Investigating the Design and Development of Multitouch Applications

Kenrick Kin

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-233
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-233.html

December 11, 2012

Copyright © 2012, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Investigating the Design and Development of Multitouch Applications

by

Kenrick Chen-Kuo Kin

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Maneesh Agrawala, Chair
Björn Hartmann
David Wessel

Fall 2012

Investigating the Design and Development of Multitouch Applications

Copyright 2012
by
Kenrick Chen-Kuo Kin

Abstract

Investigating the Design and Development of Multitouch Applications

by

Kenrick Chen-Kuo Kin

Doctor of Philosophy in Computer Science

University of California, Berkeley

Maneesh Agrawala, Chair

Multitouch is a ubiquitous input technique, used primarily in mobile devices such as phones and tablets. Larger multitouch displays have been mostly limited to tabletop research projects, but hardware manufacturers are also integrating multitouch into desktop workstations. Multitouch input has several key differences from mouse and keyboard input that make it a promising input technique. While the mouse is an indirect and primarily unimanual input device, multitouch often supports direct-touch input and encourages bimanual interaction. Multitouch also supports the use of all ten fingers as input, providing many more degrees of freedom of input than the 2D mouse cursor.

Building multitouch applications first requires understanding these differences. We present a pair of user studies that contribute to the understanding of the benefits of direct-touch and bimanual input afforded by multitouch input. We then discuss how we leverage these benefits to create multitouch gestures for a professional content-creation application run on a large multitouch display. The differences between multitouch and mouse input also greatly affect the needs of an application developer. We lastly present a declarative multitouch framework that helps developers build and manage gestures that require the coordination of multiple fingers.

In our first study, users select multiple targets with a mouse and with multitouch using one finger, two fingers (one from each hand), and any number of fingers. We find that the fastest multitouch interaction is about twice as fast as the mouse for selection. The direct-touch nature of multitouch accounts for 83% of the reduction in selection time. Bimanual interaction, using at least one finger on each hand, accounts for the remaining reduction. To further investigate bimanual interaction for making directional motions, we examine two-handed marking menus, bimanual techniques in which users make directional strokes to select menu items. We find that bimanually coordinating directional strokes is more difficult than making single strokes. But, with training, making strokes bimanually outperforms making strokes serially by 10-15%.

Our user studies demonstrate that users benefit from multitouch input. However, little work has been done to determine how to design multitouch applications that leverage these

benefits for professional content-creation tasks. We investigate using multitouch input for a professional-level task at Pixar Animation Studios. We work with a professional set construction artist to design and develop Eden, a multitouch application for building virtual organic sets for computer-animated films. The experience of the artist suggests that Eden outperforms Maya, a mouse and keyboard system currently used by set construction artists. We present a set of design guidelines that enabled us to create a gesture set that is both easy for the artist to remember and easy for the artist to perform.

Eden demonstrates the viability of multitouch applications for improving real user workflows. However, multitouch applications are challenging to implement. Despite the differences between multitouch and mouse input, current multitouch frameworks follow the event-handling pattern of mouse-based frameworks. Tracking a single mouse cursor is relatively straightforward as mouse events are broken sequentially into the order in which they must occur: down, move, and up. For multitouch however, developers must meticulously track the proper sequence of touch events from multiple temporally overlapping touch streams using disparate event-handling callbacks. In addition, managing gesture sets can be tedious, as multiple gestures often begin with the same touch event sequence leading to gesture conflicts in which the user input is ambiguous. Thus, developers must perform extensive runtime testing to detect conflicts and then resolve them.

We simplify multitouch gesture creation and management with Proton, a framework that allows developers to declaratively specify a gesture as a regular expression of customizable touch event symbols. Proton provides automatic gesture matching and the static analysis of gesture conflicts. We also introduce gesture tablature, a graphical gesture notation that concisely describes the sequencing of multiple interleaved touch events over time. We demonstrate the expressiveness of Proton with four proof-of-concept applications. Finally, we present a user study that indicates that users can read and interpret gesture tablature over four times faster than event-handling pseudocode.

Multitouch applications require new design principles and tools for development. This dissertation addresses the challenges of designing gestures and interfaces that benefit from multiple parallel touch input and presents tools to help developers build and recognize these new multitouch gestures. This work serves to facilitate a wider adoption of multitouch interfaces. We conclude with several research directions for continuing the investigation of multitouch input.

To my parents, June and C.C. Kin, for their everlasting support.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost I would like to thank my advisor, Maneesh Agrawala, whose support and guidance made this dissertation possible.

I would like to thank my fellow Soda denizens and BiD compatriots who made my stay at Berkeley both intellectually fulfilling and memorable. In particular I would like to thank Anuj Tewari, Floraine Berthouzoz, Lora Oehlberg, Nicholas Kong, Nuttapong Chentanez, Robert Carroll, Robin Held, and Wesley Willett.

In addition to my advisor, I would like to thank several Berkeley faculty members who have influenced my research ideas. Above all, I would like to thank Björn Hartmann who has played an integral role in defining my research path. Carlo Séquin, David Wessel, James O'Brien, John Canny, and Jonathan Shewchuk have inspired me with their passion and work in Computer Graphics and Human-Computer Interaction.

Working at Pixar Animation Studios for five years was a dream and a privilege. I would like to thank Tony DeRose for giving me the opportunity to spearhead the multitouch effort. In addition, I would like to thank the brilliant research interns Björn Bollensdorff, Craig Schroeder, Dominik Käser, Justin Solomon, and Ralph Wiedemeier for their suggestions and contributions to the multitouch project. And of course, I would like to thank Tom Miller, who took the leap of faith in actually using multitouch input to build virtual sets for computer-animated films.

Lastly, I would like to thank my friends and family who encouraged me to go outside and breathe fresh air every now and then. I would like to thank my sister Karin for catching typos and my brother Kevin for providing me with dining hall food.

Chapter 1

Introduction

The mouse was introduced in 1963 [32] and in conjunction with the keyboard, mouse and keyboard interfaces are still the dominant means through which humans interact with computers today. Although the utility of the mouse is undeniable, as it is has long been the primary input device to graphical user interfaces, users are limited by the single mouse cursor through which they manipulate digital content. To increase the functionality of the mouse, users often rely on mode switching to change how the mouse input will affect the application. In addition, a single mouse cursor provides just two spatial degrees of freedom, so tasks that requires manipulating more than two spatial parameters must be broken up into multiple steps. Furthermore, with a single cursor, users must make long traversals between spatially distant elements.

Multitouch interaction is a newly popularized form of interaction that frees users from the constraint of a single point of input. Multitouch allows users to apply any number of two-dimensional touches directly as input to a device with a co-located display. Although multitouch is old in its conception, dating as far back as 1982 [88], only in the last decade have large multitouch displays become commercially available. These include capacitance-based displays [30, 89, 114], infrared camera-based displays [47, 91, 108], and optical-based displays that do not rely on cameras [90, 94]. These devices were largely designed with multi-user interaction in mind. Large displays that detect simultaneous input are especially useful for multiple users to collaborate on the same digital workspace [96, 119, 142].

However, widespread adoption of multitouch did not occur until its use in mobile devices for individual users. Modern mobile phones and tablets [6, 44, 93] have adopted multitouch input as their primary input technique. Removing physical buttons allows more space for a larger display on a mobile device. The co-located display and touch surface supports direct-touch input, which removes the need for an external pointing device. The added degrees of freedom of multiple touches increase the gesture space to support gestures such as the familiar two-touch pinch-to-zoom. Hardware manufacturers are also integrating multitouch into desktop workstations [2, 108]. Multitouch displays for laptops and desktops are larger than those of mobile devices, providing users with more screen space to more easily apply touches with both hands.

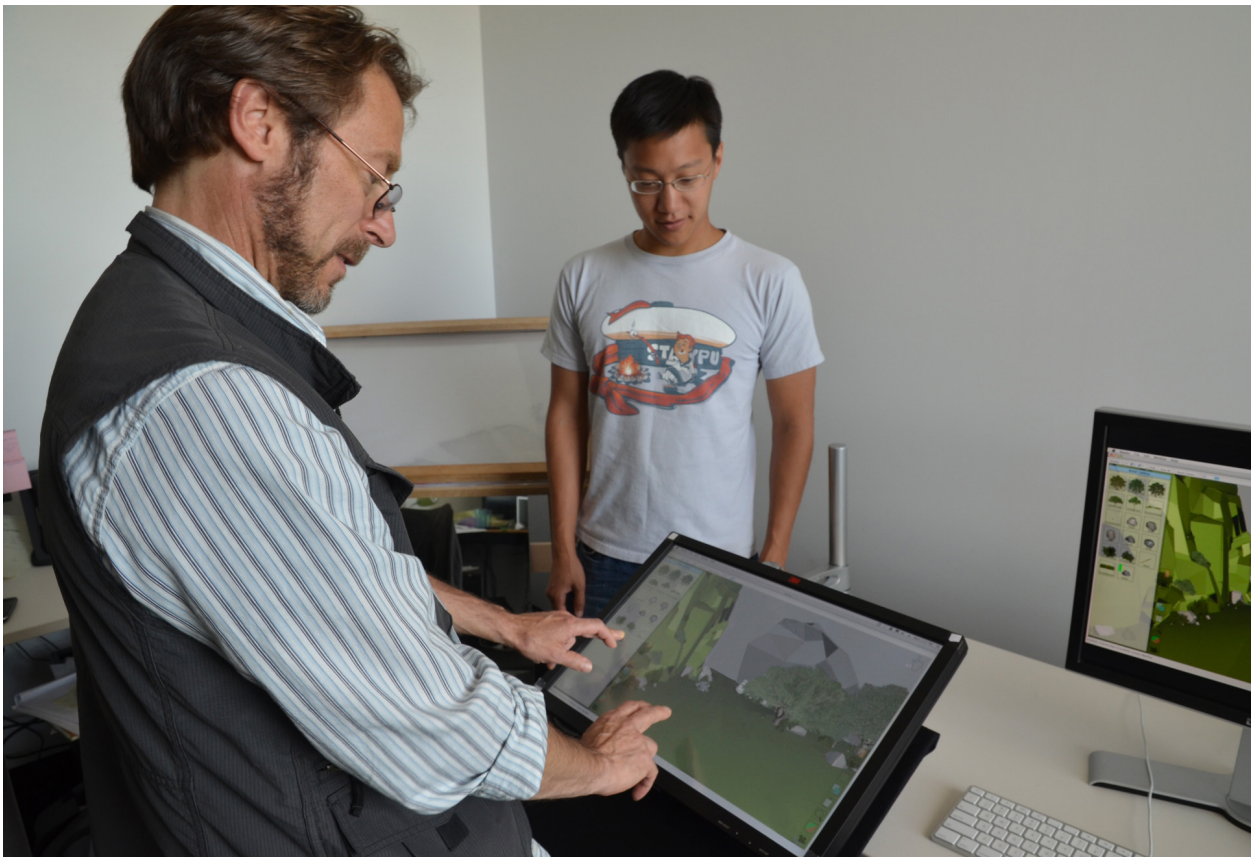


Figure 1.1: A professional set construction artist uses a multitouch interface to build a virtual set for a computer-animated film.

Multitouch input on a desktop workstation is a promising enhancement or alternative to the traditional mouse and keyboard input. Consider the goal of creating a computer-animated film. A common task is the positioning of 3D virtual objects, which is currently performed using a mouse and keyboard interface such as Autodesk Maya [7]. Although advancements in graphics processing have made the rendering of 3D virtual scenes more efficient in such tools, artists are still constrained by the two-dimensional spatial input of the mouse when manipulating 3D content. For artists working on computer-animated films, multitouch can increase their productivity. An artist can benefit from the efficiency of selecting objects with direct-touch. Working bimanually, the artist can efficiently perform operations in parallel or interact with different portions of the screen simultaneously (Figure 1.1). Using the many spatial degrees of freedom of input provided by multiple touch points, the artist can specify and control many parameters of an operation simultaneously, reducing the need to break an operation into multiple steps.

Ultimately, the promise of multitouch input are applications that increase user performance for complex tasks. Thus, it is important to build on the understanding of multitouch

input, to investigate the design of multitouch gesture sets and applications, and to create new tools that facilitate the development of multitouch applications.

1.1 Evaluating Direct-Touch and Bimanual Input

Multitouch input has several key differences from traditional mouse and keyboard input. Multitouch is often a direct-touch input, which allows the user to target a graphical object by touching it on the display. The mouse on the other hand is an indirect pointing device, so the user must position and track a digital cursor to target an object. Sears and Shneiderman [120] have shown that direct-touch is faster than mouse input for target selection. In addition, multitouch input allows the user to use up to all 10 fingers on both hands for targeting, providing 20 degrees of spatial freedom, while the mouse controls only a single cursor with two degrees of freedom.

Since touches can come from either hand, multitouch devices also support bimanual interaction. Although mouse and keyboard devices also support bimanual interaction (e.g., typing or combining keyboard shortcuts with mouse manipulation), multitouch interaction does not restrict one hand to discrete button pressing; both hands can directly interact with the graphical objects on the display. The actions of two hands often overlap in time, and the resulting parallelism can increase performance for tasks that require continuous spatial manipulation of graphical objects [11, 22, 25]. Moreover, each hand can remain in the proximity of the area of work it is responsible for, thereby reducing large-scale movements that may be required in unimanual interactions [31].

The flexibility of being able to use any finger and combinations of fingers as input greatly increases the available degrees of freedom of input over mouse and keyboard interfaces. Certain multitouch hardware [47, 91, 114, 115, 133] even increases the space of available touch parameters, adding attributes such as shape and pressure to the touch position provided by default. The multitude of touch attributes increases the space of possible gestures that can be performed by the user.

We build on research investigating direct-touch, bimanual, and multifinger input with two user studies. We first investigate the performance of multitouch for target selection, a simple and common task in graphical user interfaces. In Chapter 3 we present a multiselection study [69] in which users are shown randomly placed targets on a large multitouch display and then asked to select multiple targets with a mouse, one finger, two fingers (one from each hand), and any number of fingers. We find that the fastest multitouch condition is about twice as fast as the mouse-based workstation, independent of the number of targets. Direct-touch with one finger accounts for an average of 83% of the reduction in selection time. Bimanual interaction, using at least two fingers, one on each hand, accounts for the remaining reduction in selection time. We find no significant difference between using any number of fingers and using one finger from each hand for selection time.

Selecting targets using a finger from each hand is a bimanual interaction. We next consider bimanual interaction with additional spatial movements by examining the performance

of two-handed marking menus [70] in Chapter 4. In a one-handed marking menu, the user makes a sequence of directional strokes to select a menu item. Our two-handed marking menus are bimanual techniques in which the user makes pairs of directional strokes, one stroke with each hand, to select menu items. In one variant, the user draws both strokes simultaneously. In the second variant, the user alternates the hand that draws the strokes. We find that bimanually coordinating directional strokes is more difficult than making single strokes. But, with training, making strokes bimanually outperforms making strokes serially by 10-15%. We present several use cases of two-handed marking menus.

1.2 Designing a Multitouch Application

The performance benefits of multitouch input as demonstrated by our user studies make multitouch applications a compelling enhancement to desktop workstations. Researchers, however, have done little to determine whether multitouch is suited for professional-level tasks. Professional digital content creators still use mouse and keyboard workstations and thus have to perform many sequences of steps to achieve a desired goal. These tasks are often repetitive. If multitouch can improve even a small part of the workflow, there can be significant overall gains in production.

Due to the differences between multitouch and mouse and keyboard input, designing multitouch applications requires different considerations than mouse and keyboard applications. Designing gestures for multitouch is more challenging than for the mouse because multitouch gestures require the user to coordinate the actions of two hands and individual fingers instead of a single mouse cursor. The developer must properly map each potentially complex gesture to an operation such that it is easy to perform and learn. Furthermore, each gesture must be well integrated with and distinguishable from a number of other gestures that make up the gesture set of the application. In Chapter 5, we present a set of design principles and lessons learned from designing the gesture set of a multitouch tool that contributes to the creation of computer-animated films at Pixar Animation Studios. In particular, we developed Eden [73], a multitouch application for building virtual organic sets. Set construction is an integral step of the computer animation pipeline, in which artists arrange and place virtual 3D objects to create environments for computer-animated films.

Working with a professional set construction artist, we leverage direct-touch, bimanual, and multifinger interaction afforded by multitouch input to design a set of gestures tailored for organic set construction. In particular, we map multitouch gestures to 3D object manipulation and camera control operations. We evaluate the effectiveness of multitouch input by comparing Eden to Maya [7], a mouse and keyboard interface currently used by set construction artists today. The positive experiences of two set construction artists suggest that Eden outperforms Maya and multitouch can improve a professional workflow.

1.3 Developing Multitouch Applications

When building Eden we found that multitouch applications are challenging to implement with current frameworks. Much like frameworks for mouse-based applications, current multitouch frameworks are event-based and require developers to detect gestures by handling touch events. Multitouch application developers implement *touch-down*, *touch-move*, and *touch-up* event handlers, which are analogous to *mouse-down*, *mouse-move*, and *mouse-up* event handlers. While there is a logical order of events from mouse-down to mouse-move to mouse-up for a single mouse cursor, multitouch application developers must manage touch events generated from many fingers pressing and lifting up and making temporally overlapping movements on the multitouch device. Developers must then meticulously manage state across callbacks in order to detect the proper interleaving of touch events generated from different fingers. In addition, multitouch gestures often begin with the same touch event sequence, which leads to gesture conflicts in which multiple gestures can match the same user input. Developers must perform extensive runtime testing to identify when multiple gestures conflict and resolve them such that when a user performs a gesture, the application executes the expected command.

In Chapter 6 we present Proton [71, 72], a multitouch framework designed to improve multitouch gesture creation and management. Proton allows developers to declaratively specify a multitouch gesture as a regular expression describing a sequence of touch event symbols. Developers can extend these touch event symbols by adding custom touch attributes to the symbols, such as touch hit-target or touch direction, thereby increasing the expressiveness of multitouch gesture regular expressions. Given these expressions, Proton automatically detects when user input matches the gesture descriptions and also statically analyzes the gestures to detect gestures conflicts. In addition, we introduce gesture tablature, a graphical notation for specifying gestures.

We demonstrate the flexibility of Proton with several example attributes and their applications. We validate Proton with a series of example applications: a 2D set construction application, a sketching application, a single-stroke character entry technique [140], and a two-player Pong game. We also conduct a user study that indicates that users can read and interpret gesture tablature 2.0-2.1 times faster than gesture regular expressions and 4.2-4.7 times faster than event-handling pseudocode.

1.4 Wider Adoption of Multitouch

Multitouch technology has become widely adopted for mobile use and larger multitouch displays have become increasingly available. Our user studies contribute to the overall understanding of multitouch input, helping to inform the design of applications for these devices. Eden, our multitouch set construction application, suggests that there is still room for the workstation of professional content creators to evolve and improve. The design guidelines we present provide a starting point for the development of new multitouch interfaces for these

users. Proton, our novel declarative multitouch framework, supports the developer in the creation and management of large gesture sets. By facilitating the development of multitouch applications and introducing good guidelines for building gestures, our work serves to propel the adoption and effectiveness of multitouch interfaces.

This dissertation is partially based on papers previously published in CHCCS/SCDHM and ACM conference proceedings and journals; I am the primary author on all publications. The multitarget selection study in Chapter 3 was published in GI 2009 [69]; the two-handed marking menu techniques in Chapter 4 were published in TOCHI 2011 [70]; Eden in Chapter 5 was published in CHI 2011 [73], and Proton in Chapter 6 was published in CHI 2012 [72] and UIST 2012 [71].

Chapter 2

Related Work

Many areas of multitouch research have contributed to the creation of multitouch applications. We first cover prior work that examines the direct-touch and bimanual properties of multitouch input that distinguish multitouch input from mouse and keyboard input. In addition, we discuss multitouch research that use the many degrees of freedom of input to perform complex tasks like three-dimensional object manipulation, a task commonly performed when making computer-animated films. Some multitouch hardware detect various touch attributes in addition to touch position. We survey how researchers have used these additional degrees of freedom to create new multitouch gestures. We then discuss research that explores the design of multitouch applications and large gesture sets. Finally, we summarize the different types of multitouch frameworks available for developing multitouch applications.

2.1 Direct-Touch Input

Multitouch hardware with a co-located display supports direct-touch input. Early touchscreens also supported direct-touch, but for a single point of contact. Studies comparing touchscreen interactions to mouse-based interactions in a selection task found speed advantages for the touchscreen [62, 105]. However, occlusion due to the fingers and imprecision in locating touchpoints also reduced accuracy of the direct-touch input. Sears and Shneiderman [120] investigated the effect of target size on speed and accuracy. They found that targets of size 0.64 cm in width or larger were faster to select with a touchscreen than a mouse, while error rates were roughly equal in both conditions. At a target size of 1.28 cm users made 66% fewer errors with the touchscreen than with the mouse.

2.2 Bimanual Interaction

Guiard, in his seminal work on the Kinematic Chain Model [45, 46], assigned different, *asymmetric* roles to the hands. The non-dominant hand sets the frame of reference in which the dominant hand executes the primary action. Empirical studies have shown the promise

of two hands working together when they are assigned asymmetric roles [12, 22, 53, 59]. Other studies have also shown the potential for increased performance when assigning both hands similar, *symmetric* roles [11, 25, 80, 81]. In addition, researchers have shown that for target selection tasks there is significant overlap, or *parallelism* in movement between the hands [29, 66, 67]. These studies have informed the design of a large variety of bimanual interaction techniques [14, 17, 51, 52, 106, 142].

2.2.1 Direct-Touch & Bimanual Target Selection

With the advent of multitouch workstations that offer both direct-touch and bimanual interaction, a few groups have begun studying these two factors in combination. Barnert [13] designed an asymmetric task, where users matched a template cube by specifying the base of the cube with one hand and the height of the cube with the other hand. Barnert found that users are better at target selection and dragging using either one or two mice than using one or two hands on a multitouch table. However, Barnert's task required single pixel-accuracy which Sears and Shneiderman [120] showed to be very difficult for direct-touch devices. It is unlikely that Barnert's results would apply to selection tasks with larger targets.

Forlines et al. [37] conducted a pair of experiments that compared target selection on a mouse workstation and a multitouch workstation. In the first study they focused on unimanual interaction and showed that in a single-target selection task with targets of size 1.92 cm and larger, direct-touch offers modest speed advantages over the mouse. However, the direct-touch device also produced twice the error rate of the mouse and therefore Forlines et al. recommended using a mouse for single-target selection. In the second experiment they studied a symmetric bimanual selection and docking rectangle resizing task. Users first selected target handles at diagonally opposite corners of a rectangle using both hands and then dragged those handles onto the corners of a differently sized target rectangle. In this task they found direct bimanual multitouch to take 0.64 times as long as using a pair of mice. However error rates for the target selection part of the task were twice as large in the multitouch condition. Nevertheless Forlines et al. recommend using direct multitouch interaction for such symmetric bimanual tasks because of the speed advantages.

Because Forlines et al. did not consider unimanual and bimanual conditions together in the same task, their studies could not directly compare the performance advantages due to the multiple input capabilities of a multitouch display versus those only due to direct-touch. Our multitarget selection study in Chapter 3 fills this gap by considering both interaction dimensions—indirect/direct and unimanual/bimanual—together. We also consider the performance of multifinger input.

2.3 Many Degrees of Freedom of Input

The limited degrees of freedom of the mouse have motivated the development of input devices with more degrees of freedom, such as the *Data Glove* [150], the *Bat* [131], *GlobeFish*

and *GlobeMouse* [40], and the commercially available *SpaceNavigator* [1]. Multitouch workstations also provide many input degrees of freedom that can be leveraged for specifying an application's operations and their many parameters.

A common task that requires the manipulation of many degrees of freedom is 3D object manipulation. Hancock et. al. [48], Reisman et al. [113], and Martinet et. al. [84] investigated using multitouch for 3D object manipulation. They used the number and locations of the touches to set the object manipulation operation (translation, rotation, and scale) and mapped the motion of the touches to the corresponding parameters. Cohé et al. designed a multitouch widget [26], with which the user specifies the object manipulation operation by directly touching specific targets on the widget. Cardinaels et al. [24] also designed multitouch gestures for manipulating objects in an application for conceptualizing scenes for television productions. In Chapter 5 we also leverage direct-touch and multifinger input to design object manipulation gestures for Eden, an application designed for producing virtual environments for computer-animated films.

2.4 Touch Attribute Utilization

Modern multitouch hardware support gestures that depend on the touch positions provided by the hardware. In addition to touch position, certain multitouch hardware can detect other attributes such as touch shape and touch identity. Researchers have used these additional touch attributes to create more diverse gestures. We survey representative techniques that utilize common hardware supported touch attributes, including touch positions for creating trajectory-dependent gestures (Section 2.4.1), touch shape for increasing the design space of gestures (Section 2.4.2), and touch identity for tracking track hands and users in multi-user applications (Section 2.4.3).

