of IEEE International Conference on Automatic Face and Gesture Recognition*, FG '95, pages 372–377, 1994. 3, 10
- [64] Jakob Nielsen. Finding usability problems through heuristic evaluation. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 373–380. ACM, 1992. 2, 12
- [65] Jakob Nielsen and Jonathan Levy. Measuring Usability: Preference vs. Performance. *Commun. ACM*, 37(4):66–75, April 1994. 2, 8

- [66] Erik Nilsen, HeeSen Jong, Judith S. Olson, Kevin Biolsi, Henry Rueter, and Sharon Mutter. The Growth of Software Skill: A Longitudinal Look at Learning & Performance. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 149–156, New York, NY, USA, 1993. ACM. 12
- [67] Don Norman. *The design of everyday things: Revised and expanded edition*. Basic Books (AZ), 2013. 7, 44
- [68] Daniel L. Odell, Richard C. Davis, Andrew Smith, and Paul K. Wright. Toolglasses, marking menus, and hotkeys: a comparison of one and two-handed command selection techniques. In *Proceedings of Graphics Interface 2004*, pages 17–24. Canadian Human-Computer Communications Society, 2004. 1, 8
- [69] A. Oka and M. Hashimoto. Marker-less piano fingering recognition using sequential depth images. In *The 19th Korea-Japan Joint Workshop on Frontiers of Computer Vision*, pages 1–4, January 2013. 11
- [70] Michael Ortega and Laurence Nigay. Airmouse: Finger gesture for 2d and 3d interaction. In *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction*, INTERACT '09, pages 214–227, 2009. 3, 10
- [71] S. Camille Peres, Michael D. Fleetwood, Minmin Yang, Franklin P. Tamborello, and Danielle Paige Smith. Pros, cons, and changing behavior: An application in the use of the keyboard to issue commands. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 49 of *HFES '05*, pages 637–641, 2005. 1, 2, 8, 9, 52
- [72] Deniese Pierotti. Heuristic evaluation-a system checklist. *Xerox Corporation*, 1995. 12
- [73] Thomas Pietrzak, Sylvain Malacria, and Gilles Bailly. CtrlMouse et TouchCtrl: Duplicating Mode Delimiters on the Mouse. In *Proceedings of the 26th Conference on L'Interaction Homme-Machine*, IHM '14, pages 38–47, 2014. 2, 9, 53

- [74] Jun Rekimoto, Takaaki Ishizawa, Carsten Schwesig, and Haruo Oba. Pre-Sense: Interaction Techniques for Finger Sensing Input Devices. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*, UIST '03, pages 203–212, 2003. 10
- [75] Roger W. Remington, Ho Wang Holman Yuen, and Harold Pashler. With Practice, Keyboard Shortcuts Become Faster than Menu Selection: A Crossover Interaction. *Journal of Experimental Psychology: Applied*, 22(1):95, 2016. 8
- [76] Mary Beth Rosson. Patterns of Experience in Text Editing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '83, pages 171–175, New York, NY, USA, 1983. ACM. 12
- [77] Joey Scarr, Andy Cockburn, Carl Gutwin, and Andrea Bunt. Improving Command Selection with CommandMaps. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 257–266. ACM, 2012. 14
- [78] Joey Scarr, Andy Cockburn, Carl Gutwin, Andrea Bunt, and Jared E. Cechanowicz. The Usability of CommandMaps in Realistic Tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2241–2250. ACM, 2014. 14
- [79] Joey Scarr, Andy Cockburn, Carl Gutwin, and Sylvain Malacria. Testing the Robustness and Performance of Spatially Consistent Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3139–3148. ACM, 2013. 14
- [80] Joey Scarr, Andy Cockburn, Carl Gutwin, and Philip Quinn. Dips and Ceilings: Understanding and Supporting Transitions to Expertise in User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2741–2750, New York, NY, USA, 2011. ACM. 13
- [81] Richard A. Schmidt. Frequent Augmented Feedback Can Degrade Learning: Evidence and Interpretations. In *Tutorials in Motor Neuroscience*, NATO ASI Series, pages 59–75. Springer, Dordrecht, 1991. DOI: 10.1007/978-94-011-3626-6_6. 13