2.4.1 Trajectory

Trajectory recognition systems consider touch positions over time. Researchers have developed trajectory recognitions systems that rely on comparisons to demonstrations [116, 139], and on regular expression matching to a string representation of a gesture [141]. These recognizers can only classify trajectory at the end of a gesture and thus cannot provide recognition feedback as the user performs the gesture. In contrast, some systems detect trajectory *online*, as the user performs the gesture, and thus are able to provide continuous feedback. Bevilacqua et al. [15] use a hidden Markov model to perform gesture following and Swigart [125] detects trajectory as a sequence of directions formed by the last two positions of the touch. Our multitouch framework takes a similar approach to Swigart by allowing developers to incorporate custom direction attributes into touch event symbols for trajectory matching. Researchers have used touch trajectories to disambiguate widget selection [97], implement multitouch marking menus [70, 82], and detect stroke commands [5].

2.4.2 Touch Shape

All multitouch devices detect the positions of touches, but many devices also detect touch shape [47, 91, 114, 115, 133]. Researchers have used hand shape in multitouch applications to distinguish between different operations based on analogies to real-world manipulations [23], to control physics simulations [137], to constrain degrees of freedom for shape manipulation [135], and to distinguish commands in multi-user applications [142]. From touch shape, researchers have extracted touch area, which they have used to simulate applied pressure when selecting and manipulating objects [14, 23]. Pressure-sensitive widgets designed for stylus input can also be implemented using touch area [111]. Researchers have extracted touch orientation from touch shape by fitting an ellipse to the shape and calculating the angle of its major axis [27, 129]; they use orientation as an additional parameter for object manipulation and command selection. Using Proton, developers can integrate touch area and touch orientation into gesture expressions to detect and execute similar interaction techniques, which we describe in Chapter 6.

2.4.3 Hand and User Identification

Most multitouch devices cannot distinguish the sources of touches, such as from which finger, hand, or person the touch originated. Researchers have augmented multitouch systems with additional cameras to track the identity of hands [33]. With hand identity, developers can assign different gestures and roles to each hand, as promoted by Guiard's Kinematic Chain Model [45]. The DiamondTouch [89] table identifies users through capacitive coupling between the touch surface and a pad on each user's chair. User identity has enabled researchers to develop cooperative gestures for multiple users [96] and player tracking for multiplayer games [35]. Proton can integrate hand and user identity into touch events.

2.5 Multitouch Applications

In addition to investigating the benefits of multitouch input and mapping touch attributes to individual operations, researchers have also explored the design of complete multitouch applications. However, many of these applications primarily served as testbeds for multitouch interaction design and were not designed for deployment to real users. For example, Wu et al. [142, 143] developed a room planning application to investigate multi-user interaction and an annotation application to investigate gesture design. Brandl et al. [20] developed a sketching application to investigate touch and pen interaction. Researchers also deployed applications designed for casual users outside of a lab setting, including a card game application for a senior citizens center [41] and a large media display in a public city center [107]. They explored the social interactions between users due to these applications.

Few researchers have explored the use of multitouch for producing work in a professional environment. One notable exception is the research by Wigdor et al. [134], which investigated the long term use of a multitouch workstation for office-related tasks, such as replying to

e-mail. However, the authors used multitouch as a mouse emulation device for pre-existing mouse and keyboard interfaces. They did not redesign the application to better leverage multitouch input. In contrast, we designed Eden (Chapter 5) from the ground up to leverage multitouch input for a professional task.

Researchers have also recently examined user-defined gestures. Wobbrock et al. [138] combined gestures designed by 20 end-users to create a gesture set for 22 commonly used commands. Follow-up work by Morris et al. [95] found that users preferred gestures designed by end-users and researchers over those designed by researchers alone, seemingly because researchers proposed more physically and conceptually complex gestures than end-users. Thus, it is important to involve the end-user in the design stages of a multitouch application. We designed our gestures for Eden with the help of a veteran set construction artist, one of our target users.

2.6 Multitouch Frameworks

While the utility of a multitouch application is dependent on its design, multitouch developers rely on developer tools for bringing such a design to fruition. Current multitouch frameworks borrow heavily from mouse-based frameworks, but multitouch gestures have different recognition requirements. Multitouch application developers must track not only when an individual finger touches down and lifts up from the device, but also the simultaneous movement of multiple fingers on the device. They often track the progression of multitouch gestures by managing state machines, a common way to track the progression of user input. The regular expressions used by Proton, which we describe in Chapter 6, are closely related to finite-state machines: regular expressions describe regular languages; finite-state machines accept such languages [123]. In Section 2.6.1 we summarize prior research on modeling user input as state machines and formal languages. In Section 2.6.2 we then describe related work on multitouch event-handling and the declarative specification of multitouch gestures.

2.6.1 Modeling Input with State Machines & Formal Languages

Since Newman's pioneering work on Reaction Handler [100], researchers have modeled user interactions using formalisms such as state machines [4, 50, 54, 99], context-free grammars [19, 57, 56] and push-down automata [104]. These formalisms have been used to specify the user interactions of an application [19], to describe the context of an interaction [58, 121], and to synthesize working interface implementations [104]. A recurring theme in early work is to split user interface implementation into two parts: the *input language* (the set of possible user interface actions); and the *application semantics* for those actions. The input language is often defined using state machines or grammars; the semantics are defined in a procedural language. Proton takes a conceptually similar approach: it uses regular expressions as the underlying formalism for declaratively specifying sequences of touch events that comprise a gesture. Callbacks to procedural code are associated with positions within the expres-

sion. Proton goes beyond the earlier formalisms by also providing a static analyzer to detect conflicts between gestures and a graphical notation to further simplify gesture creation.

2.6.2 Multitouch Event-Handling

With the recent rise of multitouch cellphones and tablet computers, hardware manufacturers have created a variety of interaction frameworks [6, 44, 92] to facilitate application development. These frameworks inherit the event-based callback [103] structure of mouse-based GUI frameworks. While commercial frameworks usually support a few common interactions natively (e.g., pinch-to-zoom), implementing a new gesture requires processing low-level touch events. Open-source multitouch frameworks similarly require developers to implement gestures via low-level event-handling code [28, 39, 49, 101, 124]. Lao et al. [79] presents a state-transition diagram for detecting multitouch gestures from touch events. This diagram serves as a recipe for detecting simple gestures, but the developer must still process the touch events. Many frameworks are written for specific hardware devices. Kammer et al. [60] describe the formal properties shared by many of these frameworks. Echtler and Klinker [34] propose a layered architecture to improve multitouch software interoperability; their design also retains the event callback pattern. However, none of these multitouch frameworks give developers a direct and succinct way to describe a new gesture.

2.6.3 Describing Gestures Declaratively

Researchers have used formal grammars to describe multitouch gestures. CoGesT [43] describes conversational hand gestures using feature vectors that are formally defined by a context-free grammar. Kammer et al. [61] present GeForMT, a formal abstraction of multitouch gestures also using a context-free grammar. Their grammars describe gestures at a high level and do not provide recognition capabilities. Gesture Coder [83] recognizes multitouch gestures with state machines, which are equivalent to regular expressions. While Proton and Gesture Coder share a recognition approach, their interfaces for authoring gestures differ significantly: developers demonstrate gestures in Gesture Coder; they author gestures symbolically using tablatures and regular expressions in Proton.

In multitouch frameworks such as GDL [68] and Midas [117] developers declaratively describe gestures using rule-based languages based on spatial and temporal attributes (e.g., number of touches used, the shape of the touch path, etc.). Because these rule-based frameworks are not based on an underlying formalism such as regular expressions, it is difficult to reason about gesture conflicts at compile time. The developer must rely on heavy runtime testing to find such conflicts. In contrast to all previous techniques, Proton provides static analysis to automatically detect conflicts at compile time.

Chapter 3

Selecting Targets with Multiple Direct Touches

Multitouch workstations offer several potential benefits over mouse-based workstations including *direct-touch* input, *bimanual* interaction, as well as same-hand *multifinger* interaction. Yet, because prior work has examined direct-touch, bimanual and multifinger interactions separately, they have not been able to quantify and compare the benefits of direct-touch and potential increases in parallelism from multitouch. We examine direct-touch, bimanual and multifinger interactions all together in the context of a multitarget selection task using the traditional mouse-based workstation as a performance baseline.

We focus our study on target selection because it is one of the most common tasks in current graphical user interfaces including tools for computer animation. Although less common than single selection, multitarget selection is a frequent task in everyday computer usage. For example, users often select multiple files or photos for reorganization or multiple objects in a graphical drawing program for grouping. With a mouse-based workstation and a single input touchscreen, users are forced to serially select multiple targets. In contrast, a multitouch workstation that detects multiple contact points permits the possibility of selecting the targets in parallel. In theory users could simultaneously select targets with all ten fingers on both hands and thereby increase performance.

In our study we measure and compare the performance of four input methods for selecting multiple targets. The input methods include: 1) one mouse (indirect, unimanual), 2) one finger (direct-touch, unimanual), 3) two fingers, one on each hand (direct-touch, bimanual), and 4) any number of fingers (direct-touch, bimanual, multifinger). From this empirical data we determine the speedup due to direct-touch input over indirect mouse input as well as the additional speedup due to using two fingers, and finally the contribution of unrestricted use of all fingers.

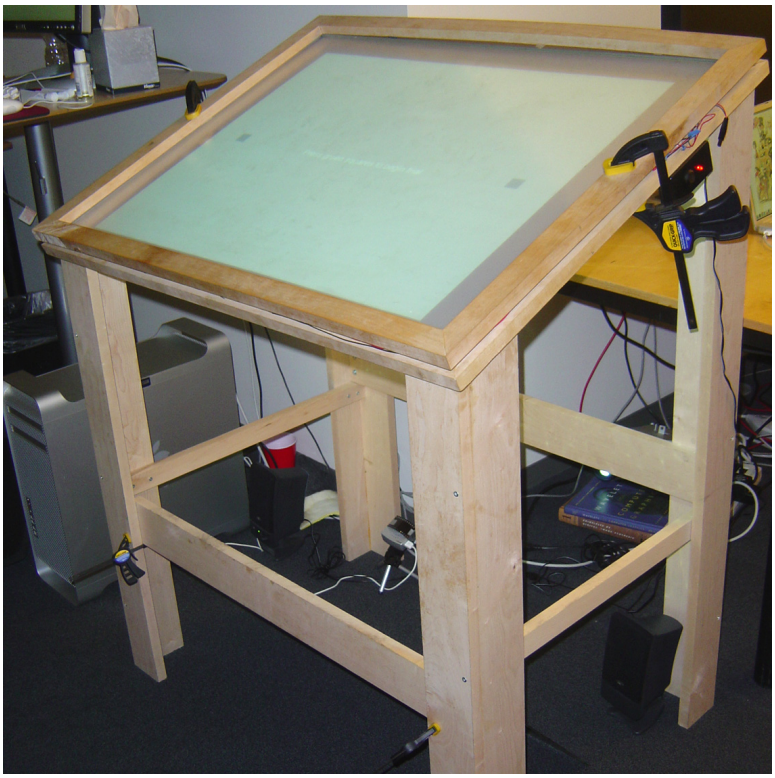


Figure 3.1: Our multitouch workstation.

3.1 Multitarget Selection Experiment

We expect one finger direct-touch to outperform the mouse for multitarget selection, since in the unimanual case multitarget selection requires a series of single target selections, which is faster with direct-touch than with the mouse [120]. We further expect that two-finger bimanual multitarget selection will be faster than one-finger unimanual selection due to the parallelism achieved from two hands [11, 29, 53, 66, 67]. Two hands also provide a division of labor for multitarget selection. Each hand can work in its respective area of the screen and thus does not need to travel long distances [31]. Finally, bimanual, multifinger selection has the potential to exhibit even more overlapping action as each finger on each hand can simultaneously move towards a different, nearby target.

Note however, that even though both the two finger and multifinger conditions permit parallelism, the one finger condition has some advantages that can make up for its serial nature. Using a single hand rather than two hands can decrease occlusion due to the hands and thereby make it easier to see the targets on the display surface. In addition, since the one finger condition forces serializing the task, users may realize it more quickly when they miss a target and which target they missed. They may be able to correct this mistake before moving too far away.

3.1.1 Participants and Apparatus

We recruited eight participants (7 male, 1 female) who were all experienced desktop workstation users, but had no significant experience with a multitouch workstation. Seven participants were right-handed, and one participant claimed to be ambidextrous but used the right hand to control the mouse.

To increase the ecological validity of our study, we chose our workstation configurations for the task to reflect current best practices for professional single-user applications. We used a Dell optical wheel mouse for the indirect input with mouse acceleration enabled and a 30" HP LP3065 monitor at its native resolution of 2560 x 1600 pixels. Using the Mac OS X system preferences keyboard and mouse controls, we set the mouse tracking speed to a comfortable level so that a user could traverse the entire screen without the need for reclutching the mouse. We used the built-in Mac OS X mouse acceleration profile.

We built the multitouch workstation used in this experiment (Figure 3.1) – it is patterned after a drafting table and uses the frustrated total internal reflection technique described by Han [47]. The table is capable of detecting an arbitrary number of simultaneous of touches. The size of the screen is 76.2 x 57.2 cm with a resolution of 1024 x 768 pixels. Participants stood in front of the screen, which was mounted at a 23 degree incline off horizontal.

Because large multitouch workstations have not yet been widely adopted, we believe that there is no clear consensus on best practice methods for adapting single-user applications, originally designed to work with a mouse and a desktop display, to the larger tabletop display format of a multitouch workstation. Therefore, we designed our experiment to work comfortably for desktop display sizes and then uniformly scaled them up for use on our multitouch display. To guarantee uniform scaling, it is important that the desktop and multitouch displays use the same aspect ratio. Since the native aspect ratios differ (4:3 for multitouch, 8:5 for the desktop), we chose to use only a 4:3 portion (53.3 x 40.0 cm, 2133 x 1600 pixels) of the HP monitor for the mouse condition.

3.1.2 Task and Experiment Design

To separate the effects of direct-touch, bimanual and multifinger interaction, we compared the performance on a multitarget selection task across four input conditions: 1) one *mouse* (indirect, unimanual), 2) *one finger* (direct-touch, unimanual), 3) *two fingers*, one on each hand (direct-touch, bimanual), and 4) unconstrained *multifinger* input (direct-touch, bimanual, multifinger). For the one finger condition, we required participants to use the same finger during the duration of a trial but were permitted to change the finger between trials. For the two finger condition, we instructed participants to use one finger on each hand and to use those same two fingers during the duration of a trial. We allowed participants to change fingers between trials. For the multifinger condition, we allowed participants to use all ten fingers. Participants could choose to select targets serially or in parallel in the two finger and multifinger conditions.

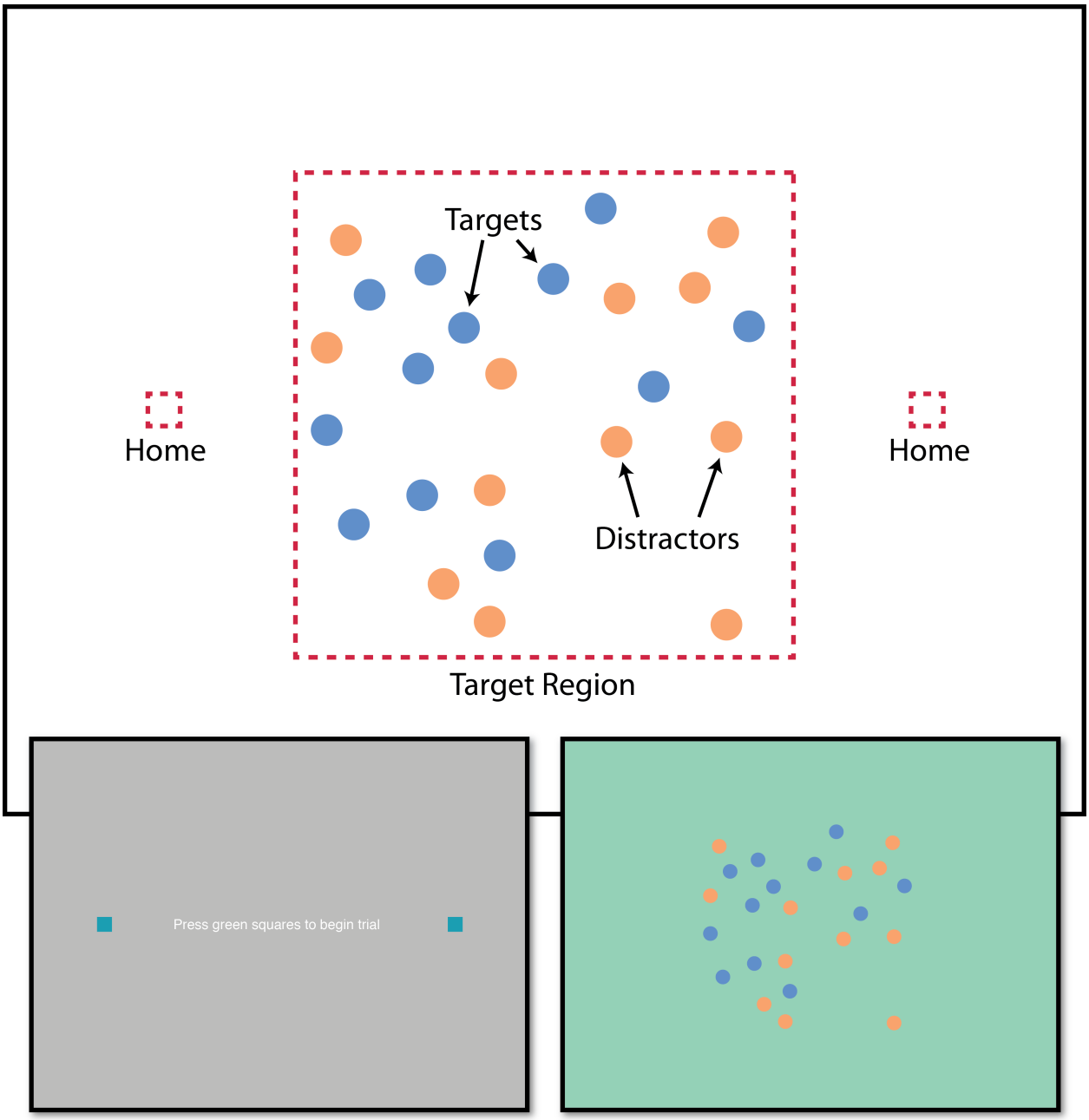


Figure 3.2: Top: Conceptual layout of the multitarget selection task. Bottom: Screenshot of home positions and screenshot of task. Text in left screenshot reads: "Press green squares to begin trial."

Our experiment did not include a two-mouse condition because we did not expect it to perform well based on the results of prior work. Forlines et al. [37] found the time to select two corners of a rectangle to be much longer for two mice than for two fingers on a multitouch display. Balakrishnan and Hinckley [11] suggested that visually tracking two mice is difficult for users especially when no line or other visual mark connects the two. In our own experience operating two mice, we found that selecting targets, especially with the non-dominant hand, was imprecise and indeed difficult.

To begin a trial, participants first selected home positions. For the one mouse and one finger conditions, there was a single home position located on the right side of the screen. For the two finger and multifinger conditions, there were two home positions, one located on each side of the screen (Figure 3.2). After selecting the home position(s), the participant was immediately presented with blue target discs and orange, non-selectable distractor discs that were randomly placed between the two home positions. All discs were presented simultaneously. We instructed participants to select all the blue targets as quickly and as accurately as possible for each trial. We included the orange distractors to mimic cases in which a user could not simply lasso select all the targets and had to select each one individually. We counted a target as hit when the mouse cursor fell within the target on a mouse click or when the center of a finger's contact area fell within the target. On successful hits, the system played a *ding!* sound. A trial ended after all blue targets were selected. Our multitouch workstation is incapable of identifying which finger or which hand corresponds to each touch. We identified the finger and hand after the experiment, by manually reviewing a video recording of the multitouch trials.

We varied the number of blue targets across five levels: 3, 6, 9, 12, and 15, with an equal number of orange distractors. The diameter of both the targets and distractors were 1.5 cm and 2.1 cm for the mouse-based and multitouch workstations respectively. We chose these diameters based on the recommendations of Sears and Shneiderman [120], and we believe they are large enough for us to expect that direct-touch selection would be faster than mouse-based selection. For placing targets and distractors, we used a Poisson disk distribution, which assigns targets and distractors random positions, while ensuring that they are some minimum distance apart. All of the targets and distractors were separated by at least the width of a target. The centers of all targets and distractors were situated in the 22.6 x 24.0 cm and 32.3 x 34.3 cm region of the center of the screen for the mouse and multitouch workstations respectively. The home positions were placed on the horizontal midline of the screen, 7.5 cm and 10.7 cm away from the target region for the mouse and multitouch workstations respectively.

We used a within-subject design so that each participant used each of the four input methods in four blocks of trials, with five levels for number of targets and two trials per level. To account for ordering effects we used a Latin square to set the order of the input methods across participants. Thus, for each participant there were 4 input methods x 4 blocks x 5 levels x 2 trials = 160 total trials. Participants had the option to rest between trials. To mitigate learning effects we required participants to execute one block of 10 practice trials when initially starting with a new input method. After completing the practice block,

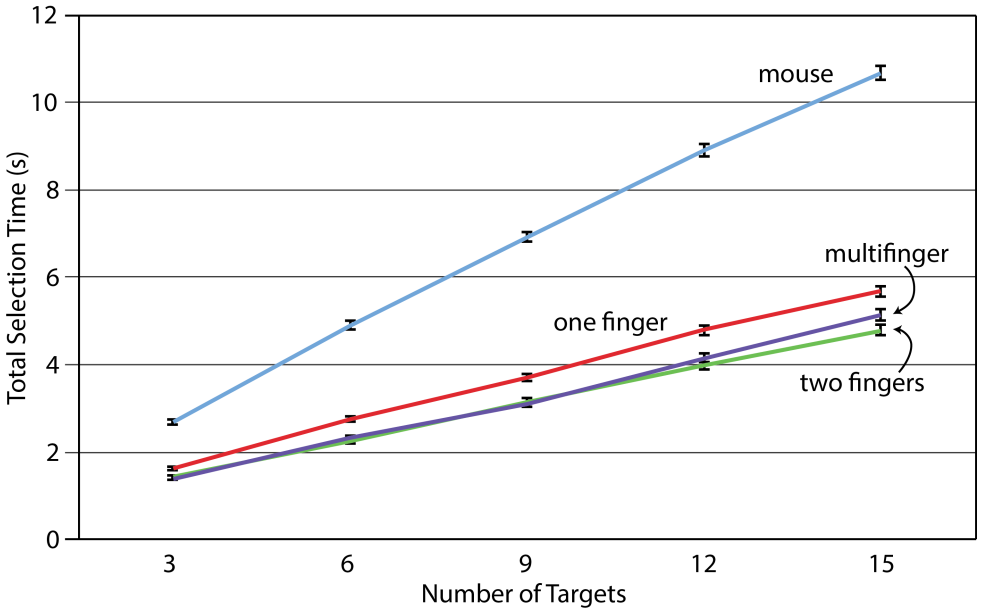


Figure 3.3: Average total selection times of the four input methods as a function of number of targets. The standard error bars are shown.

participants could perform as many additional practice trials as they wanted until they felt comfortable enough to begin the experiment.

3.2 Results

For each trial of the experiment we recorded the total selection time. We started the timer immediately after the participant touched the home position(s) to begin a trial and stopped the timer as soon as the participant successfully finished selecting all presented targets. Because misses occur naturally in real-world target selection tasks, we included the time to correct misses in the total selection time. The average selection times are shown in Figure 3.3 with a detailed summary of the data in Table 3.1. A repeated measures ANOVA on the selection times found significant main effects for input method ($F_{3,21}$=418.02, p<.001) and number of targets ($F_{4,28}$=509.77, p<.001), as well as a significant interaction between these two independent variables ($F_{12,84}$=99.18, p<.001). The effect size of the interaction (η^2=.071) was small relative to the effect sizes of the input method (η^2=.372) and the number of targets (η^2=.464). We attribute the interaction to the observation that the selection time for the mouse increases at a much faster rate than for the other input conditions as the number of targets increases (Figure 3.3).

Pairwise comparisons of selection times for each pair of input methods revealed significant differences between all pairs (p<.012 between one finger and multifinger and p<.001 for the rest), with the exception of the difference between the two finger and multifinger conditions

Average selection time in seconds

	mouse	one finger	two fingers	multifinger
3	2.69 (.053)	1.62 (.035)	1.44 (.031)	**1.37** (.021)
6	4.90 (.102)	2.75 (.057)	**2.26** (.060)	2.32 (.049)
9	6.93 (.111)	3.72 (.077)	3.14 (.090)	**3.10** (.072)
12	8.91 (.135)	4.79 (.108)	**4.00** (.099)	4.15 (.105)
15	10.68 (.160)	5.68 (.113)	**4.79** (.123)	5.14 (.133)
avg time	6.82	3.71	3.12	3.22

Table 3.1: The rows correspond to number of targets and the columns correspond to input method. Each table contains the average time to select the corresponding number of targets using the corresponding input method. Standard errors are noted in parentheses. The fastest selection time per number of targets is given in bold. The last row contains the average completion time for each input condition.