- [82] Richard A. Schmidt, Douglas E. Young, Stephan Swinnen, and Diane C. Shapiro. Summary knowledge of results for skill acquisition: Support for the guidance hypothesis. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 15(2):352–359, 1989. 14
- [83] Katherine Schramm, Carl Gutwin, and Andy Cockburn. Supporting Transitions to Expertise in Hidden Toolbars. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 4687–4698. ACM, 2016. 14
- [84] Barry Schwartz, Andrew Ward, John Monterosso, Sonja Lyubomirsky, Katherine White, and Darrin R. Lehman. Maximizing versus satisficing: happiness is a matter of choice. *Journal of personality and social psychology*, 83(5):1178, 2002. 12
- [85] Atsushi Sugiura and Yoshiyuki Koseki. A user interface using fingerprint recognition: Holding commands and data objects on fingers. In *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*, UIST '98, pages 71–79, 1998. 11
- [86] Hemant Bhaskar Surale, Fabrice Matulic, and Daniel Vogel. Experimental Analysis of Mode Switching Techniques in Touch-based User Interfaces. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 3267–3280, New York, NY, USA, 2017. ACM. 11
- [87] Susanne Tak, Piet Westendorp, and Iris van Rooij. Satisficing and the Use of Keyboard Shortcuts: Being Good Enough Is Enough? *Interacting with Computers*, 25(5):404–416, September 2013. 1, 2, 8, 9, 12
- [88] Stuart Taylor, Cem Keskin, Otmar Hilliges, Shahram Izadi, and John Helmes. Type-hover-swipe in 96 Bytes: A Motion Sensing Mechanical Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 1695–1704, 2014. 3, 10
- [89] Md Sami Uddin and Carl Gutwin. Rapid Command Selection on Multi-Touch Tablets with Single-Handed HandMark Menus. In *Proceedings of the 2016 ACM on Interactive Surfaces and Spaces*, pages 205–214. ACM, 2016. 14

- [90] Md. Sami Uddin, Carl Gutwin, and Benjamin Lafreniere. HandMark Menus: Rapid Command Selection and Large Command Sets on Multi-Touch Displays. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 5836–5848, New York, NY, USA, 2016. ACM. 5, 14
- [91] Feng Wang, Xiang Cao, Xiangshi Ren, and Pourang Irani. Detecting and Leveraging Finger Orientation for Interaction with Direct-touch Surfaces. In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '09, pages 23–32, New York, NY, USA, 2009. ACM. 11
- [92] Feng Wang and Xiangshi Ren. Empirical Evaluation for Finger Input Properties in Multi-touch Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1063–1072, New York, NY, USA, 2009. ACM. 11
- [93] Jingtao Wang and John Canny. Fingersense: Augmenting expressiveness to physical pushing button by fingertip identification. In *CHI '04 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '04, pages 1267–1270, 2004. 11
- [94] Malte Weiss, Julie Wagner, Yvonne Jansen, Roger Jennings, Ramsin Khoshabeh, James D. Hollan, and Jan Borchers. Slap widgets: Bridging the gap between virtual and physical controls on tabletops. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 481–490, 2009. 10
- [95] Andrew D. Wilson. Robust Computer Vision-based Detection of Pinching for One and Two-handed Gesture Input. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, UIST '06, pages 255–258, 2006. 3, 10
- [96] Andrew D. Wilson and Edward Cutrell. FlowMouse: A Computer Vision-Based Pointing and Gesture Input Device. In *Proceedings of IFIP TC 13 International Conference on Human-Computer Interaction*, INTERACT '05, pages 565–578, 2005. 3, 10
- [97] Jacob O. Wobbrock, Leah Findlater, Darren Gergle, and James J. Higgins. The Aligned Rank Transform for Nonparametric Factorial Analyses

Using Only Anova Procedures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 143–146, 2011. 22

- [98] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. Gestures without libraries, toolkits or training: A \$1 recognizer for user interface prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, pages 159–168, 2007. 29
- [99] Xin Yi, Chun Yu, Mingrui Zhang, Sida Gao, Ke Sun, and Yuanchun Shi. ATK: Enabling Ten-Finger Freehand Typing in Air Based on 3d Hand Tracking Data. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, pages 539–548, New York, NY, USA, 2015. ACM. 10
- [100] Erdem Yoruk, Ender Konukoglu, Bulent Sankur, and Jérôme Darbon. Shape-based hand recognition. *Image Processing, IEEE Transactions on*, 15(7):1803–1815, July 2006. 28
- [101] Shumin Zhai and Per-Ola Kristensson. Shorthand Writing on Stylus Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, pages 97–104, New York, NY, USA, 2003. ACM. 5, 13
- [102] Shumin Zhai, Per Ola Kristensson, Pengjun Gong, Michael Greiner, Shilei Allen Peng, Liang Mico Liu, and Anthony Dunnigan. Shapewriter on the Iphone: From the Laboratory to the Real World. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '09, pages 2667–2670, 2009. 13
- [103] Haimo Zhang and Yang Li. GestKeyboard: Enabling Gesture-based Interaction on Ordinary Physical Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 1675–1684, 2014. 10