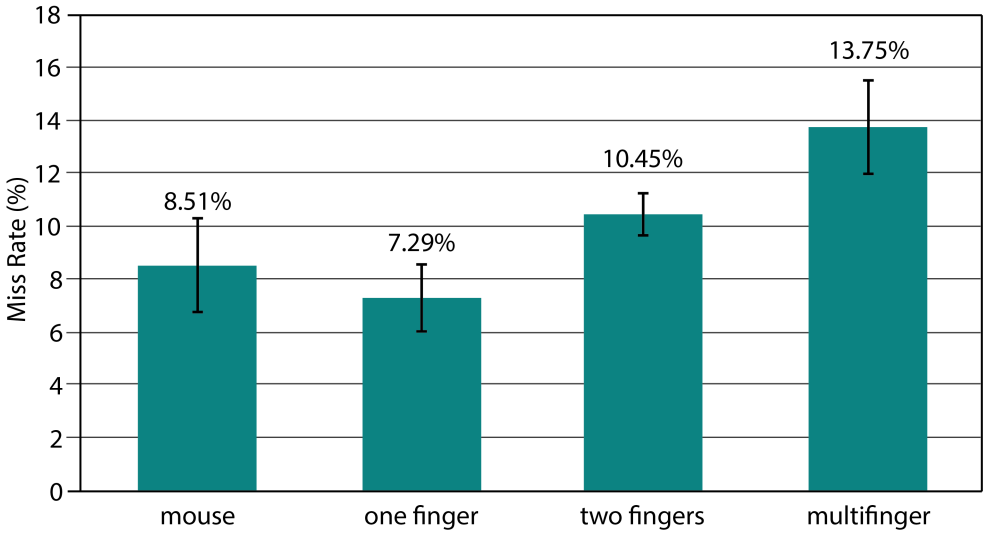


Figure 3.4: Average miss rate per input method, with standard error bars.

(p=.426). The fastest multitouch condition (either two finger or multifinger) was always between 1.96 and 2.33 times faster than the mouse. The two finger and multifinger conditions were consistently faster than one finger, which was consistently faster than the mouse condition. The direct-touch, one finger condition accounted for about 83% of the reduction in selection time, while bimanual interaction, either two finger or multifinger, accounted for the remaining reduction. There was little difference in average selection times between the two finger and multifinger conditions even in the densest condition of 15 targets.

We counted a miss each time the participant performed a mouse click or touch that did not successfully hit a target. We also recorded a miss if the participant hit a target that was

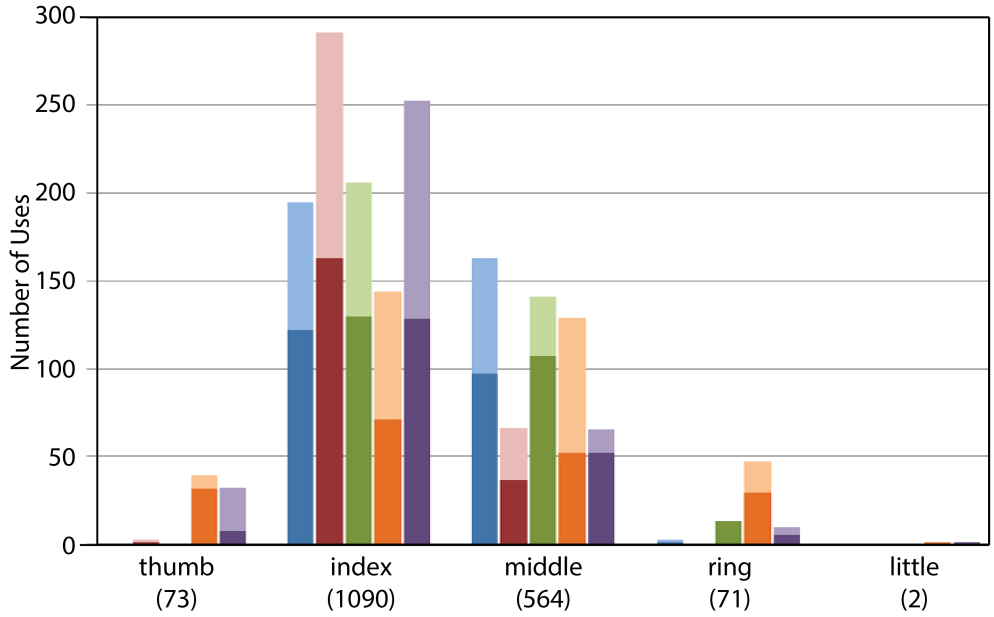


Figure 3.5: Number of uses per finger type across participants in parentheses. Distinct colors denote distinct participants. The dark and light hues correspond to the right and left hand contributions respectively.

already selected. Each participant had to select a total of 360 targets for each input method. As shown in Figure 3.4 participants had an average miss rate of 8.51% for the mouse, 7.29% for one finger, 10.45% for two fingers, and 13.75% for multifinger. A repeated measures ANOVA found a significant effect due to input method ($F_{3,21}$=3.72 p=.027) and number of targets ($F_{4,28}$=46.11 p<.001) as well as a significant interaction between these two factors ($F_{12,84}$=3.74 p<.001). However, pairwise comparisons across pairs of input methods revealed that the only significant difference in miss rate is between the one finger and multifinger conditions (p<.001). The effect size of the number of targets (η^2=.144) was much larger than the effects of the input device (η^2=.030) and the interaction (η^2=.030).

For the one and two finger conditions, one participant exclusively used middle fingers and five participants exclusively used index fingers. The other two participants used index fingers for the two finger condition, but for the one finger condition, one participant used the middle finger, while the other participant used the index finger but switched to the middle finger between trials.

For the multifinger condition, five participants opted to use more than two fingers, while the remaining three participants preferred to just use two fingers, one on each hand. Figure 3.5 shows the number of times each finger was used to select a target, aggregated across the five participants that used more than one finger on each hand. Out of 1800 total uses of the different fingers, these five participants used the index fingers most frequently with 1090 uses and they used the little fingers least frequently with two uses. The data also showed

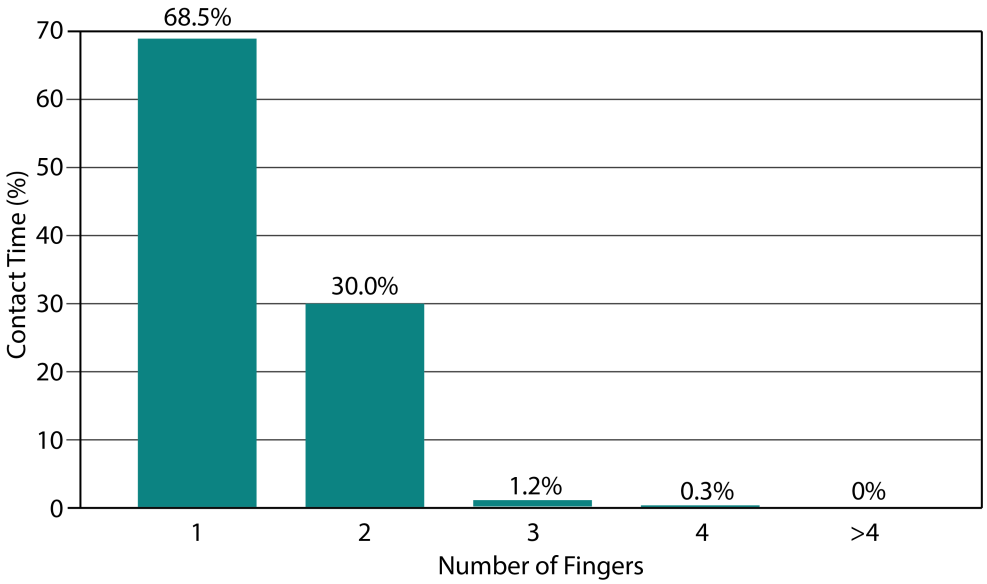


Figure 3.6: Proportion of time users spent with different number of fingers simultaneously in contact with the display surface.

that across all participants in the multifinger condition, one finger was in contact with the multitouch surface 68.5% of the time, two fingers were simultaneously in contact 30.0% of the time, and three and four fingers were simultaneously in contact 1.2% and 0.3% of the time respectively (Figure 3.6). We never observed more than four fingers in contact with the surface simultaneously.

In addition to measuring selection time and miss rates we also collected subjective preference data from the participants in an exit survey. We asked the participants to rate the mouse-based workstation versus the multitouch workstation on a scale 1-9, with 1 indicating preference for the mouse and 9 indicating preference for multitouch. The average rating was 8.4. We also asked the participants which of the four input conditions they thought was fastest for selection. Four participants rated themselves as fastest using two fingers, while three thought they were fastest in the multifinger condition. One participant thought he was fastest with just one finger. The proponents for two fingers thought that using multifinger caused more screen occlusion from fanning their fingers, and found it difficult to orient multiple fingers on a single hand and target multiple targets at the same time. Those who thought multifinger was the fastest condition liked the option of being able to reach for a target with a different finger on the same hand.

3.3 Discussion

The goal of our study was to compare the performance of one mouse, one finger, two finger and multifinger techniques for multitarget selection. Consistent with previous work, we found that using one finger direct-touch is faster than using a mouse and, as expected, bimanual interactions are faster than using one finger. About 83% of the decrease in selection time was due to moving from the indirect mouse to the one finger direct-touch with remaining benefit of about 27% due to moving from unimanual multitouch to bimanual multitouch.

Miss rates were similar across the four input methods. However, the bimanual conditions tended to have higher miss rates than the unimanual conditions and the difference of about 6.5% between one finger and multifinger is significant. Even though misses occurred more frequently in the bimanual conditions and we included the time to correct these misses in the total selection times, we found that users are still able to successfully complete the multitarget selection task faster using two hands.

Although the multifinger condition provides more opportunity for simultaneously selecting multiple targets, it does not perform better than the two finger condition, at least for novice multitouch users. The data reveals a strong preference for using one or two index fingers, even when given the opportunity to use more. We found that in the multifinger condition users rarely put more than two fingers in contact with the table simultaneously (just 1.5% of the cases) and never placed more than four fingers in contact simultaneously (Figure 3.6). These results suggest that the tracking of two simultaneous contacts may be enough to support even multifinger multitarget selection. The relatively high miss rate for multifinger may be due to increased cognitive load required to plan movements for multiple fingers independently. In addition, fingers on the same hand are physically constrained to the palm, which limits the set of targets these fingers can touch simultaneously.

Our study utilized targets with a diameter of 2.1 cm, which is large enough for users to select them with direct-touch more quickly than with the mouse. Targets for traditional mouse and keyboard interfaces are not always that large, so developers should consider adjusting target sizes when using multitouch input for an interface designed for the mouse and keyboard.

3.3.1 Design Guidelines

Based on our experiment we recommend the following set of design guidelines for developing applications for multitouch workstations. Since our studies focus on multitarget selection, all of these guidelines are aimed at applications where target selection is the primary task.

- A one finger direct-touch device delivers a large performance gain over a mouse-based device. For multitarget selection tasks even devices that detect only one point of touch contact can be effective.

- Support for detecting two fingers will further improve performance, but support for detecting more than two fingers is unnecessary to improve multitarget selection performance.
- Reserve same-hand multifinger usage for controlling multiple degrees of freedom or disambiguating gestures rather than for independent target selections.
- Uniformly scaling up interfaces originally designed for desktop workstations for use with large display direct-touch devices is a viable strategy as long as targets are at least the size of a fingertip.

3.3.2 Limitations

It is worth noting that several aspects of our experimental design limit generalizability of our results. For example, we do not vary the size or shape of our targets. As a result targets of differing sizes and shapes may affect performance. Prior work has shown that targets must be larger than the size of a fingertip to obtain good performance with multitouch devices [120].

While our experiment does not explicitly control density of targets at a fine-grained level, we do test different numbers of targets. Since we place targets randomly based on a Poisson disk distribution, it is unclear how well our results would generalize to the case when targets are placed extremely close (within a target diameter) to one another. Moreover, real-world applications rarely lay out targets randomly and therefore our results may not generalize cleanly to more realistic applications. The performance impact of bimanual and multifinger interactions may increase or decrease depending on the layout and clustering of targets. Nevertheless, because our experiments are based on randomized layouts, they may serve as baseline data for the more structured target layouts.

Learning, practice effects and muscle-memory may also play important roles in target selection. A practiced pianist or touch-typist can simultaneously target keys in fixed locations with many fingers. Such learning effects and chording for multitarget selection on a multitouch workstation require further study.

3.4 Conclusion

Our multitarget selection experiment demonstrated that multitouch has significant performance advantages over the mouse for selecting multiple targets. The developer should design multitouch applications such that targets are large, to ensure that they are easily selectable. Users can then quickly select multiple objects, such as files or photos for reorganization. Applications for creating digital presentations and vector graphics also contain many selectable objects. With multitouch, users can quickly select multiple objects to perform group manipulations. Although target selection is a common task across graphical user interfaces, it is just one of many possible actions a user can perform. We continue to investigate multitouch bimanual interaction for drawing strokes in the next chapter.

Chapter 4

Drawing Directional Strokes Bimanually for Menu Selection

In Chapter 3 we demonstrated that direct-touch input outperforms mouse input for target selection. We also found that bimanual target selection, using one finger on each hand, provides the maximum benefit. To investigate whether bimanual interaction also improves performance in drawing directional strokes, we developed and tested two-handed multi-stroke marking menus.

Introduced by Kurtenbach and Buxton [76, 77, 78], marking menus are gesture-based menus that allow users to select a menu item by drawing a directional stroke. These menus exhibit a number of desirable properties. Marking menus are *scale-independent* – the selection depends only on the orientation of the stroke, not on its length, and therefore users can efficiently draw short strokes with ballistic motions to select items [86, 98]. Users can draw strokes *in-place* and do not have to make large round-trip traversals to select items from a fixed location menu. Moreover, users can draw the straight-line strokes in an *eyes-free* manner without diverting attention from their primary task. Finally, marking menus provide a seamless *novice-to-expert transition* path; novices draw exactly the same selection strokes as experts.

A drawback of marking menus is that selection accuracy depends on menu *breadth*, or the number of items that appear at a single level of the menu. Kurtenbach and Buxton [76] found that accuracy declines substantially when breadth is greater than eight items. To increase the number of menu items available, researchers have added menu *depth*, or the number of levels in the menu hierarchy. Compound-stroke [76] and multi-stroke [148] marking menus allow for hierarchical traversal of marking menus using either zig-zag strokes or a sequence of strokes. Multi-stroke marking menus have the added benefit that users can draw each stroke in the same location, conserving desk and display space. At breadth-8, however, compound-stroke and multi-stroke techniques perform well only up to depth-2 or depth-3 respectively. More recent techniques have used additional stroke attributes such as stroke position [147] and curvature [9] to further increase menu breadth. However, all of these techniques have focused on one-handed marking menus with either mouse or stylus-based input devices.

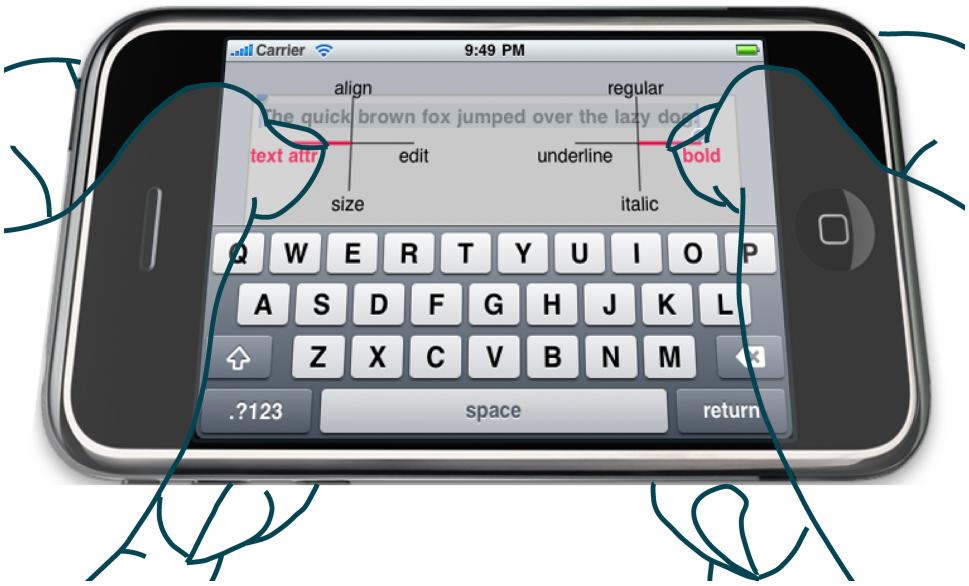


Figure 4.1: Using a two-handed ordered marking menu, the left thumb strokes to select "Text Attributes" and then the right thumb selects "Bold" to modify the sentence. With a two-handed simultaneous marking menu, users draw both strokes at the same time.

Marking menus are a good match for multitouch devices because they do not require precise targeting of individual menu items. Instead, marking menus rely on coarse directional movement, thereby circumventing the fat finger problem [110]. Unlike a mouse or stylus, multitouch devices detect multiple points of contact and therefore support two-handed interactions. These devices have the potential to significantly increase the efficiency of interaction because users can overlap their hand motions and work with both hands in parallel.

We examine the speed and accuracy of one and two-handed multi-stroke marking menus with two multitouch devices: a small-screen Apple iPod Touch operated with the thumbs (Figure 4.1) and a Fingerworks iGesture [36] operated with the index or middle fingers as one would on a large-screen interactive surface. We introduce two new two-handed variants of multi-stroke marking menus:

Two-Handed Simultaneous: Users draw two strokes, one with each hand, at the same time. This variant is designed to maximize parallelism in hand motions and thereby offer the fastest selection times.

Two-Handed Ordered: Users alternate the hand used to draw each stroke. Since either hand (left or right) can start the stroke sequence, this variant offers access to twice as many menu items for the same number of strokes, while also allowing for some temporal overlap in hand motions.

We compare the speed and accuracy of these two-handed designs to one-handed marking menus. The anatomy of the hand imposes constraints on the range of motions different fingers can make. It may be easier to draw individual strokes or pairs of strokes in some

directions rather than others. To better understand these constraints, we also examine how stroke direction affects speed and accuracy. We conclude with a set of design guidelines that multitouch designers should consider when developing one- or two-handed multi-stroke marking menus at both handheld (iPhone/iPod Touch) and larger (iGesture/iPad) scales. We present several demonstration applications that show how two-handed marking menus could be used to support real-world tasks and to facilitate the transition from novice to expert use.

4.1 Related Work

In addition to multi-stroke marking menus, our two-handed multi-stroke marking menus build on touch-based and bimanual menu techniques.

Marking Menus on Touch Devices: Touchpads that can track a single point of contact have been commonplace on laptops for the last decade. Balakrishnan and Patel [10] integrated such a touchpad with a mouse to allow the non-dominant hand to select commands using a compound-stroke marking menu. Isokoski and Käki [55] found that curved strokes were more accurate but slower in movement time than drawing straight-line selection strokes. Touch-sensing screens are now commonplace on mobile devices. Karlson et al. [63] used directional strokes for thumb-based navigation on a PDA. Yatani et al. [146] used a combination of position and directional strokes to disambiguate the selection of closely packed items on a touch-based mobile device. Lepinski et al. [82] developed chording marking menus in which users draw simple directional strokes using combinations of fingers on a single hand. While all of these stroke-based techniques are designed for touch-based devices, none of them have examined the use of multiple strokes in different directions or two-handed interactions.

Bimanual Menu Techniques: Odell et al. [102] presented an asymmetric bimanual marking menu technique in the context of a shape drawing system. The dominant hand selects a shape and the non-dominant hand selects a command to perform on this shape using a marking menu. Unlike this approach we develop symmetric two-handed marking menus in which both hands perform the same actions. By splitting the strokes of a multi-stroke marking across both hands, we allow for overlap in the hand motions and increase the speed of the interaction.

Controllers for console-based gaming systems such as the Xbox [144] usually include two joysticks, one for each hand. Wilson and Agrawala [136] developed a two-joystick based text-entry system using an onscreen keyboard and showed that such a symmetric bimanual approach is faster than using the default, single joystick technique. TwoStick [75] extends Quikwriting [109], a technique that uses directional joystick movements to enter text, for use with two joysticks. Weegie [132] is another two-stick-based text entry system in which each stick operates a separate marking menu. Unlike our two-handed marking menus, the two menus in Weegie work independently of one another.

4.2 Designing Two-Handed Marking Menus

The number of menu levels and number of menu items per level determine the total number of menu items accessible by a multi-stroke marking menu. The *breadth* of the menu corresponds to the number of menu items at each level, which is the number of possible stroke directions. The *depth* of a menu corresponds to the number of levels, which is the number of strokes required to traverse the menu hierarchy.

A breadth-M, depth-N marking menu thus has $M * N$ menu items. We extend multi-stroke marking menus for use on multitouch devices by splitting the stroke sequence between two hands. Thus, each hand is responsible for half the number of strokes. We consider several aspects of two-handed operation that can further increase menu selection performance:

Temporal Overlap of Motion: Users can temporally overlap motions of their hands and this parallelism can reduce the time required to complete the interaction.

Hand Identity (Left/Right): Multitouch devices detect multiple points of contact; we can use heuristics based on contact position to infer the hand corresponding to each contact – e.g., the left-most touch is from the left hand and the right-most touch is from the right hand. We can then increase the number of menu items that are accessible with a single stroke, by assigning a different set of items to each hand. With a hierarchical marking menu of depth-N in which either hand can draw each stroke in the sequence, we can use hand identity to increase the number of accessible items by a factor of 2^N. However, if the same hand is used to draw more than one consecutive stroke the potential for temporal overlap in the hand motions is reduced.

Chunking: Buxton [21] has shown that users can mentally group together frequently co-occurring compound motor tasks into a single chunk that they automatically perform together. With two-handed multitouch devices, users can draw a pair of strokes simultaneously, one with each hand, and may learn to chunk these pairs together into a single action. Thus, users can mentally flatten two levels of a multi-stroke hierarchy into a single level, and convert a breadth-M, depth-2 marking menu into a breadth-M^2, depth-1 menu. Such increased breadth may allow interface designers and users to fit more items in a single cognitive grouping.

Based on these design considerations we propose the following two-handed multi-stroke marking menus designs:

4.2.1 Two-Handed Simultaneous Marking Menus (2HS)

Users simultaneously draw two strokes, one with each hand (Figure 4.2 Left). Users can draw additional stroke pairs to traverse a menu hierarchy. This variant is designed to maximize the temporal overlap of motions and also facilitate chunking of the stroke pairs into a single action. However, this variant does not use hand identity as an extra bit of information. Therefore, for a given number of strokes it does not increase the number of accessible menu items over the one-handed multi-stroke marking menu design. However, when users chunk

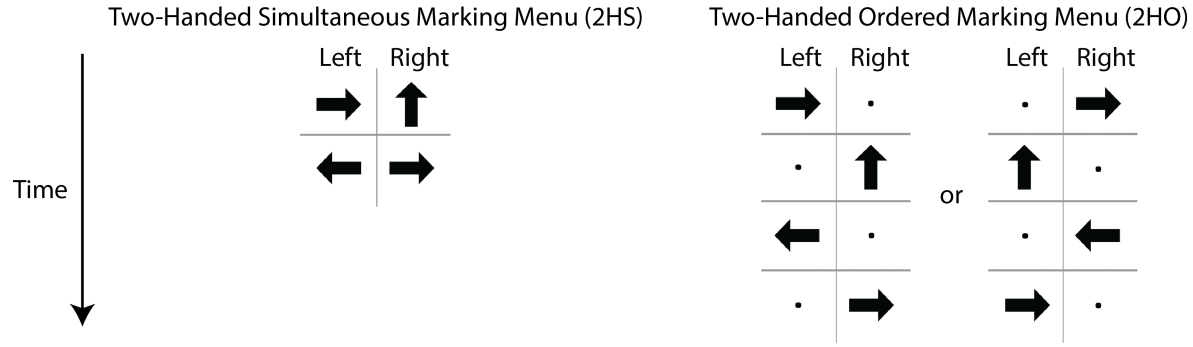


Figure 4.2: Left: With the two-handed simultaneous marking menu, users can draw pairs of strokes simultaneously. Right: With the two-handed ordered marking menu, users alternate drawing strokes between hands. The identity of the initiating hand provides an additional bit of information, doubling the number of accessible menu items.

pairs of simultaneous strokes this variant can be considered as flattening the depth and thus, squaring the breadth of the menu.

4.2.2 Two-Handed Ordered Marking Menus (2HO)

Users draw a sequence of strokes with alternating hands (Figure 4.2 Right). Although the strokes must be drawn in order, the ordering only depends on the start-time of each stroke. Users can begin a second stroke before the first stroke is complete to increase temporal motion overlap. This variant considers hand identity, but because the hands are forced to alternate, only the hand initiating the stroke sequence can vary (left or right), providing one additional bit of information. Thus, this approach doubles the number of accessible menu items for a fixed number of strokes. Although using hand identity could in theory increase the number of accessible menu items by a factor of 2^N, our ordered design forces users to alternate hands to maximize the potential for temporal motion overlap in stroke drawing.

4.3 User Study 1: Comparison of One- and Two-Handed Marking Menus

To investigate the performance benefits of our two-handed multi-stroke marking menu designs we conducted an initial user study comparing both of our designs to standard one-handed multi-stroke marking menu designs. To simplify analysis we selected only right-hand dominant participants, but we included both right- and left-handed unimanual marking menus in our study. We conducted the experiment using two multitouch devices – an iPod Touch to represent handheld interaction and the iGesture to represent larger-screen interaction. Our hypotheses were:

Figure 4.3: The Fingerworks iGesture multitouch pad.

H1: The two-handed simultaneous menu is faster for selecting menu items than all other menus including two-handed ordered and one-handed menus. The two-handed simultaneous design maximizes the opportunity to temporally overlap hand motions and therefore reduces selection time.

H2: The two-handed ordered menu is faster than a one-handed marking menu for selecting a menu item. Users do not have to wait for one stroke to finish before starting the next stroke and are able to overlap their hand movements.

H3: Of the one-handed conditions the right-handed multi-stroke marking menu outperforms the left-handed multi-stroke marking menu. Since our participants are right-handed, their dominant hand moves more quickly and accurately than their non-dominant, left hand.

Two-handed motions on touch devices may have different constraints than the mouse and stylus strokes investigated in prior work [76, 77, 98, 147, 148]. Studies of bimanual motion suggest that mirrored pairs of strokes may be easier to draw than other pairs [65, 87]. Thus, our study also investigates how well users can draw directional strokes with their left and right hands individually, and how well they can draw pairs of strokes with two hands.

4.3.1 Participants and Apparatus

We recruited 16 right-handed participants (12 male, 4 female, between 21 and 26 years old). All were experienced computer users and ten were experienced iPhone or iPod Touch users. None of the participants had experience with marking menus or with large multitouch screens. Participants performed the experiment using two multitouch devices:

Apple iPod Touch: The 2nd generation iPod Touch is a commonly used handheld device with a multitouch screen. It has a working area of 7.5 $\times$ 5 cm and display resolution

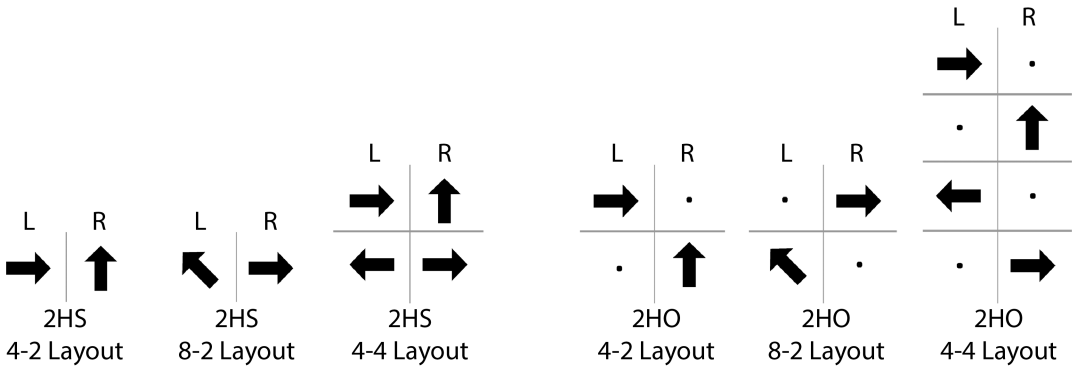


Figure 4.4: Example stimuli for the two-handed simultaneous (2HS) and two-handed ordered (2HO) conditions. Arrows appear in separate columns to indicate the hand that should draw the stroke. Pairs of arrows in the same row indicate strokes that must be drawn simultaneously. Participants must draw strokes in order from top to bottom. In the one-handed conditions (not shown), the stimuli only contain arrows in either the left or right column.

of 480 × 320 pixels. Participants used their thumbs on this device.

Fingerworks iGesture Pad: The iGesture is an indirect multitouch pad with a working area of 16.5 × 12.4 cm (Figure 4.3) mapped by absolute coordinates to a 40.6 × 30.5 cm Dell monitor with a display resolution of 1600 × 1200 pixels. The study ran on Mac OS X. Participants used either their index fingers or middle fingers on this device.

4.3.2 Task and Stimuli

We designed the study to test expert-level performance. However, our participants had practically no prior experience using marking menus. Training participants to use multi-stroke marking menus with realistic menu items would force them to learn a complex menu organization. But, an expert user would require little effort to recall the necessary strokes for a command. To better elicit expert-level performance with far less training, we adopted the strategy of previous marking menu studies [76, 147, 148] and gave participants stimuli in the form of arrows that directly indicated the strokes they should draw.

Examples of the stimuli we used are shown in Figure 4.4. Arrows appeared in separate columns to indicate the hand that should draw the stroke. Pairs of arrows in the same row indicated strokes that must be drawn simultaneously. Participants had to draw the strokes in order from top to bottom. In the one-handed conditions (not shown in figure), the stimuli only contained arrows in either the left or right column.

To begin a trial the participant tapped the device with one finger for the one-handed conditions or one finger on each hand for the two-handed conditions. The stimulus appeared at the top of the screen and, following the approach of Zhao et al. [147], as soon as the participant touched the input device, the cue disappeared so that the participant could not

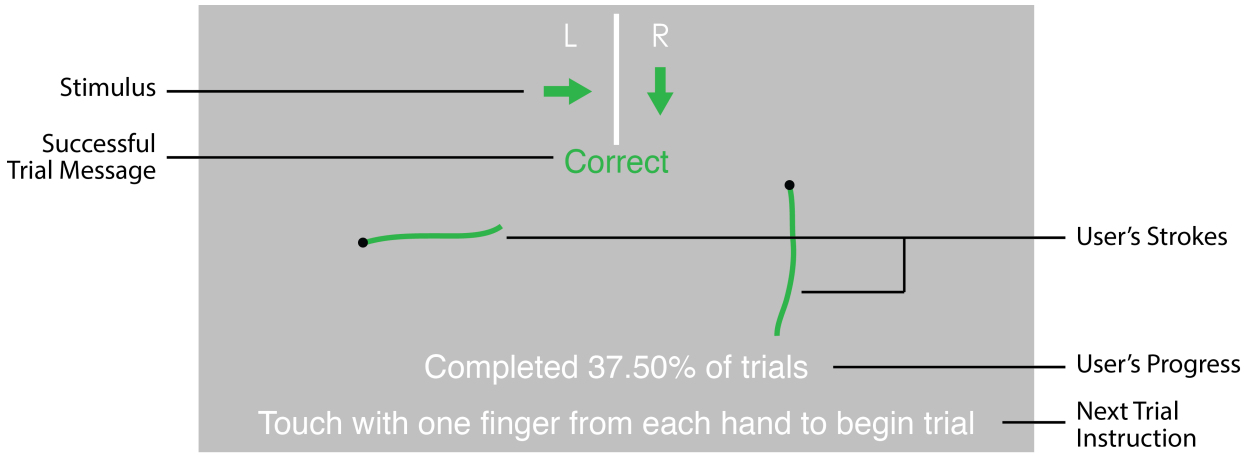


Figure 4.5: Screenshot of experimental setup with feedback given after a successful trial.

read the arrows while drawing strokes. This approach was designed to better elicit expert-level performance because it prevented participants from interleaving drawing the strokes and reading the stimulus. We asked the participant to draw the strokes as quickly and accurately as possible.

After the participant made the designated number of strokes (two or four strokes in our experiment), a feedback screen showed whether or not the trial was successful and the percentage of trials completed so far (Figure 4.5). On correct trials we colored the strokes and stimuli green, and on incorrect trials we colored them red. The participant could rest between trials while the feedback was onscreen and they could continue to the next trial by tapping on the device.

4.3.3 Study Design

Our experiment used a within-subjects design and included three independent variables: *device*, *menu technique*, and *menu layout*. We fully counterbalanced the device variable so that half the participants used the iPod Touch first and the other half used the iGesture first.

Participants had to select menu items using one of four menu techniques: left-handed multi-stroke marking menu (1HL), right-handed multi-stroke marking menu (1HR), two-handed simultaneous multi-stroke marking menu (2HS), or two-handed ordered multi-stroke marking menu (2HO). We used a Latin square to counterbalance the ordering of the menu techniques.

For each menu technique we tested three different *breadth-depth* menu layouts: 4-2, 8-2, and 4-4. A breadth-4 layout includes only the four cardinal directions, whereas a breadth-8 layout includes both the cardinal and diagonal directions. We only considered the number of strokes in multiples of two, so that strokes could be evenly distributed between hands for the two-handed conditions. Although the two-handed techniques would work with an

odd number of strokes, we believe it should be possible to extrapolate performance for those conditions using the data we collected for even numbers of strokes.

We fixed the ordering of the three layouts from least to most complex (4-2, 8-2, 4-4). As the number of accessible menu items or stroke combinations increases, more trials are necessary to obtain good coverage. Our three layouts allow a total of 16, 64, and 256 possible stroke combinations and we used 24, 32, and 32 trials respectively. For the 4-2 layout, each stroke combination was performed at least once, in randomized order. For the 8-2 layout there are four possible pairs of on- and off-axis strokes: on-on, on-off, off-on, off-off. We randomized the stroke combinations such that each participant performed eight trials from each axis grouping. For the 4-4 layout, we randomly chose the stroke combination from all possible combinations for that layout. For the two-handed ordered condition, we randomized the order of the starting hand with half the trials beginning with the left hand.

We considered a trial to be a miss if any one of the strokes was drawn in an incorrect direction. To check for misses, we compared the angle of the line segment connecting the start and end points of the drawn stroke with the angle of each possible stroke direction in the menu. If the angle of the drawn stroke was closest to the angle cued in the stimulus it was considered correct, otherwise it was considered a miss. We added each missed trial to the end of the trial queue so that users would have to perform it again until successful.

Before testing each menu layout, we gave participants a practice block to train them in reading the stimulus and move them towards expert-level performance. For the 4-2 conditions we required 20 practice trials, while we required 8 practice trials for the 8-2 and 4-4 conditions. In all cases participants had the option to continue practicing until they felt comfortable with the task. The entire experiment took each participant roughly one hour.

We measured four dependent variables: *reaction time*, *movement time*, *total time*, and *accuracy*. Reaction time was the interval between the first display of the stimulus and the start of the touch beginning the first stroke. It represents the time required for participants to process the stimulus and decide which strokes to draw. Movement time was the interval between the first touch and completion of all strokes and represents the time required to physically draw the strokes. Total time was the sum of the reaction and movement times. We only considered timing data from correct trials to better account for expert-level performance. We computed accuracy as the fraction of correctly performed trials to the total number of incorrect trials. We did not include the trials that were added to the end of the queue as the result of a miss in our accuracy measure.

4.4 Results

We performed a 2 (device) $\times$ 4 (technique) $\times$ 3 (layout) repeated measures ANOVA for each dependent variable. The average total, reaction, and movement times are shown in Figure 4.6 for the iPod Touch and in Figure 4.7 for the iGesture. The corresponding numerical results are shown in Tables 4.1, 4.2, and 4.3 for total, reaction, and movement times respectively.

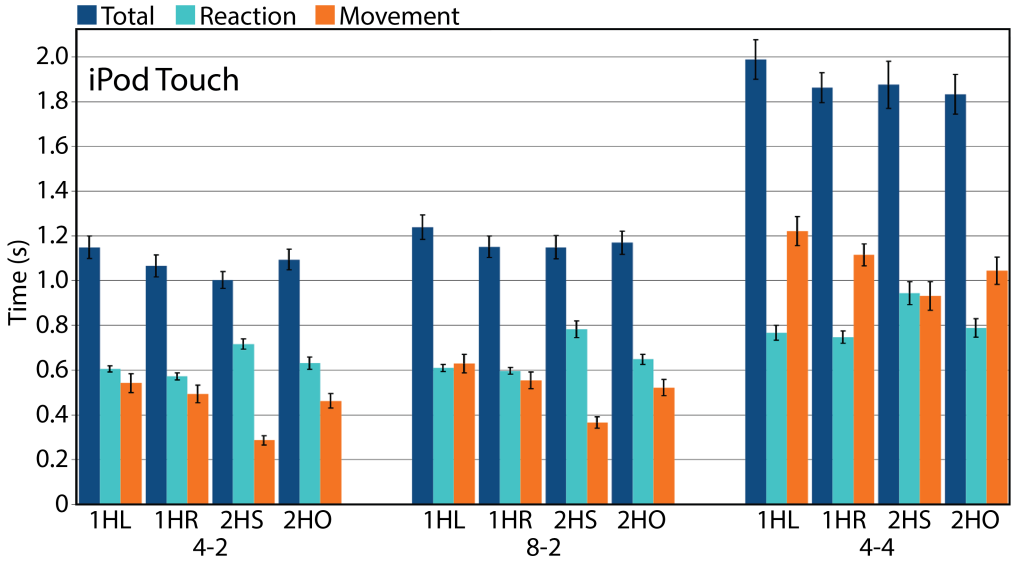


Figure 4.6: Average times (with standard error bars) for each menu technique and menu layout on the iPod Touch.

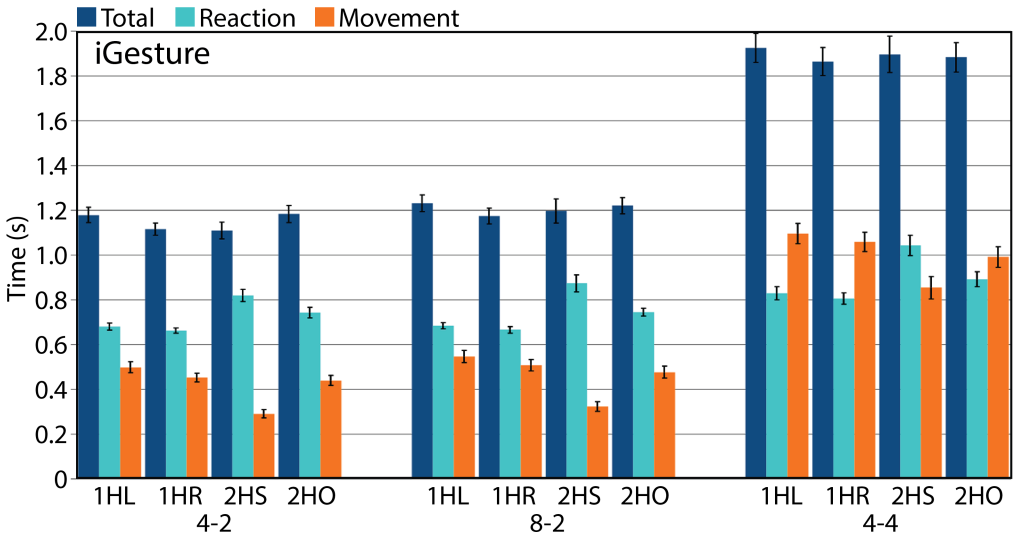


Figure 4.7: Average times (with standard error bars) for each menu technique and menu layout on the iGesture.

The Figures 4.8 and 4.9 show the average accuracy rates for the iPod Touch and iGesture respectively and Table 4.4 contains the corresponding numerical results.

For each of the menu layouts, we also compared the performance between same-axis stroke pairs and different-axis stroke pairs. Finally, for the two-handed conditions, we examined differences in the amount of temporal overlap of the two hands and the effects of the starting hand on performance for the ordered technique.

Total time in milliseconds

	iPod Touch			iGesture			
	4-2	**8-2**	**4-4**	**4-2**	**8-2**	**4-4**	**All**
1HL	1149 (52)	1238 (57)	1988 (91)	1179 (36)	1231 (39)	1926 (67)	1452 (51)
1HR	1065 (51)	1151 (49)	1861 (69)	1116 (29)	1174 (36)	1864 (65)	1372 (46)
2HS	1003 (38)	1149 (54)	1875 (109)	1110 (39)	1197 (55)	1897 (84)	1372 (59)
2HO	1094 (48)	1170 (54)	1832 (91)	1183 (39)	1221 (38)	1884 (68)	1397 (52)

Table 4.1: Average total times in milliseconds and standard errors in parentheses.

Reaction time in milliseconds

	iPod Touch			iGesture			
	4-2	**8-2**	**4-4**	**4-2**	**8-2**	**4-4**	**All**
1HL	606 (15)	610 (17)	767 (35)	680 (16)	684 (14)	829 (30)	696 (17)
1HR	572 (15)	597 (14)	747 (28)	663 (12)	666 (15)	805 (26)	675 (16)
2HS	716 (24)	783 (40)	944 (52)	820 (28)	874 (40)	1042 (46)	863 (34)
2HO	631 (29)	648 (24)	788 (42)	743 (25)	744 (18)	893 (35)	741 (24)

Table 4.2: Average reaction times in milliseconds and standard errors in parentheses.

4.4.1 Total Time

As shown in Table 4.1, the average total times pooled across device and layout were 1452 ms for 1HL, 1372 ms for 1HR, 1372 ms for 2HS, and 1397 ms for 2HO. We found significant main effects for technique ($F_{3,45}$=6.45, p=.001) and layout ($F_{2,30}$=378.70, p<.001). We also found a significant two-way interaction between device and technique ($F_{3,45}$=5.25, p=.003) indicating that technique affects total time differently on the two devices. Thus, we performed separate post hoc multiple means comparison tests with Bonferroni correction for each device. For the iPod Touch we found that 1HL was significantly slower than the three other techniques across all three layouts (p<.008). For the iGesture the only significant result we found was that 1HL was slower than 1HR across all layouts (p=.006). These results suggest that for both devices and across all layouts, the non-dominant, left-handed menu is significantly slower than the dominant, right-handed menu. In addition, the total times required by the two-handed menus are not significantly different from the total time required by the right-handed menu.

4.4.2 Reaction Time

As shown in Table 4.2, the average reaction times pooled across device and layout were 696 ms for 1HL, 675 ms for 1HR, 863 ms for 2HS, and 741 ms for 2HO. We found significant main effects for device ($F_{1,15}$=88.26, p<.001); the average reaction time for the iPod Touch (701 ms) was faster than for the iGesture (787 ms). We also found significant main effects

Movement time in milliseconds

	iPod Touch			iGesture			
	4-2	**8-2**	**4-4**	**4-2**	**8-2**	**4-4**	**All**
1HL	542 (44)	629 (43)	1221 (67)	499 (26)	547 (28)	1097 (47)	756 (37)
1HR	493 (41)	555 (39)	1115 (51)	453 (20)	509 (27)	1059 (45)	697 (33)
2HS	287 (21)	366 (26)	931 (26)	291 (18)	324 (22)	854 (52)	509 (31)
2HO	463 (34)	521 (38)	1044 (63)	440 (24)	477 (27)	991 (47)	656 (35)

Table 4.3: Average movement times in milliseconds and standard errors in parentheses.

for technique ($F_{3,45}$=62.16, p<.001) and layout ($F_{2,30}$=61.13, p<.001). In addition, we found two-way interactions between technique and device ($F_{3,45}$=4.36, p=.009) as well as technique and layout ($F_{6,90}$=2.57, p=.024) indicating that technique affects reaction time differently for each device and for each layout. For the iPod Touch, we ran post hoc multiple means comparison tests with Bonferroni correction and found that 2HS was significantly slower than all other techniques regardless of layout (p<.003). We found only one other significant difference between 1HR (597 ms) and 2HO (648 ms) for the 8-2 layout (p<.020). For the iGesture, post hoc multiple means comparison tests with Bonferroni correction found no significant differences between the one handed techniques (1HL and 1HR) across all layouts. We also found that 2HS was significantly slower than all other techniques for every layout (p<.001) with one exception – in the 4-2 layout we found no significant difference between 2HS and 2HO. All other pairs were significantly different ($p\leq$.026). These results suggest that the two-handed simultaneous technique has the slowest reaction times of all the techniques.

4.4.3 Movement Time

As shown in Table 4.3, the average movement times pooled across device and layout were 756 ms for 1HL, 697 ms for 1HR, 509 ms for 2HS, and 656 ms for 2HO. We found a significant main effect for device ($F_{1,15}$=7.48, p=.015); the average movement time for the iGesture (628 ms) was faster than for the iPod Touch (681 ms). We also found significant main effects for technique ($F_{3,45}$=74.91, p<.001) and layout ($F_{2,30}$=399.22, p<.001). Post hoc multiple marginal means comparison tests with Bonferroni correction found that movement times between all pairs of techniques were significantly different (p=.035 for pair 1HR-2HO, p<.001 for all other pairs), regardless of device and layout. The only significant two-way interaction was between device and layout ($F_{2,30}$=5.18, p=.012). These results suggest that the two-handed techniques have faster movement times than the one-handed techniques. The two-handed simultaneous technique is faster than the two-handed ordered technique, while the right-handed technique is faster than the left-handed technique.

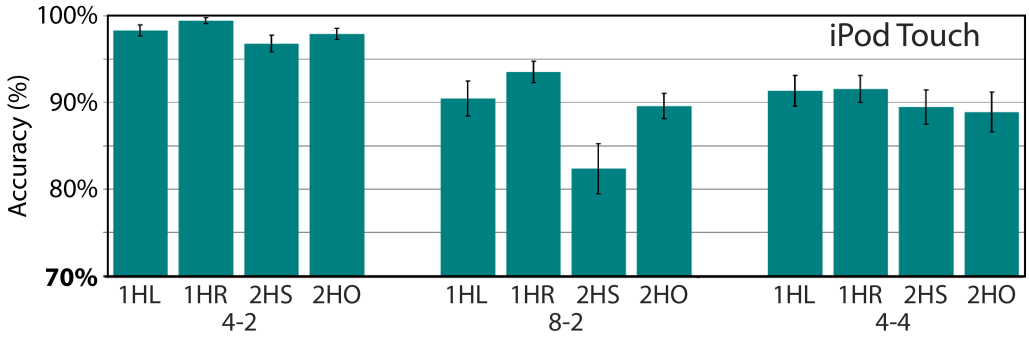


Figure 4.8: Average accuracies (with standard error bars and baseline value of 70%) for each menu technique and menu layout on the iPod Touch.

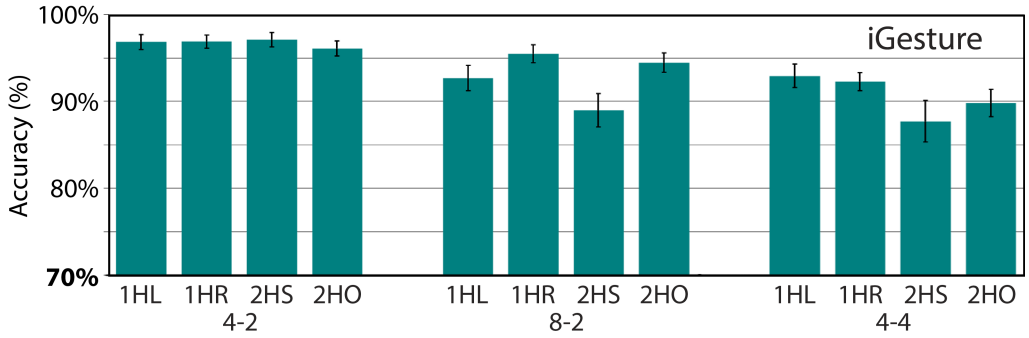


Figure 4.9: Average accuracies (with standard error bars and baseline value of 70%) for each menu technique and menu layout on the iGesture.

Accuracy (%)

	iPod Touch			**iGesture**			
	4-2	**8-2**	**4-4**	**4-2**	**8-2**	**4-4**	**All**
1HL	98.4 (0.7)	90.4 (2.1)	91.4 (1.8)	96.8 (0.9)	92.7 (1.5)	92.9 (1.4)	93.7 (0.7)
1HR	99.5 (0.4)	93.5 (1.3)	91.6 (1.6)	96.9 (0.8)	95.5 (1.1)	92.3 (1.1)	94.9 (0.5)
2HS	96.8 (1.0)	82.5 (2.9)	89.5 (2.0)	97.1 (0.8)	89.0 (2.0)	87.7 (2.5)	90.4 (1.2)
2HO	97.9 (0.6)	89.6 (1.5)	88.9 (2.4)	96.1 (0.9)	94.5 (1.1)	89.9 (1.6)	92.8 (0.8)

Table 4.4: Average accuracy and standard errors in parentheses.

4.4.4 Accuracy

As shown in Table 4.4, the average accuracy rates pooled across device and layout were 93.7% for 1HL, 94.9% for 1HR, 90.4% for 2HS, and 92.8% for 2HO. We found significant main effects for technique ($F_{3,45}$=9.03, p<.001) and layout ($F_{2,30}$=32.95, p<.001). In addition, we found a significant two-way interaction between technique and layout ($F_{6,90}$=3.36, p=.005),

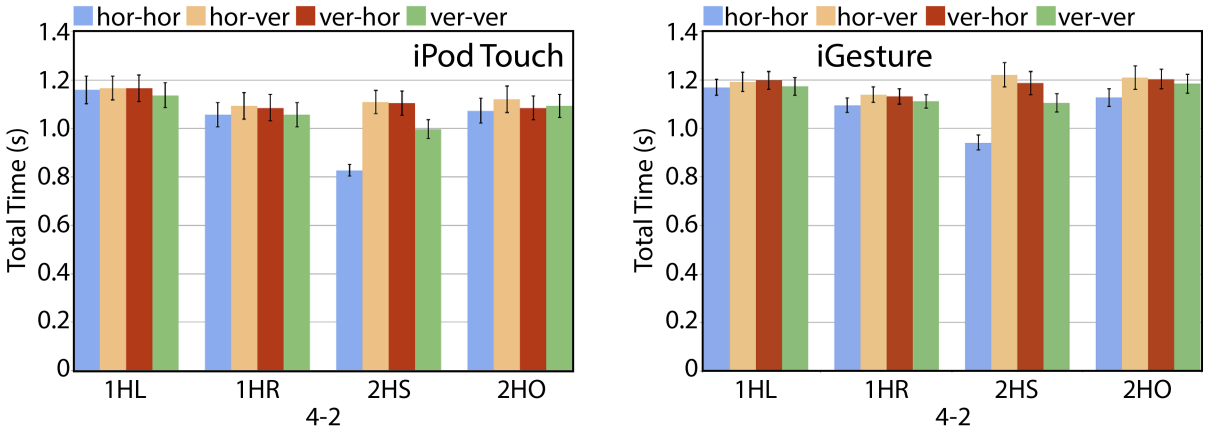


Figure 4.10: Average total times for horizontal or vertical stroke groupings for the 4-2 layout using the iPod Touch (Left) and the iGesture (Right). Standard error bars are shown.

indicating that technique affects accuracy differently for each layout. Post hoc tests found no significant differences in accuracy between the four techniques for the breadth-4 layouts. However, for the 8-2 layout we did obtain a a significant simple main effect for technique ($F_{3,45}$=11.63, p<.001) where 2HS was less accurate than 1HR (p<.001) and 2HO (p=.020). We also found a significant two-way interaction between layout and device ($F_{2,30}$=13.94, p<.001) as the iPod Touch was more accurate than the iGesture for layout 4-2 (p=.012), but less accurate for layout 8-2 (p=.006). These results indicate that there are no significant differences in accuracy between all four techniques at breadth-4 but that the two-handed simultaneous technique is less accurate than other techniques at breadth-8.

4.4.5 4-2 Layout Stroke Groupings

For the 4-2 layout, we compared horizontal-horizontal, horizontal-vertical, vertical-horizontal, and vertical-vertical groupings of stroke pairs. Total times for the stroke groupings are shown in Figure 4.10. Conducting a 2 (device) × 4 (technique) × 4 (stroke grouping) repeated measures ANOVA with total time as the dependent variable, we found significant main effects for device ($F_{1,15}$=9.24, p=.008), technique ($F_{3,45}$=8.23, p<.001), and stroke grouping ($F_{3,45}$=59.87, p<.001). On average, horizontal-horizontal strokes (1057 ms) and vertical-vertical strokes (1108 ms) were faster than mixed stroke pairs (1157 ms for horizontal-vertical and 1146 ms for vertical-horizontal). We also found a significant two-way interaction between technique and stroke grouping ($F_{9,135}$=35.44, p<.001); the 2HS technique had a larger difference in speed between same-axis pairs and different-axis pairs, than the other techniques. In addition, for the 2HS condition, stroke pairs that were either bilaterally symmetric or in the same direction (translationally symmetric) (894 ms) were faster than the remaining pairs (1147 ms) by 28%. For accuracy we found no significant main effect for stroke grouping nor any significant interactions between any of the factors. These results show that the average

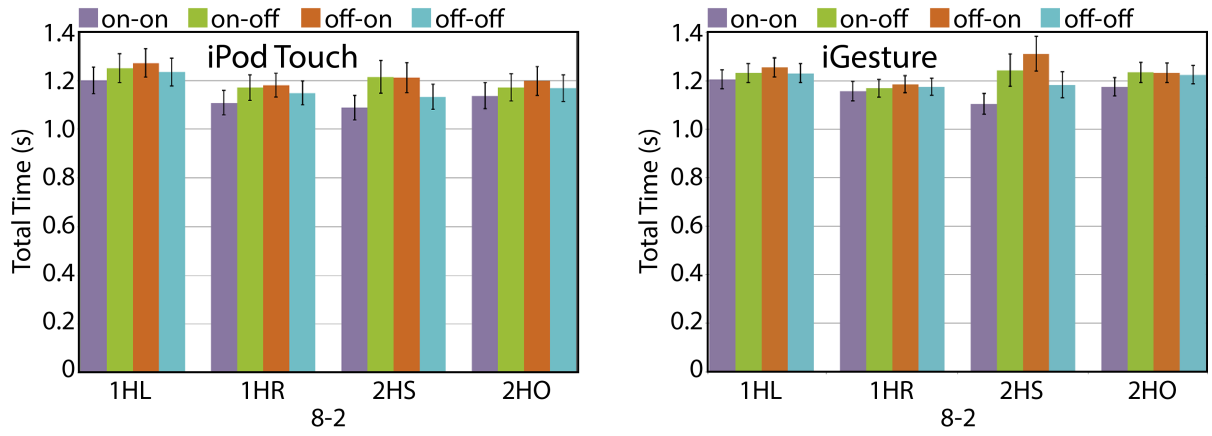


Figure 4.11: Average total times for on- or off-axis stroke groupings for the 8-2 layout using the iPod Touch (Left) and the iGesture (Right). Standard error bars are shown.

total times for same-axis pairs is faster than for strokes along different axes and that the largest differences in speed occurs for the two-handed simultaneous technique.

4.4.6 8-2 Layout Stroke Groupings

We compared on-on, on-off, off-on, and off-off axis selections for the 8-2 layout. An on-axis stroke is one drawn in a cardinal direction while an off-axis stroke is one drawn along a diagonal. Total times for the stroke groupings are shown in Figure 4.11. Conducting a 2 (device) $\times$ 4 (technique) $\times$ 4 (stroke grouping) repeated measures ANOVA with total time as the dependent variable, we found significant main effects for technique ($F_{3,45}$=3.90, p=.015) and stroke groupings ($F_{3,45}$=46.59, p<.001). On average, on-on strokes (1148 ms) and off-off strokes (1188 ms) were faster than mixed stroke pairs (1212 ms for on-off and 1232 ms for off-on). There was also a significant two-way interaction between technique and stroke groupings ($F_{9,135}$=6.01, p<.001), as the 2HS technique had a bigger speed difference between pairs that were on-on or off-off compared to the mixed pairs than the other techniques. We also found a significant three-way interaction ($F_{9,135}$=2.73, p=.006). For accuracy we found no significant main effect for stroke grouping nor any significant interactions between any of the factors. These results show that the average total times for on-on and off-off pairs is faster than mixed pairs and that the largest differences in speed occurs for the two-handed simultaneous technique.

4.4.7 Temporal Overlap

To assess the level of temporal overlap in the two-handed conditions, we computed the delay between the initial touch of the first hand and the initial touch with the second hand. A shorter delay indicates a greater likelihood of overlap between the hand motions. We

performed a 2 (device) × 2 (2HS and 2HO techniques) × 3 (layout) repeated measures ANOVA with delay as the dependent measure. We found significant main effects for technique ($F_{1,15}$=157.47, p<.001) and layout ($F_{2,30}$=10.45, p<.001). The average delay was shorter for 2HS (44 ms) than for 2HO (261 ms). We also found an interaction between technique and device ($F_{1,15}$=12.16, p=.003), as the delay for 2HS was slightly longer for the iGesture than the iPod Touch, but the delay for 2HO was slightly longer for the iPod Touch than for the iGesture. These results confirm that the two-handed simultaneous technique has a much shorter delay between initial touches and greater temporal overlap than the two-handed ordered technique.

4.4.8 Starting Hand for Two-Handed Ordered

For 2HO we compared whether using the left hand or the right hand for the first stroke had an impact on performance. We performed separate 2 (hand) × 3 (layout) × 2 (device) repeated measures ANOVAs on total time and accuracy. For total time we found significant main effects for device ($F_{1,15}$=6.65, p=.021) and layout ($F_{2,30}$=229.12, p<.001). More interestingly, however, we found no significant main effect for hand ($F_{1,15}$=0.31, p=.584), nor any interaction effects between any of the factors, which indicates that the starting hand does not significantly affect total time, regardless of layout and device. For accuracy we found only a significant main effect for layout ($F_{2,30}$=12.17, p<.001). Again, we found no significant main effect for hand ($F_{1,15}$=2.10, p=.168), nor any significant interactions between hand and the other two factors, which indicates that there was no significant difference in accuracy due to starting hand, regardless of layout and device. The only significant two-way interaction was between device and layout ($F_{2,30}$=4.97, p=.014). These results suggest that the starting hand does not affect performance (speed or accuracy) of the two-handed ordered technique.

4.4.9 Single Stroke Direction

To better understand which individual stroke directions are easiest to perform with the thumbs and fingers, we examined the movement time for the first stroke in the 8-2 layout for both of the one-handed conditions (Figures 4.12 and 4.13). One participant did not have any trials for one of the directions with the right hand in the iPod Touch condition, so we removed that person's data from the right hand analysis. After removing this data we could not perform a complete multi-way repeated measures ANOVA across all factors. Therefore, for the iPod Touch, we ran separate ANOVAs for each hand with stroke direction as the factor and we found that stroke direction had a significant effect on movement time for both the left hand ($F_{7,105}$=3.74, p=.001) and the right hand ($F_{7,98}$=3.46, p=.002). Examining the average movement times in more detail, we found that the left thumb was fastest at selecting strokes in the upper left quadrant (N, NW, W directions) and the right thumb was fastest at selecting strokes in the upper right quadrant (N, NE, E directions). In addition, on-axis stroke directions tended to be faster than their neighboring off-axis directions. Pooling the data across all directions, we found that the right thumb (161 ms) was significantly faster

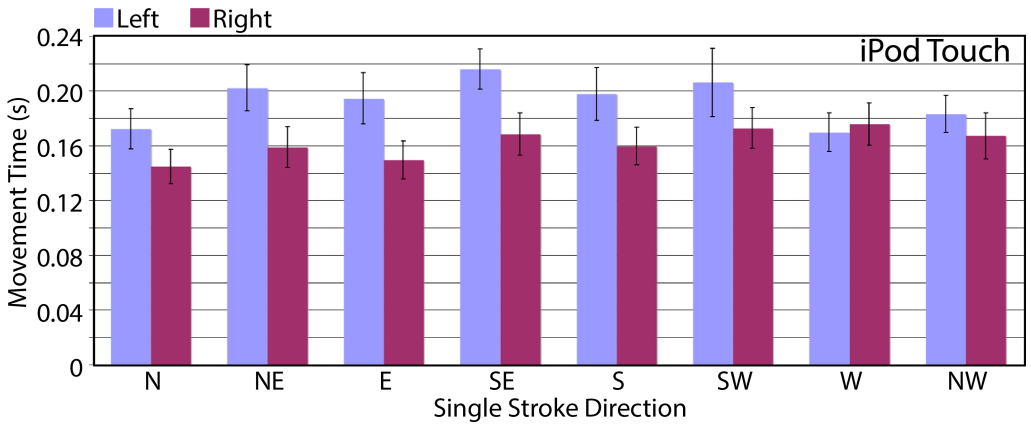


Figure 4.12: First stroke average movement (with standard error bars) times using the iPod Touch for the 8-2 layout.

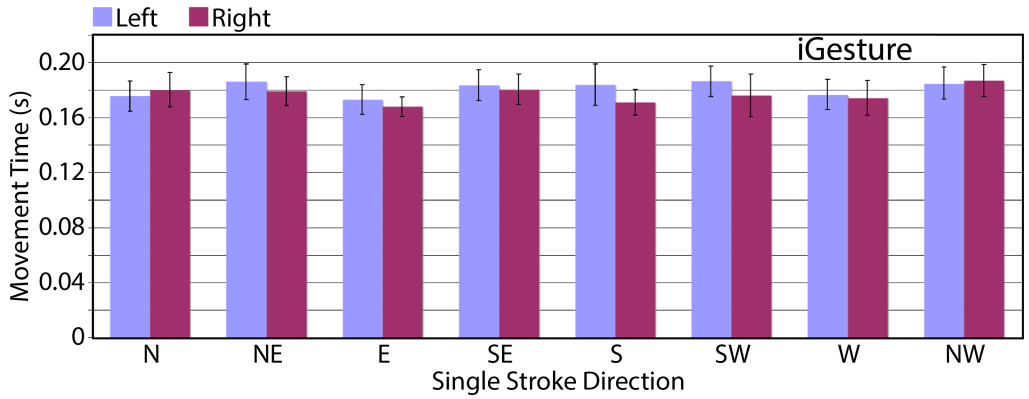


Figure 4.13: First stroke average movement (with standard error bars) times using the iGesture for the 8-2 layout.

than the left thumb (191 ms) (p=.005) at drawing a single stroke. These results show that the right thumb is faster than the left thumb, and both thumbs are faster at moving upwards and inwards, with respect to the hands, than in other directions.

For the iGesture, we ran a two-way ANOVA comparing with hand and stroke direction as the factors. We found no significant main effects for hand ($F_{7,105}$=0.83, p=.565) or direction ($F_{1,15}$=0.36, p=.559), and we found no significant interaction effect ($F_{7,105}$=0.49, p=.840). These results indicate that on the iGesture, where a participant uses either the index or middle finger, neither the hand nor the direction significantly affects movement time of a single stroke.

4.5 Discussion

The overall goal of our study was to investigate the performance of multi-stroke marking menus in the context of multitouch devices. We summarize the results as follows:

Two-handed simultaneous marking menus are as fast as dominant-handed multi-stroke marking menus. We hypothesized (*H1*) that 2HS would outperform the other menu techniques because it maximizes temporal overlap in hand motion. However, the total time for 2HS was very similar to that for 1HR and we found no statistically significant differences between them. Although 2HS was fastest in movement time, 16.5-41.8% faster on the iPod Touch and 19.4-36.4% faster on the iGesture than 1HR, it also incurred a slower reaction time, indicating that users spent more time remembering and planning their strokes when coordinating simultaneous motions of two hands. These results do not allow us to accept hypothesis *H1*. However, based on our own experience with 2HS we believe that with practice users can cognitively chunk the simultaneous two-stroke gestures and greatly reduce their reaction time. The two strokes proceduralize into a single automated action, like a form of "muscle memory." As we describe in Section 4.6, we conducted a longitudinal study to test this hypothesis.

At breadth-4, 2HS was just as accurate as the other menu designs, but at breadth-8, which requires more precise motions and thus more careful coordination, 2HS was less accurate. Nevertheless, accuracy remained above 82% across all conditions we tested and can improve significantly with practice, which we show in Section 4.6.

Two-handed ordered marking menus are as fast as dominant-handed multi-stroke marking menus and provide access to twice as many items. Although we hypothesized (*H2*) that 2HO would be faster than the one-handed designs (1HR and 1HL) due to overlap in hand movement, we found that total time and accuracy were not significantly different between 2HO and 1HR. The reaction times were faster for 2HO than for 2HS, but slightly slower than for the one-handed menus. This result indicates that planning finger movements takes more time for the two-handed menus. Since starting with the right hand or left hand makes no significant difference on total time or accuracy for the 2HO design, both alternating orders are equally useful for selecting menu items.

Dominant-handed multi-stroke marking menus are faster than non-dominant-handed multi-stroke marking menus. Our results show for one-handed marking menus, the right hand outperforms the left hand in total time for both the iPod Touch and iGesture conditions, confirming hypothesis *H3*. All of our participants were right-handed, so we expected the right, dominant hand to be faster than the non-dominant hand. Surprisingly, we found no significant difference in accuracy between the right and left hand marking menus, suggesting that the only penalty for using the non-dominant hand is in speed.

Stroke direction affects performance on the iPod Touch. When drawing a single stroke on the iPod Touch, we found a significant difference between hands, with the dominant hand faster than the non-dominant hand. We also found that stroke direction had a significant effect on movement time. For each hand, participants were fastest at pulling their thumbs up or inward with respect to the hand, and on-axis strokes were faster than

the neighboring off-axis strokes. For the slower stroke directions, the participants may have been limited by the rotational constraints of the thumbs while holding the iPod Touch, but when the participants could freely move their hands on the iGesture we found no significant difference between hands nor any significant differences due to stroke direction.

Some stroke pairs are faster to draw than others. We also examined how different groupings of stroke directions affect drawing speed. For the 4-2 layout, pairs of both vertical strokes and both horizontal strokes tended to have faster average total times than the mixed pairs. In the two-handed simultaneous condition, drawing bilaterally symmetric strokes or strokes in the same direction was faster than drawing non-mirrored strokes. For the 8-2 layout, on-on and off-off pairs were faster than the mixed pairs for the two-handed simultaneous design. On-on strokes were slightly faster than off-off strokes on the iPod Touch.

Two-handed simultaneous exhibits more temporal overlap than two-handed ordered. For the two-handed conditions, the delay between the start of the first stroke and the second stroke was significantly different between the simultaneous and ordered conditions by about 200 ms. One direction for future work is to use this delay to distinguish which of these two menu techniques the user is performing.

4.6 User Study 2: Longitudinal Evaluation

In our first study, we found that the significant improvement in movement time for the two-handed simultaneous technique was offset by an increase in reaction time required to process the stimulus and plan the hand movements. However, with enough practice, users may require less time to plan movements as drawing two strokes at the same time becomes more automatic. We hypothesized:

H4: With practice, the reaction times for the 1HR and 2HS techniques converge.
H5: With practice, the 2HS technique outperforms the 1HR technique in total time.

To test these hypotheses, we conducted a longitudinal study with five right-handed participants (3 male, 2 female, between 24 and 32 years old) using the iPod Touch. We focused the longitudinal study on the iPod Touch as handheld multitouch devices are more ubiquitous than larger interactive surfaces like the iGesture. We gave each participant an iPod Touch to use over five consecutive days. Each participant spent approximately 45 minutes every day performing three blocks of 50 trials, using both the 1HR and 2HS menu techniques. The participants first performed the trials using the 4-2 layout and then the 4-4 layout. We used the same stimuli and feedback as in the first study. The participants received no practice trials and our analysis includes all data. As in the first study, our dependent variables were reaction time, movement time, total time, and accuracy. Misses were counted similarly, except the participants did not have to repeat missed sequences, as sequences were generated randomly.

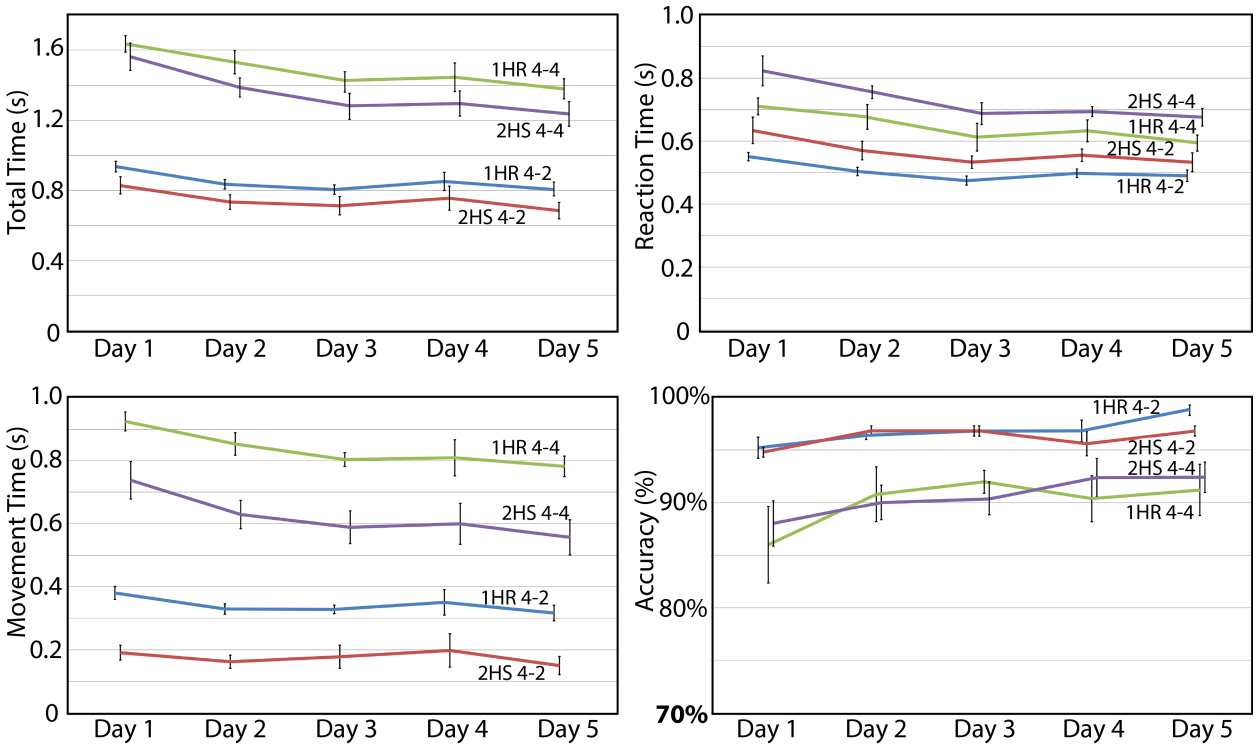


Figure 4.14: Average time and accuracy (with standard error bars) for menu techniques 1HR and 2HS and layouts 4-2 and 4-4, across five days. The baseline value of the accuracy graph is 70%.

4.6.1 Longitudinal Results

As shown in Figure 4.14, the total, reaction and movement times all decrease with practice while accuracy increases. Across both layouts 4-2 and 4-4, by day five, 2HS was 10-15% faster in total time and 29-52% faster in movement time than 1HR. However, reaction time remained 10-12% slower for 2HS than for 1HR, even after five days. Finally, on day five, accuracy was between 92.4-96.8% for 2HS and between 91.2-98.8% for 1HR.

For each layout (4-2, 4-4) and dependent variable (total time, movement time, reaction time, accuracy) we ran separate two-way repeated measures ANOVAs with day and menu technique (1HR, 2HS) as factors. We found significant main effects ($p<.05$) in all but the following cases: 1) for menu layout 4-2, day did not significantly affect movement time and menu technique did not significantly affect accuracy, 2) for menu layout 4-4, neither day nor menu technique significantly affected accuracy. We found no significant interactions in any of the ANOVAs. More detailed numerical results is as follows.

4-2 Menu Layout

Total Time: By day five, the average total time was 805 ms for 1HR and 682 ms for 2HS, reduced from 932 ms and 823 ms respectively on day one. Menu technique (p=.005) and day (p=.009) had significant effects on total time, but there was no interaction effect ($F_{4,16}$=0.308, p=.869).

Reaction Time: By day five, the average reaction time was 590 ms for 1HR and 533 ms for 2HS, reduced from 552 ms and 633 ms respectively. The difference between 1HR and 2HS reaction times dropped from 81 ms to 57 ms (29.6%). Menu technique (p=.012) and day (p<.001) had significant effects on reaction time, but there was no interaction effect ($F_{4,16}$=0.907, p=.483).

Movement Time: By day five, the average movement time was 315 ms for 1HR and 149 ms for 2HS, reduced from 380 ms and 190 ms respectively. The difference between 1HR and 2HS movement times dropped from 190 ms to 166 ms (12.6%). Menu technique (p=.001) had a significant effect on movement time, whereas day did not (p=.277). There was no interaction effect ($F_{4,16}$=1.263, p=.325).

Accuracy: By day five, the average accuracy was 98.8% for 1HR and 96.8% for 2HS, increased from 95.2% and 94.8% respectively on day one. The day had a significant effect on accuracy (p=.195) whereas the menu technique did not (p=.195). There was no interaction effect ($F_{4,16}$=1.758, p=.187).

4-4 Menu Layout

Total Time: By day five, the average total time was 1376 ms for 1HR and 1234 ms for 2HS, reduced from 1635 ms and 1559 ms respectively on day one. Menu technique (p=.008) and day (p<.001) had significant effects on total time, but there was no interaction effect ($F_{4,16}$=0.820, p=.531).

Reaction Time: By day five, the average reaction time was 594 ms for 1HR and 676 ms for 2HS, reduced from 711 ms and 822 ms respectively. The difference between 1HR and 2HS reaction times dropped from 111 ms to 82 ms (26.2%). Menu technique (p=.005) and day (p<.001) had significant effects on reaction time, but there was no interaction effect ($F_{4,16}$=0.484, p=.748).

Movement Time: By day five, the average movement time was 782 ms for 1HR and 557 ms for 2HS, reduced from 924 ms and 737 ms respectively. The difference between 1HR and 2HS movement times dropped from 225 ms to 187 ms (16.9%). Menu technique (p=.001) and day (p<.007) had significant effects on movement time, but there was no interaction effect ($F_{4,16}$=0.389, p=.814).

Accuracy: By day five, the average accuracy was 91.2% for 1HR and 92.4% for 2HS, increased from 86.0% and 88.0% respectively on day one. Neither menu technique (p=.774) nor day (p=.084) had a significant effect on accuracy. There was no interaction effect ($F_{4,16}$=0.544, p=.706).

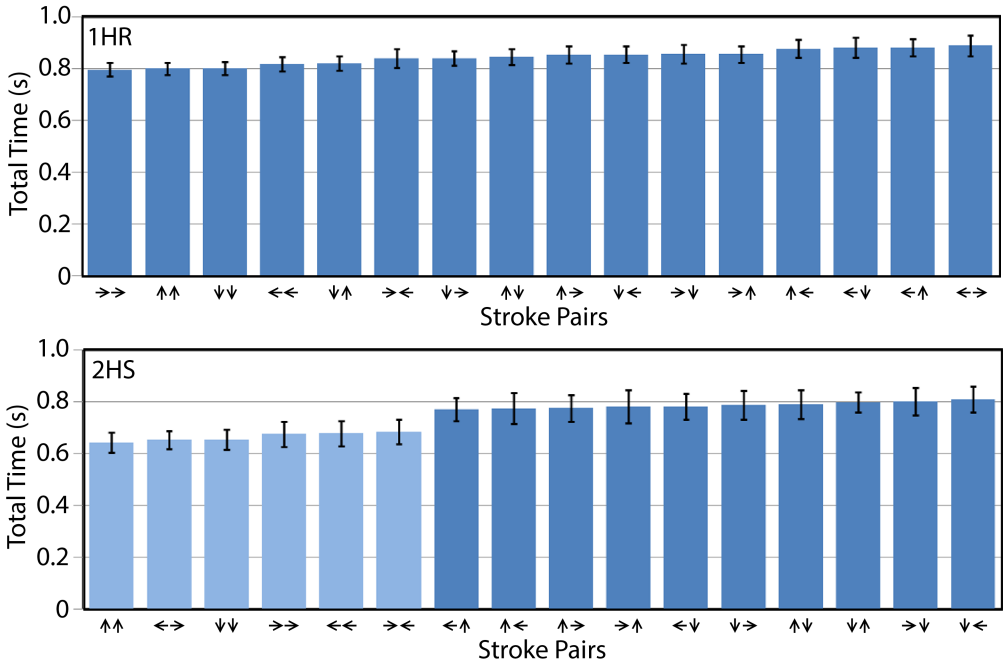


Figure 4.15: Average total time (with standard error bars) per stroke pair for the 4-2 layout. For 2HS, pairs of strokes that are bilaterally symmetric or share the same direction (light blue) are 18% faster to draw than the other pairs.

4.6.2 Longitudinal Discussion

Time and Accuracy: While we cannot accept our hypothesis (*H4*) that reaction times would converge, we found that the difference in reaction times between 1HR and 2HS did decrease significantly after five days. The decrease was 29.6% for the 4-2 layout and 26.2% for the 4-4 layout, suggesting that time required to coordinate two hand movements diminishes with practice. For the 4-2 layout there was no significant difference in movement time across days. For the 4-4 layout reaction and movement time improved for both techniques, but the movement time advantage for 2HS outweighed the reaction time advantage for 1HR. Menu technique had a significant effect on total time, and by day five, total time was faster for 2HS than 1HR by 15.3% for layout 4-2 and 10.3% for layout 4-4, confirming hypothesis *H5*. In our initial study 2HS had a relatively low accuracy rate for the 4-4 layout (89.5%). Although we did not find the day to have a significant effect on accuracy, the average accuracy did improve from 88.0% to 92.4% after five days and there was no significant difference between the two techniques. Together these results suggest that although 2HS may be more difficult to use than 1HR at first, with moderate practice an expert user can access menu items more quickly using 2HS than 1HR, while maintaining reasonably good accuracy.

Stroke Directions: Figure 4.15 shows the total time in sorted order for each of the 16 stroke pairs in the 1HR condition (top) and the 2HS condition (bottom). For 2HS we find

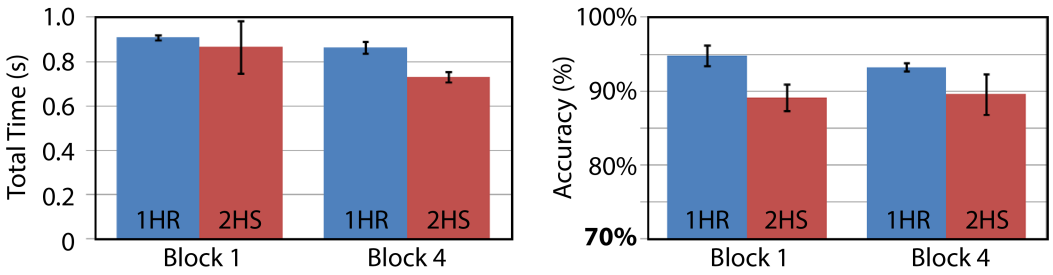


Figure 4.16: Average total time and accuracy (with standard error bars) for the first and last blocks for the 8-2 layout. The baseline value of the accuracy graph is 70%.

that drawing strokes that are bilaterally symmetric or in the same direction (translationally symmetric), is 18% faster on average than drawing the remaining stroke pairs. For 1HR, we find no large difference in total time between groups of stroke pairs, but the pairs in which both strokes are drawn in the same direction are the four fastest.

Our 2HS results indicate that users are most efficient at drawing symmetric strokes and are consistent with prior studies of two-handed motion control [65, 87]. Together these studies suggest that people find it more difficult to draw pairs of asymmetric strokes than symmetric strokes. However, the stimuli for the task may also affect performance. Balakrishnan and Hinckley [11] and Franz et al. [38] have shown that users perform two-handed tasks more efficiently when the stimuli are seen as representing a single task rather than two independent tasks. We leave it to future work to investigate how the visual design of the stimuli might affect performance.

4.6.3 Longitudinal Study: 8-2 Menu Layout

Based on the results of our longitudinal study on the breadth-4 menus, we conducted a small follow-up experiment with three right-handed participants (2 male, 1 female, between 24 and 25 years old) to examine long term performance for the breadth-8 layout using menu techniques 2HS and 1HR. Each participant performed four blocks of trials, where each block consisted of three repetitions of the 64 stroke pairs in randomized order for each menu technique. Again, the participants received no practice trials and our analysis includes all data.

As shown in Figure 4.16, by the fourth block the total time was 15.4% faster with 2HS than with 1HR and average accuracies were 89.6% for 2HS and 93.2% for 1HR. The speedup was in line with our breadth-4 results. Moreover, the 2HS accuracy of 89.6% was much higher than the 82.3% we saw in our initial study of the 8-2 layout, suggesting that practice can improve accuracy. Although 2HS was not quite as accurate as 1HR, examining the individual stroke pairs for 2HS (Figure 4.17), we found that when the left stroke was parallel to the SW-NE axis or the right stroke was parallel to the SE-NW axis accuracy (highlighted in orange) dropped to 86.6%, while the remaining 36 stroke pairs maintained an accuracy

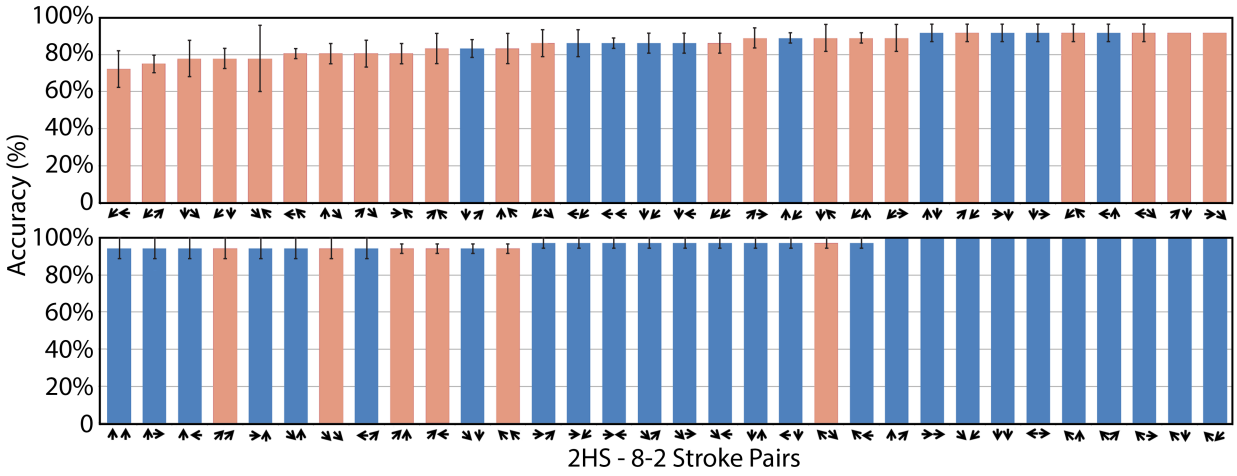


Figure 4.17: Average accuracy (with standard error bars) for all stroke pairs in the 8-2 layout using the 2HS technique. The pairs where the left stroke is parallel to the SW-NE axis or the right stroke is parallel to the SE-NW axis are highlighted in orange. Note that standard error bars have zero radius when all three participants had the same accuracy.

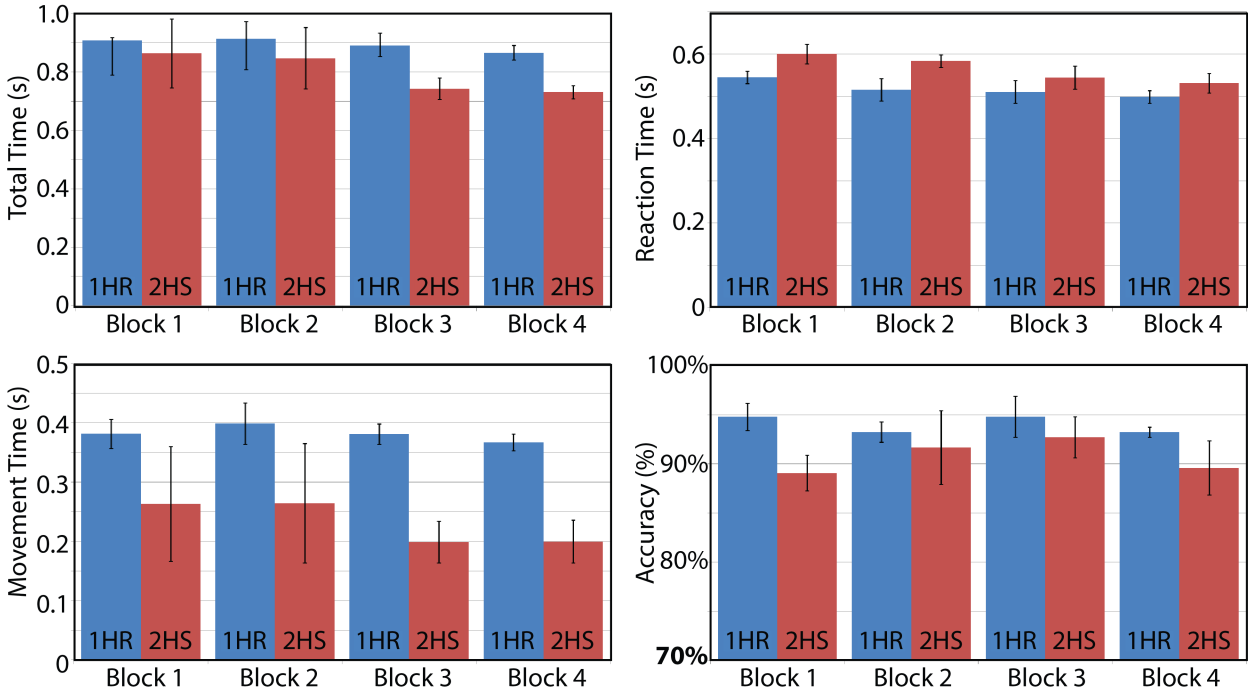


Figure 4.18: Average times and accuracy (with standard error bars) for menu techniques 1HR and 2HS and menu layout 8-2, across four blocks of trials. The accuracy graph's baseline value is 70%.

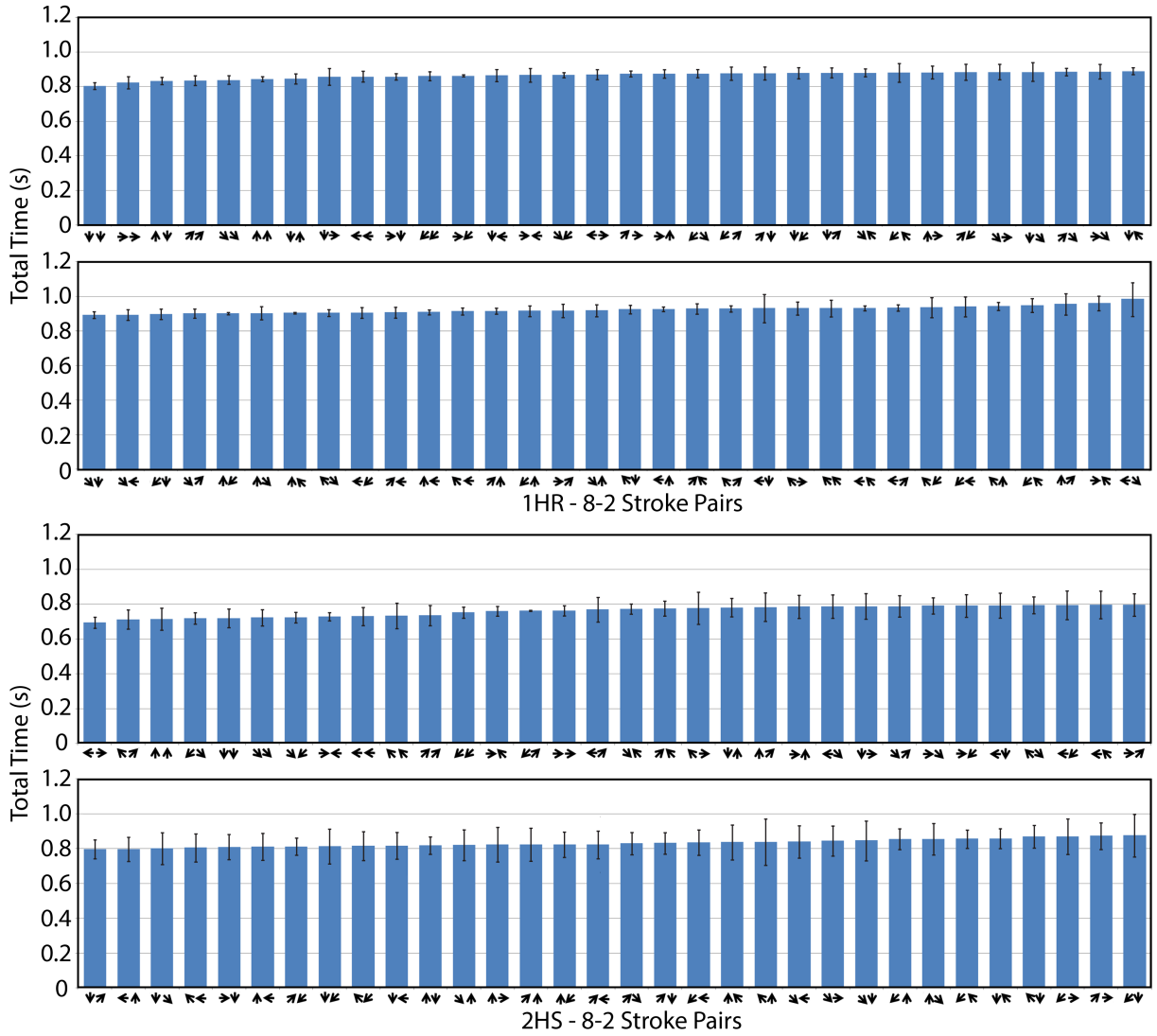


Figure 4.19: Average total time (with standard error bars) for all stroke pairs in the 8-2 layout using the 1HR and 2HS technique.

at 95.0%. Our results confirm Karlson et al.'s [63] observation that for the right thumb, SE-NW strokes are the most difficult to draw. Just as for the breadth-4 layout, we found that pairs of bilaterally symmetric or same-direction strokes (731 ms) were faster to draw than other pairs (815 ms).

We present more detailed numerical results in Figure 4.18. Complete stroke pairs total time and accuracy are presented in Figure 4.19 and Figure 4.20 respectively.

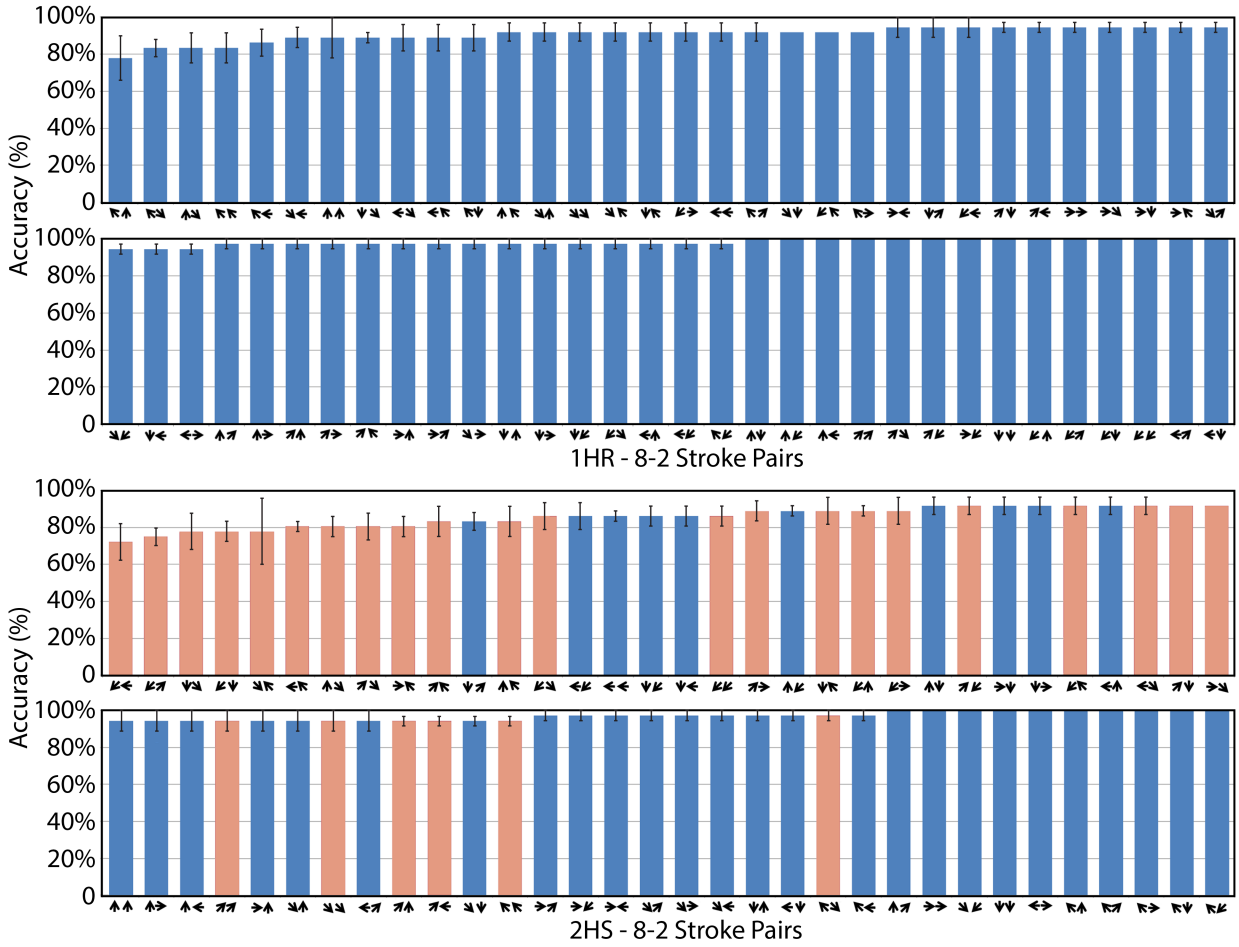


Figure 4.20: Average accuracy for all stroke pairs in the 8-2 layout using the 1HR and 2HS technique. For 2HS, the pairs where the left stroke is parallel to the SW-NE axis or the right stroke is parallel to the SE-NW axis are highlighted in orange. Note that standard error bars have zero radius when all three participants had the same accuracy.

4.7 Design Guidelines

Based on our results we make the following design recommendations. Two-handed simultaneous multi-stroke marking menus provide the fastest performance with good accuracy at breadth-4 and acceptable accuracy at breadth-8. The most frequently used commands should be bound to pairs of bilaterally symmetric strokes or same-direction strokes as they are the fastest to perform. For handheld thumb-operated devices it may be possible to improve breadth-8 accuracy by avoiding the use of the SW-NE directions for the left thumb and the SE-NW directions for the right thumb. Alternatively, designers may bind commands that require conscious commitment, such as deletion or quitting, to these pairs that are harder to perform.

With a one-handed marking menu operated by a thumb, the right thumb is faster than the left thumb. Within the right-handed menu the most frequently used menu items should be placed in the upper right quadrant of directions, and within the left-handed menu, the most frequently used menu items should be placed in the upper left quadrant, as those directions had the fastest movement times.

4.8 Display Menu Items for Novice Users

Although our studies focused on expert performance, real-world usage of our two-handed designs requires methods for training novice users in the mapping between stroke pairs and menu items. We offer two novice-mode visualizations that facilitate exploration of the menu items bound to stroke pairs.

4.8.1 Hierarchical Display Menu

In a hierarchical display (Figure 4.21 Left), the left hand selects the parent menu item by dialing through the items [149]. The right menu continuously updates to show the corresponding child menu items. The closest parent and child menu items are always highlighted to indicate which option the user has currently chosen. Users can continuously explore all possible menu items without backtracking or lifting up any fingers. However, navigating a four-stroke menu would still require backtracking if the wrong initial pair was selected.

4.8.2 Full-Breadth Display Menu

In a full-breadth display (Figure 4.21 Right), all the items are presented [147], and the left hand specifies the menu cluster, while the right hand specifies the item within a cluster. The cluster that the left hand is currently selecting is highlighted as feedback to the user. The user can dynamically switch to any of the options without lifting up any fingers. This display can be particularly useful when there is no logical way to group items into equally-sized clusters.

4.9 Applications

We have shown that our two-handed multi-stroke marking menus are viable menu selection techniques. Although many iPhone/iPod Touch applications and games use two-thumb controls, we have not yet seen two-thumb marking menus on these devices. We believe many such handheld applications could benefit from our two-thumb marking menus and we present several usage scenarios.

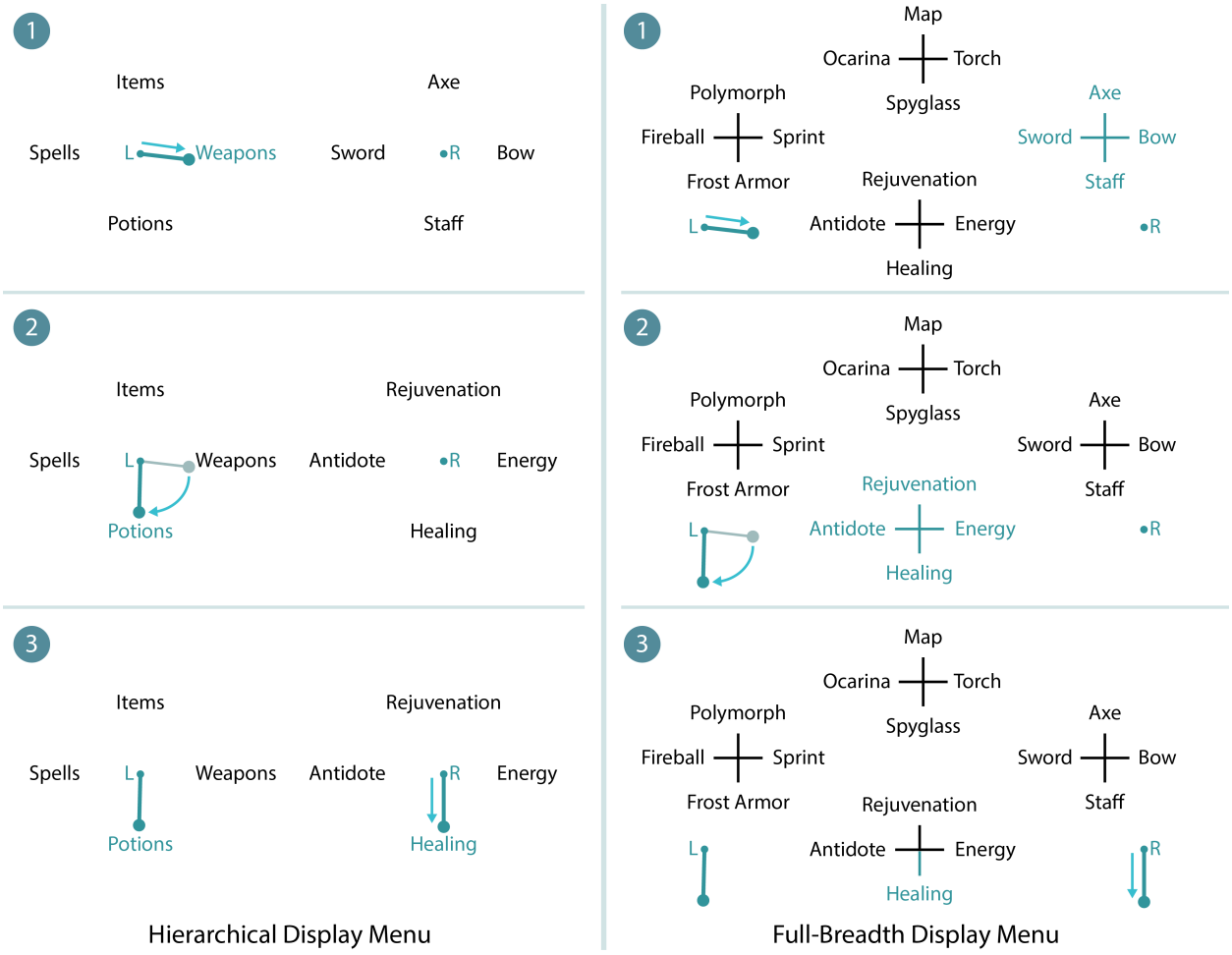


Figure 4.21: Left: Hierarchical Display – The left hand explores the parent menu items by dragging through menu items, and the child menu items continuously update for the right hand. Right: Full-Breadth Display – The entire menu space is displayed, and the left hand chooses the four-item cluster, while the right hand chooses the item within a cluster.

4.9.1 Dual-Joystick Navigation

Many iPhone games require players to hold the device in landscape mode and use two virtual joysticks operated by the thumbs, one to control movement and the other to control character orientation. Our two-thumb techniques could be integrated into such games and used to trigger commands such as diffusing or disposing of mines in a mine disposal game (Figure 4.22). For example, we could integrate our marking menus by overlaying them on the joysticks. In this case, users would need to use some distinguishing gesture to indicate whether they wished to invoke the marking menus or use the joysticks. For instance, quick strokes might indicate menu selection while longer strokes might indicate joystick-based

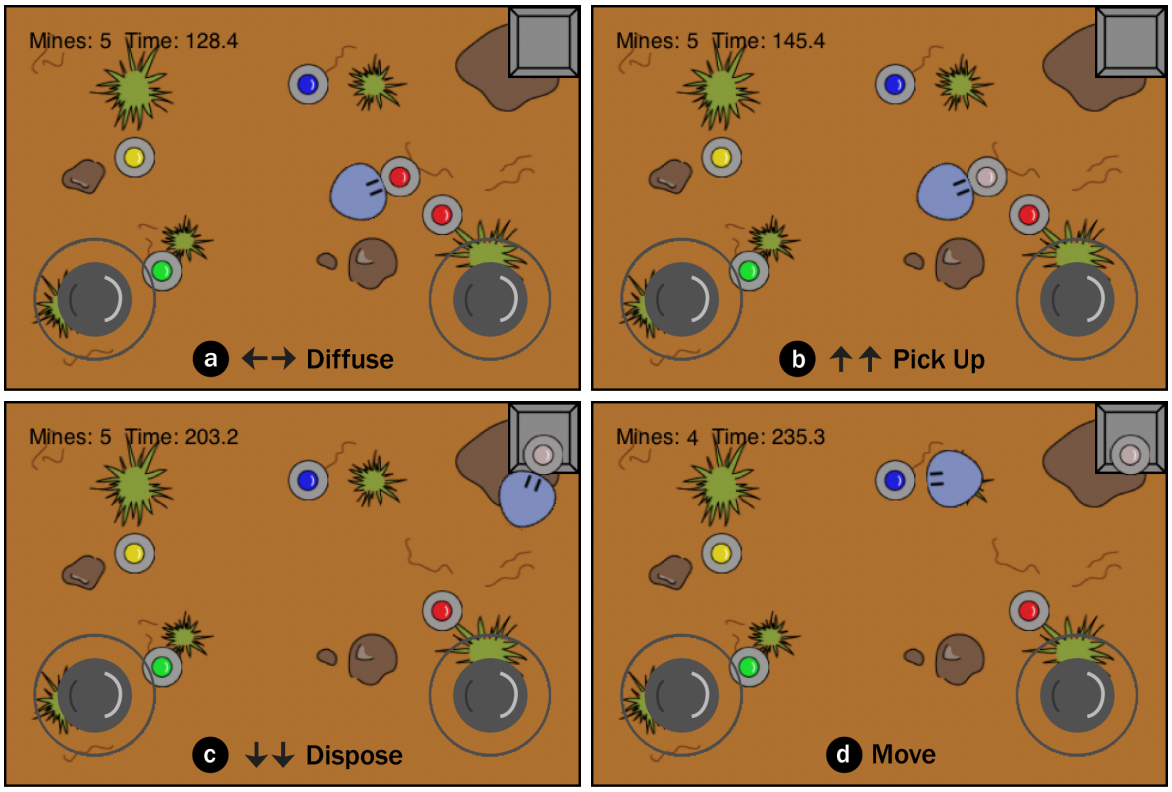


Figure 4.22: In this mine disposal game, the user moves a robot using two joysticks. The left joystick controls movement and the right joystick controls orientation. Two-handed marking menus invoke commands as shown in the four screenshots and can be executed anywhere on the screen.

movements. Another option would be to interpret all strokes that lie outside the joystick regions as menu commands. However, this approach might require users to repeatedly move their hands on and off the joystick regions. We utilize the first approach and interpret strokes under 0.25 seconds in duration as menu commands. Because our techniques are eyes-free, expert users can keep their focus on the current task and select commands without visually searching for soft buttons to select a command. Examining data from a small, informal sample of two members of our lab, we have found that expert users can switch from joystick-based movement to two-handed menu selection and back to movement in less than 0.5 seconds.

4.9.2 Text Editing

In text-editing mode on touchscreen devices such as the iPhone, the keyboard takes up half the screen, leaving little room for text formatting options. We have integrated our two-handed marking menus into a text-editing application allowing users to quickly change the

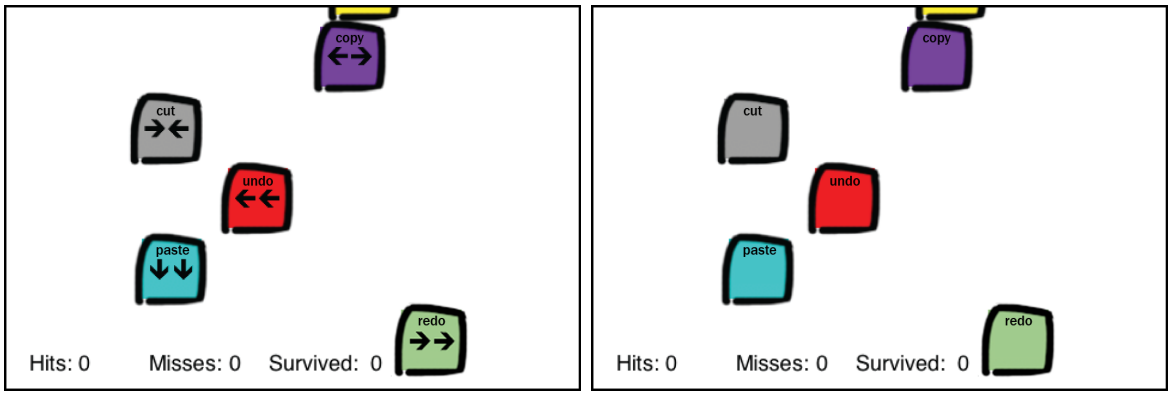


Figure 4.23: In the Falling Blocks game, the user must destroy each block by drawing the corresponding strokes. Left: Novice Mode – Strokes are shown. Right: Expert Mode – No strokes are shown.

attributes of the text (Figure 4.1) by selecting from a two-handed marking menu located above the keyboard, rather than through soft buttons.

4.9.3 Falling Blocks Game for Training Novice Users

Touch-typing is a complex task that requires coordination of ten fingers. Novice typists often use typing games to become proficient. Similarly, we offer a *Falling Blocks* game to train novices to quickly draw two directional strokes simultaneously to execute commonly used system commands. In the game, colored blocks continually fall down the screen, and each block is associated with a command and a stroke pair. Users must execute the correct strokes to destroy each block before it falls to the bottom of the screen. In novice mode, the strokes are shown to the user, but in expert mode, the user must remember the mapping between commands and strokes (Figure 4.23).

The iPhone App Store provides an opportunity for researchers to introduce applications to the general public. As an experiment, we released *Block Blender*, a variant of our *Falling Blocks* application, on the App Store. In the game, blocks continuously fall down the screen and the player must draw the specified pair of strokes to destroy the block. When the stack of blocks reaches the top of the screen, the game is over. We had nearly 1000 downloads after the first seven weeks of deployment. Users could opt to send scores and data from their completed games back to our servers. Of the scores we received, two players destroyed at least 1000 blocks with 93.7% and 85.1% accuracy. Six players were able to outscore the first author's best score of 184, each doing so in six tries or less. These players were able to stay alive with a new block generated every 0.82 seconds while performing pairs of strokes with average movement times between 0.13 second and 0.29 seconds. Although this application demonstrates that some players can quickly pick up the skill to draw pairs of strokes simultaneously and outperform even the first author, most players did not produce such high scores. These players may still be learning the mechanics of the game.

4.10 Conclusion

Along with our multitarget selection study, we have shown the value of bimanual interaction when using multitouch for target selection and drawing directional strokes. Two hands can divide and conquer the interaction space while saving time by performing movements in parallel. We believe techniques like two-handed marking menus that leverage bimanual interaction are a step towards exploiting the full potential of multitouch devices. In the next chapter, we present a multitouch application for building virtual environments for computer-animated films, where we often utilize bimanual interaction in the design of the gestures.

Chapter 5

Designing a Professional Multitouch Application

The advantages of multitouch input over mouse input include the benefits of direct-touch selection and bimanual interaction, which we have demonstrated in our user studies. Developers should leverage these benefits when designing a multitouch gesture. Developers should also ensure that the user can comfortably coordinate multiple fingers in a multifinger gesture. Although a single gesture might be relatively easy to design, it is only one of many gestures in a fully functional multitouch application. Thus, building an entire gesture set also requires careful consideration to create distinguishable gestures that are both easy to remember and easy to perform.

We investigate the design and development of a gesture set for a professional-level multitouch application. While most multitouch applications are designed for mobile devices and content browsing, we focus on using multitouch for creating content currently done on a traditional desktop workstation with a mouse and keyboard. In particular, we examine whether the benefits of multitouch input can improve the task of the construction of virtual organic sets for computer-animated feature-length films.

The production of computer-animated films, such as Pixar's *Toy Story* and DreamWorks' *How to Train Your Dragon*, consists of many distinct stages, commonly referred to as the production pipeline. Set construction is one of these stages. Similar to a physical set for live-action films, a virtual set is the environment in which animated films are shot. Set construction artists select and position geometric models of objects, such as furniture and props to build manmade environments, and vegetation to build organic environments.

Today, many animation studios use off-the-shelf modeling and animation packages (e.g., Maya, 3ds Max) for set construction (Figure 5.1). Despite more than a decade of interface refinement, the process required to build a set using these mouse and keyboard interfaces is long and tedious. A set construction artist commonly places hundreds if not thousands of 3D objects in the set, but is usually limited to placing one object at a time. Moreover, to properly place a single object in 3D space, the artist often performs several individual 3D manipulations, such as translation, rotation, and scale. However, the mouse only has two

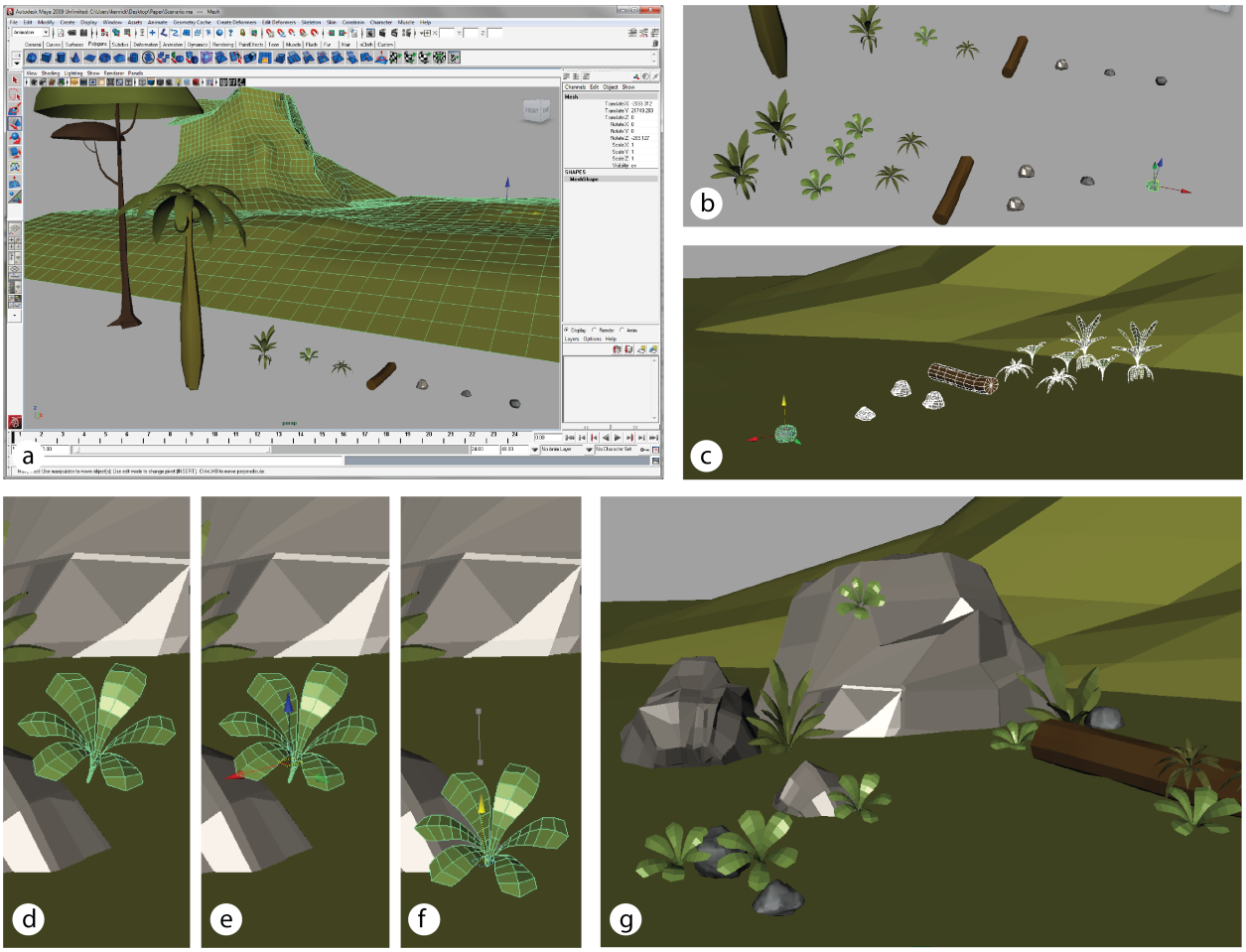


Figure 5.1: Constructing a set with Maya: (a) The set construction artist creates a model catalog by lining up the models he plans on using away from the terrain. (b-c) He then makes duplicates of the objects and translates them to the region of the terrain where he is constructing the set. (d-f) To translate an object, he first selects the object, then switches to translation mode with a hotkey, and finally picks and drags the arrow manipulator. (g) He translates, rotates, and scales objects one by one until he completes the set.

degrees of freedom, so the artist cannot manipulate more than two spatial parameters of the object at a time. In addition, existing interfaces introduce significant overhead: the artist must manage modes, select small manipulators, and traverse long distances with the mouse.

We investigate whether *Eden*, a new organic set construction application that leverages multitouch input, can address these concerns. With two hands, the artist can work in two different parts of the screen at the same time, thereby reducing the need to travel back and forth between spatially distant screen regions. The artist may also become more efficient by performing simultaneous operations, one with each hand. Furthermore, multitouch work-

stations can sense the position of each finger and thus two hands provide many degrees of freedom of input. Multitouch interfaces can use these many degrees of freedom to allow users to specify both target object and operation, while manipulating more than just two of the object's spatial parameters at a time. As a result, the application can reduce the number of modes and the number of individual steps needed to complete the placement of a single object.

Despite these advantages, building a multitouch application presents design challenges, such as choosing gestures that are efficient, memorable, and comfortable to perform. There are many different ways to map multitouch sensor data to operations, and the best gesture for a given task is often not obvious. An application might also require a large set of operations, and in order for the application to unambiguously interpret the user's actions, no two gestures can be the same. Finally, touch input has several drawbacks that can reduce the user's efficiency, including imprecision due to the fat finger problem [110] and occlusion of content by the hands [127].

To address these challenges, we built Eden while working in close collaboration with Tom Miller (TM), a veteran set construction artist at Pixar Animation Studios. We relied on his feedback and experience to create a set construction application suitable for professional-level use. From our design process, we found that restricting Eden to support one operation at a time allowed us to design simple, easy to learn gestures that split the workload across two hands. Using Eden, TM has built a set for a feature film and found the system to be more efficient and more pleasant than Maya.

5.1 Organic Set Construction

Set construction is the process of selecting and positioning virtual objects to build a virtual environment inhabited by characters of a computer-animated film. Before building a set, the set construction artist first works with the story and art departments to determine the aesthetics and rough layout of the set. Then the set construction artist works with the layout department, which is responsible for placing the foundation of the set by positioning the terrain and any key architectural elements that help dictate the action in the scene. The set construction artist then populates the sets with the geometric objects built by the modeling department to flesh out the world. The layout department also provides the set construction artist with shot cameras, which are used to make the final renders. Using the shot cameras, the set construction artist "constructs to camera," to avoid building sections of the set that will not be seen in the final film. Throughout this process, the set construction artist continues to work iteratively with the story, art, and layout departments to finalize the set. Once the director approves the set, it is then sent to the animation department.

To gain a better understanding of the set construction process, we observed TM, who has over 10 years of set construction experience at Pixar Animation Studios. TM specializes in building organic sets, such as forests, parks, and other outdoor environments consisting primarily of vegetation (Figure 5.2).

Figure 5.2: An organic set in Pixar's *Up*. Copyright Disney/Pixar.

To build a set, TM traditionally uses Autodesk Maya [7], a 3D modeling and animation package. His workflow, whether for manmade or organic sets, typically proceeds as follows (Figure 5.1). First, TM loads the objects he plans to use for a set and lines them up in a location away from the terrain. These objects serve as his *model catalog*. To add objects to the set, he duplicates them in the model catalog area and moves them into the region of the set he is working on. Then using the Maya translation, rotation, and scale manipulators, he positions and orients each object into place. To translate an object, for example, he selects the object, hits the 'W' hotkey to enter translation mode, and picks the appropriate arrows on the translation manipulator (Figure 5.1e,f) to drag the object into position. He can inspect the set by using the default Maya camera controls: while holding the 'alt' key, a left mouse button drag performs arcball rotation, a middle mouse button drag translates the camera along the view plane (truck and pedestal), and a right mouse button drag moves the camera forward and back (dolly). He also uses the shot cameras to construct to camera. He repeats this process, working region by region, until he completes the set.

Our original intent was to build a multitouch application for general set construction. However, we found that the imprecision of touch makes the construction of manmade sets particularly difficult. Manmade environments are often structured and rigid. They contain highly regularized elements like furniture arrangements, books on a shelf, or city streets. The positions and orientations of objects often depend precisely on the positions and orientations of other objects. Placing these objects requires precision and fine-tuning, which

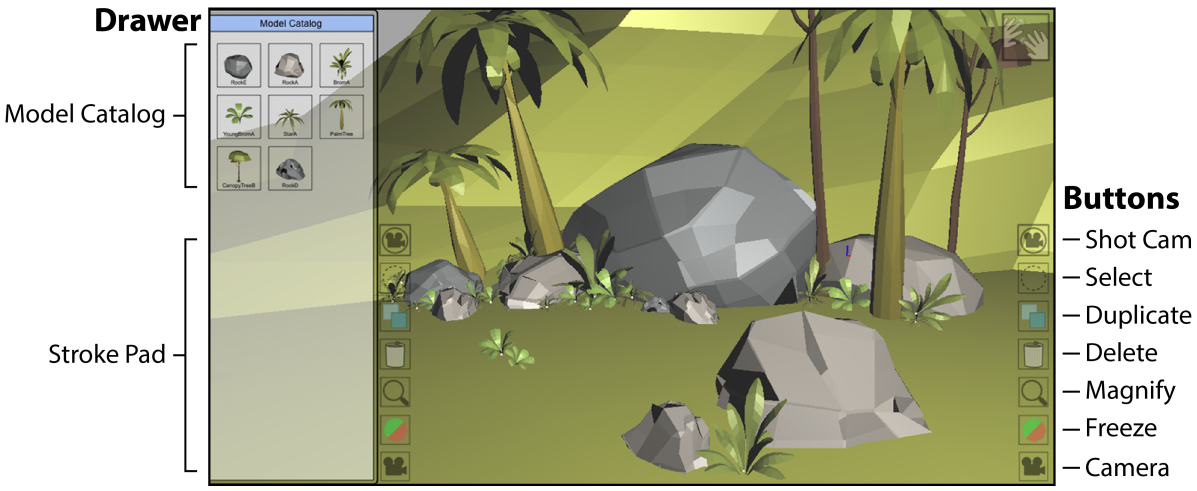


Figure 5.3: The interface of Eden consists of the main content view, a drawer containing the model catalog and stroke pad overlaid on the left, and two matching columns of buttons.

is problematic as touch is imprecise and the artist's hands can obscure the content being manipulated.

Instead, we chose to first target organic set construction, since it is less affected by precision issues. According to TM, he is less concerned about precision when constructing organic sets because he places vegetation coarsely compared to manmade objects. In addition, he often places a large amount of vegetation in an organic set, so he can frequently make use of the fast coarse targeting of direct-touch [37, 69, 120] to indicate the positions of vegetation. We can then apply the experience we gain from designing a multitouch application for organic set construction to the more involved task of designing a multitouch application for building general sets.

5.2 Eden

The interface of Eden (Figure 5.3), our multitouch set construction application, is composed of a main view, a virtual drawer, and two columns of buttons. The main view presents the scene through a perspective camera and the artist can directly manipulate objects through this view. We designed the main view to take up virtually the entire screen to help keep the artist's focus on the content. On the left side of the interface is the *drawer*, which houses the model catalog and the stroke pad. The model catalog consists of objects available to the artist for a given session. On the stroke pad, the artist can draw single-stroke symbols that execute infrequent commands. If the artist wants to maximize the content area, he can slide the drawer closed. In addition, we provide two matching columns of buttons that map to additional set construction commands. We repeat the buttons on both sides of the interface to allow either hand to invoke them.

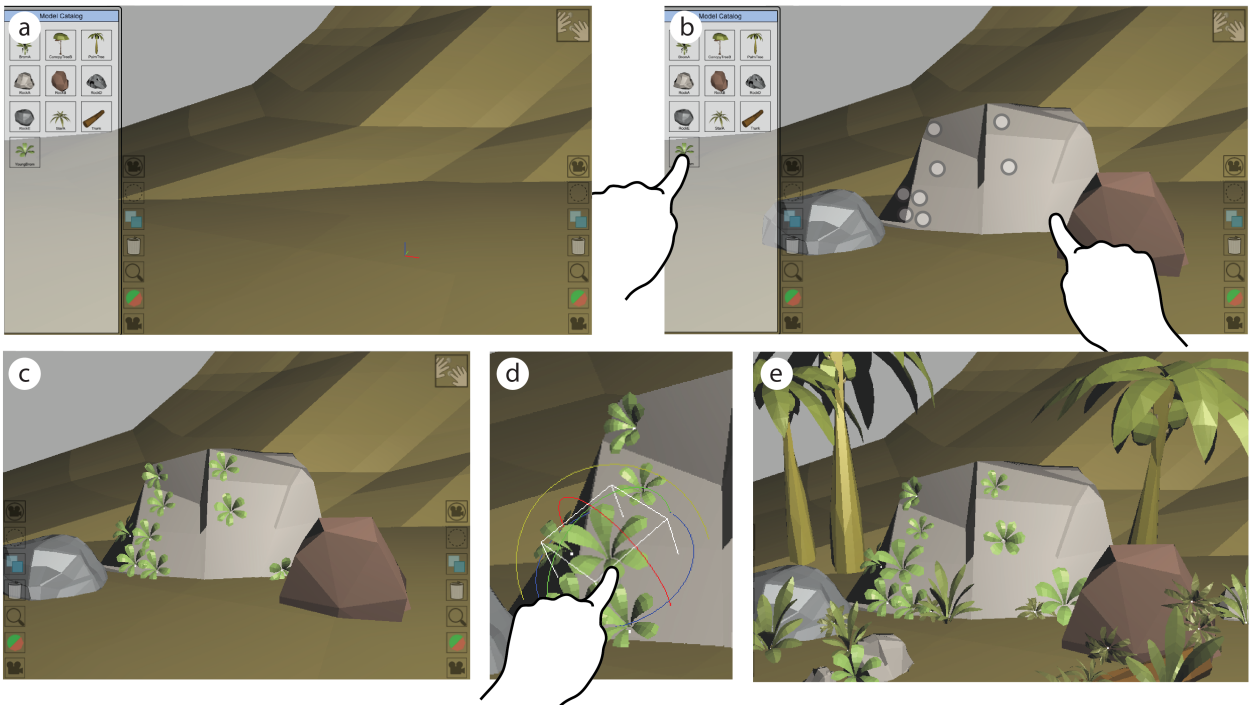


Figure 5.4: Constructing a set with Eden. (a) The set construction artist starts with the empty terrain. (b-c) Using the model catalog in the drawer, the artist can touch one finger on the model, and with a second hand touch the locations for where to place copies of the model. He taps several times on the boulder to quickly add nine bromeliads. (d) He makes additional adjustments to each bromeliad by performing an arcball rotation for example. (e) He continues adding and manipulating objects until the set is complete.

TM's process for building a set with Eden typically proceeds as follows (Figure 5.4): TM starts a new session by loading the terrain and key architectural elements provided by the layout department into the set. He then creates a model catalog by drawing an 'L' in the stroke pad to open a panel, from which he chooses the geometric objects he wants to add to the model catalog. After building the catalog, he adds objects into the set. He might touch a tree in the model catalog and make multiple taps on the terrain to indicate the locations at which to plant each tree. If he is dissatisfied with how a tree looks he can translate, rotate, or scale the tree by performing the corresponding gesture; we describe the object manipulation gestures in Section 5.2.2. In addition to using the default camera to inspect the quality of the set, TM also loads in shot cameras via a stroke command so he can construct to camera by checking the quality of the set through the shot cameras' views. TM continues to place objects and adjust them until he is satisfied with the set.

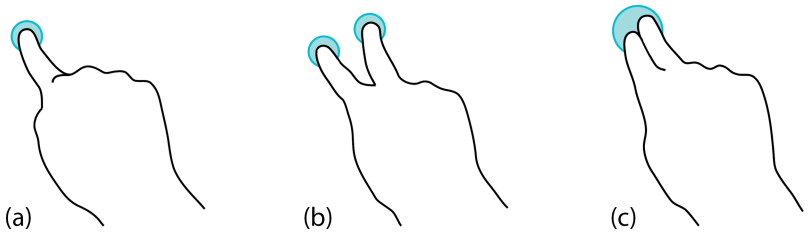


Figure 5.5: (a) One-touch using a single finger. (b) Two one-touches using two fingers. (c) Conjoined touch using two fingers next to each other.

5.2.1 Design Principles

Eden supports a variety of object manipulation and camera control operations. Our challenge is to design gestures that map the many degrees of freedom of multitouch input to these operations and their parameters. Working with continuous feedback from TM, we went through several iterations in the design of the gesture set for Eden. We evaluated which aspects of the gestures TM found effective and identified the following design principles for the creation of our final gesture set.

- **Use simple gestures for frequently used operations** – Gestures that require fewer touches and fewer movements require less coordination and are faster to perform. We bind such simple gestures to the more frequently used operations to increase overall efficiency.

- **Conjoined touch as a modifier** – To increase the size of the gesture space while keeping gestures simple, we introduce the *conjoined touch* into our gestures. A one-touch is a standard touch where a single finger touches the screen and yields a single 2D contact point. We detect a conjoined touch whenever two touches are adjacent to each other. Specifically, the two touches are combined into a single instance of a conjoined touch where the centroid of the two touches serves as the 2D contact point for the conjoined touch. Thus, two fingers on the same hand can represent three static states: one-touch, a pair of one-touches, and a conjoined touch (Figure 5.5). We can use a conjoined touch instead of a one-touch to differentiate two operations similar in function, while maintaining the same underlying motion of the hands.

- **One operation at a time** – We initially designed one-handed gestures for object manipulation so the artist could perform two operations simultaneously, one with each hand. However, we found that TM concentrates on manipulating a single object at a time and seldom requires the ability to manipulate two objects at a time. According to Raskin [112], a person only has a single locus of attention, and thus can only focus on the position of one object at time, making the simultaneous manipulation of two objects mentally difficult. Moreover, allowing only one operation at a time reduces the ambiguity of interpreting touch input. For instance, if we had permitted simultaneous

gestures, then the application could interpret two touches as either two simultaneous one-touch gestures or a single gesture that uses two touches.

- **Split touches across both hands** – Since we only support one manipulation at a time, when a gesture uses more than one touch, we split the touches of the single gesture across both hands for two reasons. First, fingers on separate hands are not constrained by the palm, which makes them more mobile than fingers on the same hand. This increased mobility makes performing complex motions easier and more comfortable. Second, assigning touches to a second hand can reduce the amount of occlusion of the object being manipulated as the second hand can perform movements in an indirect fashion away from the object.

- **Use at most two fingers from each hand** – Although a single hand supports up to five touches, anatomical constraints of the hand limits the flexibility of each touch. For example, the middle and ring fingers on the same hand cannot move arbitrarily far apart. In Section 3.2, we found that participants rarely used fingers other than the two middle and index fingers on each hand. The more fingers a gesture requires, the more complicated and uncomfortable the gesture can become. Therefore, we designed gestures that limited the number of fingers used to at most two per hand.

- **Interchangeability of hands** – For bimanual interaction, Guiard assigns fixed roles to the hands in his Kinematic Chain Model [45]: the non-dominant hand sets the frame of reference while the dominant hand performs the primary action. We, however, permit the artist to begin an operation with either hand. Since an object can be located anywhere on the screen, interchangeability of the hands allows the artist to choose the most convenient hand to manipulate an object.

- **Motion of gesture reflects the operation** – If the motion of the gesture is similar to the effect of the operation, then the artist can more easily guess how the gesture will affect the target object. Also, the association between motion and operation can help the artist recall gestures. For example, the motion of the popular two-finger pinch gesture for photo resizing suggests the corresponding operation: reducing the distance between the two fingers reduces the photo size, while increasing that distance increases the photo size.

- **Combine direct and indirect manipulation** – Direct-touch combined with direct manipulation allows the artist to touch and manipulate a virtual object as one might in the physical world [113]. A drawback, however, is the hands occlude the object they are manipulating. Using indirect manipulation, the artist can perform movements away from the target object, so the hands occlude less of the object.

- **Control at most two spatial parameters at a time** – To properly place an object within a set, the artist must be able to adjust the object's position, orientation, and size. We had intended to design gestures that allow an artist to manipulate more

than two spatial parameters of an object at a time. However, TM prefers having more individual control of these spatial parameters, so each of our gestures controls just one or two spatial parameters of an object. Research has also shown that even with a six degree of freedom input device, users perform translation and rotation separately [85].

5.2.2 Object Manipulation

Eden supports eight operations for object manipulation that utilize two types of touches: one-touch and conjoined touch. In Eden, the world is oriented such that the x and y axes correspond to the horizontal ground plane, and the z-axis corresponds to the up direction. As shown in Figure 5.6, the object manipulation operations and gestures consist of:

- **x-y translation** – a conjoined touch on the object, then drag
- **z translation** – a conjoined touch on the object, together with a one-touch drag up and down
- **arcball rotation** – a one-touch on the object, then drag
- **local z rotation** – a one-touch on the object, together with a second one-touch drag left and right
- **world z rotation** – a one-touch on the object, together with a conjoined touch drag left and right
- **uniform scale** – a one-touch on the object, together with a two-touch pinch
- **one-dimensional scale** – a one-touch on the object, together with a conjoined touch drag on the bounding box face perpendicular to the local scaling axis
- **throw-and-catch** – a one-touch on the object, and a second one-touch tap at another location

We tailored the set of operations for organic set construction to support the operations most useful to TM. A set construction artist typically uses separate x and y translations to carefully align manmade objects with each other. For organic objects, however, TM finds controlling both of these translational degrees of freedom at the same time to be more efficient. Thus, we support simultaneous x-y translation, instead of separate x and y translations. We also provide a separate z translation to give TM full 3D positional control.

In addition to positioning each object, TM also adds variation to each object by rotating and scaling it. For example, he can build a grove of oak trees replicating just one oak tree model, and rotate and scale each copy to make it appear different from the other trees. TM needs just enough rotational control to tilt each object off the world z-axis and spin it about its local z-axis to make the object appear unique. Therefore, arcball rotation and z rotation

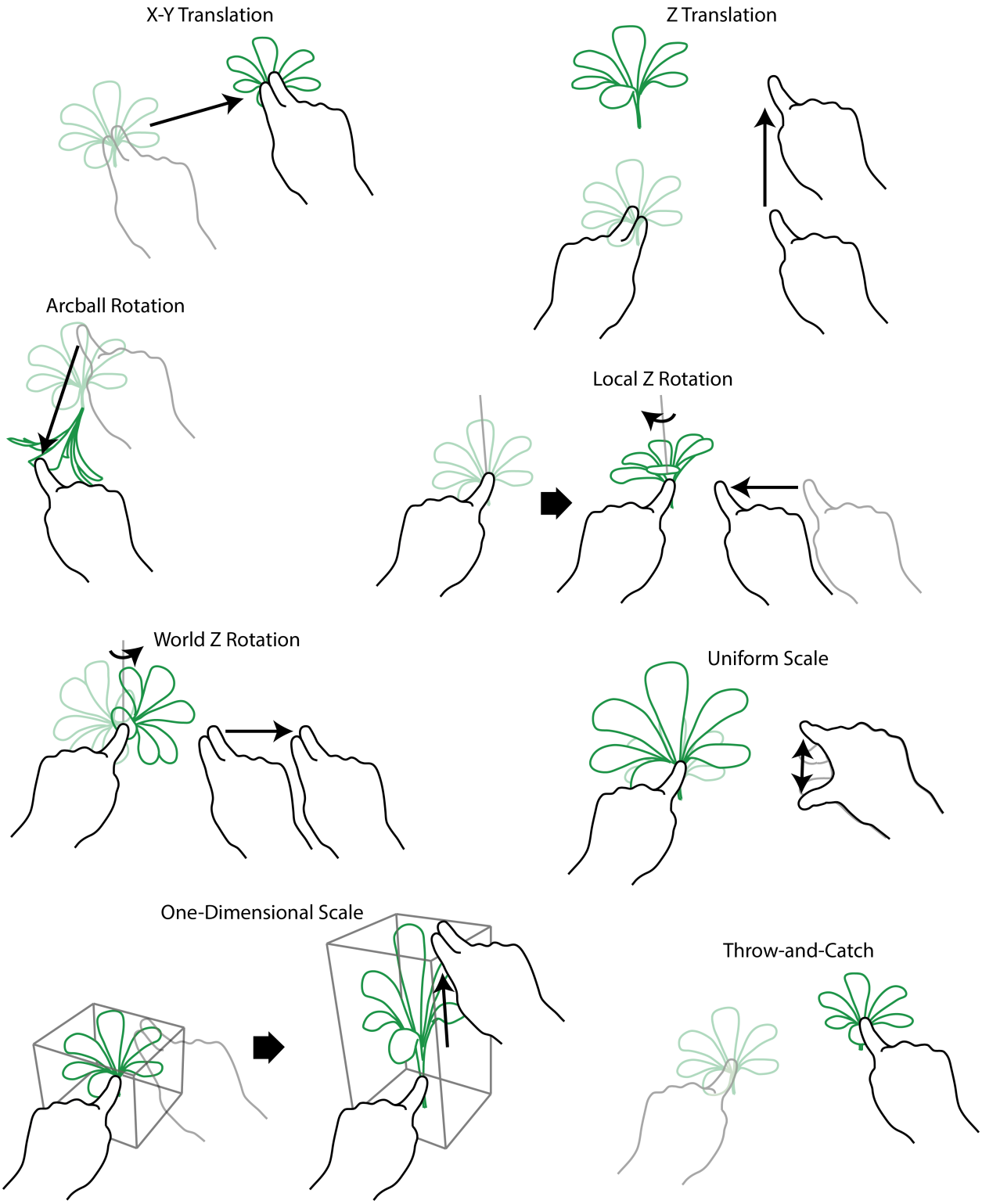


Figure 5.6: Set of object manipulation gestures.

are sufficient for specifying the orientation of an organic object. For some objects such as rocks that do not have a natural orientation, we provide world z rotation. We also include both uniform and one-dimensional scaling along the object's local axes, to provide additional methods to add variation to an object.

To help TM transport objects across long distances, we provide the *throw-and-catch* operation. Mouse-based interfaces often require dragging to transport an object from one location to another, as typically exhibited in the *drag-and-drop* technique. The *Boomerang* [74] technique for use with a mouse allows the user to suspend the dragging component by using a mouse flick gesture to *throw* the object off the screen. The user can later *catch* the object to resume dragging. With multitouch throw-and-catch, TM teleports an object by specifying the source and target locations simultaneously, thus eliminating the time needed to drag the object.

The gestures bound to object manipulation operations all require the artist to first use direct-touch to select an object for manipulation with either a one-touch or a conjoined touch. The most frequently used operations should be the simplest to perform, so arcball rotation and x-y translation only require the first touch to select the object and then a drag to manipulate it. For the remaining gestures, the artist uses both hands with no more than two fingers per hand. He selects the object with one hand, and then with the second hand, he adds touches away from the object to perform indirect manipulation. For each object manipulation gesture, the artist needs only to select the object and place any additional touches eyes-free to specify the object, operation, and parameters. In Maya, however, the artist needs to select a mode and sequentially target the object and manipulator.

To help make these gestures easy to remember, we used type of the first touch (one-touch, two-touch or conjoined) to indicate the category of manipulation. A conjoined touch on the object always begins a translation and a one-touch on the object begins either a rotation or a scale. When possible, we designed the motion of a gesture's second or third touch to reflect the motion of the object being manipulated. For example, translation along the z-axis moves an object up and down in screen space, so the second touch of the z translation gesture moves in an up and down motion. The second touch of the z rotation gesture moves side to side, which provides the sensation of spinning the object about a vertical axis. The second hand of the uniform scale gesture performs a pinching motion, which is commonly used for resizing photos on multitouch devices.

5.2.3 Camera Control

It is essential to provide a set of camera controls in Eden so the artist can inspect the scene from different angles. To control the camera, the artist first holds down the camera button, which invokes a *quasimode* [112] in which Eden interprets any additional touches as a camera control gesture. This technique is analogous to holding down the 'alt' key in Maya [7] to invoke camera control. We designed our camera control gestures to be similar to object manipulation gestures so they would be easier to remember. A one-touch drag rotates the camera in an arcball fashion, as it does for object manipulation. A conjoined touch drag

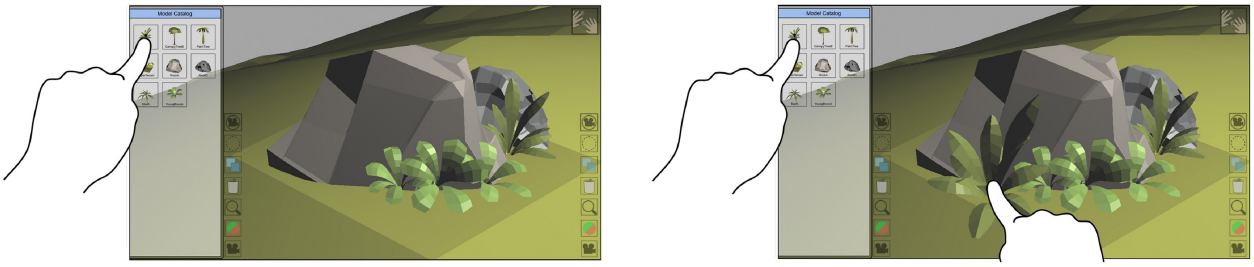


Figure 5.7: To add an object using throw-and-catch, the first finger selects the model and the second finger taps the position to place it.

translates the camera along the view plane (truck and pedestal), which is the same gesture for the planar translation in object manipulation. Lastly, we used the two-touch pinch gesture to move the camera forward and back (dolly), which is similar to the pinch used for scaling an object. We also included view direction rotation (roll) using the same two touches as dolly, as the orientation of the two fingers maps well to the camera's orientation. While holding the camera button, the artist can also choose a custom camera pivot by tapping on the scene or the artist can *frame* on an object (i.e., position the camera to provide a close-up view of the object) by tapping the object with a conjoined touch.

In an early iteration of Eden, we distinguished camera control from object manipulation not by a quasimode, but by touch location. If the first touch did not hit an object, then the system interpreted the touches as a camera control gesture, otherwise it interpreted the touches as manipulating the touched object. However, this method had a major flaw as objects could easily fill the entire view, making camera control impossible.

5.2.4 Adding Objects

The artist can add an object to the set using throw-and-catch. Specifically, he selects and holds the object in the model catalog to throw with one finger and specifies the destination to catch the new object instance by tapping with a second finger (Figure 5.7). The base of the new object rests directly on the terrain or the closest object underneath the touch. This technique allows the artist to quickly drop a pile of shrubs onto the terrain, for example. The artist can even use all five fingers to place five new objects with one action, although in practice it could be difficult to position all five fingers in the desired configuration. Since no two objects are identical in nature, if the user selects an object in the model catalog with a conjoined touch, we add a small amount of randomness in scale and orientation to the placed object.

In addition to adding objects from the model catalog to the set, the artist can throw a copy of an object from the set into the model catalog. To store a new object, the artist holds an object in the scene with one finger and then taps inside the drawer with a second finger. Adding objects into the model catalog allows the artist to set the size and other parameters

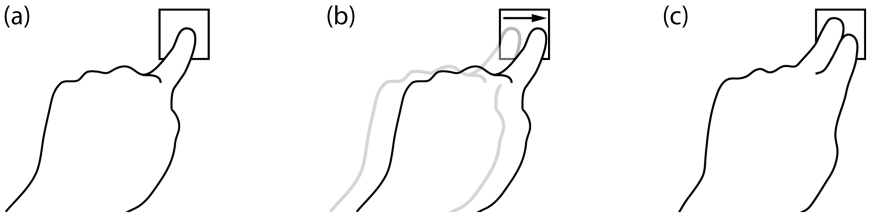


Figure 5.8: (a) One-touch to invoke quasimode. (b) Swipe on button triggers secondary action. (c) Conjoined touch to make mode explicit.

of the object and save it for future use. For example, he can scale up a rock object to the size of a boulder and then save it to the model catalog using this throw-and-catch technique.

5.2.5 Additional Commands

We incorporate quasimodes and stroke-recognition to support additional set construction commands.

Quasimodes and Buttons

Quasimodes in our application have the general advantage of keyboard-based quasimodes: the muscle tension needed to hold a key or button down reminds the user that a mode is currently invoked (Figure 5.8a). In addition to camera control, we use quasimodes for various secondary operations that TM finds useful for organic set construction. Although we intended to avoid modes, quasimodes allow us to reuse simple gestures thereby keeping gestures easy to perform. The simplest gesture is a tap, and touch-based interfaces are particularly good for tapping on objects [37, 69, 120]. By holding down one of the quasimode buttons (Figure 5.3), the artist can simply use another finger to tap on objects to freeze/unfreeze, delete, duplicate, or group select them.

We augment our buttons in a number of ways. We place descriptive icons on the buttons so the artist can recognize the icon, whereas with a keyboard the artist would need to memorize key bindings. More importantly, a user can perform gestures directly on the icon. For example, if we have saved camera positions, a swipe through the icon (Figure 5.8b) can cycle back and forth between the saved cameras in a manner similar to Moscovich's *Sliding Widgets* [97]. In addition, a conjoined touch tap on the camera icon (Figure 5.8c) can activate persistent camera mode, where the application only recognizes camera control gestures even if the camera button is not held down. Although we avoided regular modes, we provide camera mode so the artist can keep a hand free when only inspecting a set.

To make the buttons easy to access, we carefully considered their layout. Our multitouch screen sits almost horizontally, so in order to minimize the reach needed to hit buttons, we placed the buttons towards the bottom of the screen. Moreover, we put the same set of

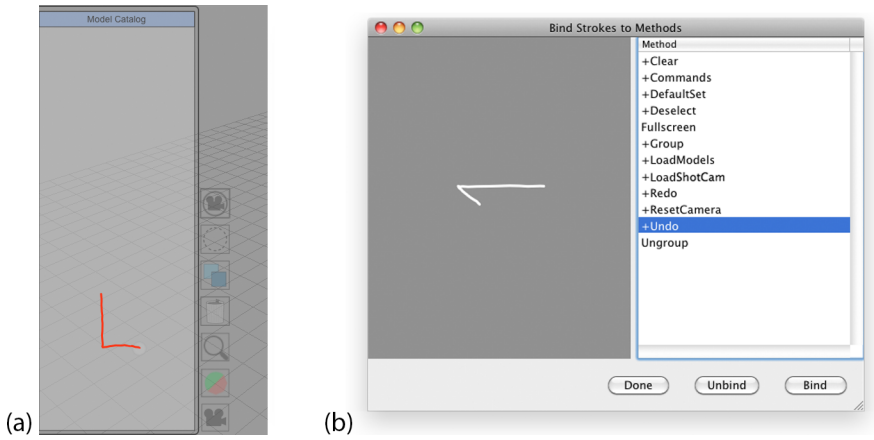


Figure 5.9: (a) Stroke pad. Drawing a stroke executes the corresponding command. (b) Stroke binding panel. The left panel displays the stroke bound to the highlighted command in the right panel. The artist can choose his own stroke by drawing a new stroke in the left panel.

buttons on both sides of the screen to allow either hand to initiate a quasimode. We also made our buttons larger than the width of a finger to provide easy targeting.

Stroke Commands

In mouse and keyboard interfaces, commands are typically executed with hotkeys and menus. To keep the artist's focus on the content, we avoided cluttering the interface with buttons or requiring the artist to navigate through menu hierarchies. Instead, the artist can execute commands by drawing single-stroke symbols in the stroke pad of the drawer (Figure 5.9a). For example, drawing an 'L' opens a load model panel, whereas drawing a left arrow performs undo. The stroke pad interprets any touch as a potential stroke command, which allows the artist to execute single-stroke commands that do not conflict with default object manipulation operations. Since the stroke pad is large and always in the same location, the artist can easily target the pad and draw a stroke with the left hand. Strokes can be difficult to remember, so the artist can define his own strokes for the supported commands, using a stroke binding panel (Figure 5.9b). We use the dollar gesture recognizer [139] for stroke recognition.

5.3 Qualitative Evaluation

We asked TM to evaluate his experience using Eden to build a set for a feature film developed by Pixar Animation Studios. We also asked a second set construction artist who has not previously used Eden to evaluate the system from the perspective of a novice user.

5.3.1 Apparatus

Eden runs on the same multitouch workstation that we used in Chapter 3. The size of the screen is 72.5 x 43.5 cm with a resolution of 1280 x 768 pixels. The artist interacts with the table by standing in front of the screen, which is mounted at a 23 degree incline. For text entry the artist uses a keyboard connected to a terminal next to the multitouch workstation. Text entry is reserved for infrequent actions such as naming a new set before saving it.

5.3.2 Veteran User Experience

Over the course of two 30-minute sessions, TM used Eden to build a set consisting of 136 trees for the feature film. He had built the same set previously in Maya, but he and his supervisor found no difference in quality between the two sets. We summarize his experience and evaluation of the system.

Object manipulation: According to TM, the rotation and scaling gestures on Eden are particularly effective because he does not need to break object selection and manipulation into separate steps. In Maya, he must first select the object to manipulate and then carefully pick a small manipulator to adjust the object. In Eden, both the object and the manipulation operation are specified by the gesture. For rough placement, x-y translation in Eden is faster than in Maya. However, TM needs more precision when fine-tuning object positions, and x-y translation is cumbersome on a small object because the conjoined touch obscures the position of the object. Also, TM occasionally needs to dolly close to an object in order to select it, because distant or partially occluded objects have small target areas making them difficult to select. In working with Eden, TM did discover an unintended but positive side effect: in certain situations our implementation permits him to switch between operations without lifting the finger selecting the object. For example, if TM first performs an x-y translation, he can then fluidly transition to z translation by adding a one-touch with a second hand, without lifting the conjoined touch used for x-y translation.

Camera control: For TM, the Eden camera controls have slight usability advantages over Maya. Clutching a mouse is a physical annoyance for TM as he sometimes inadvertently slides the mouse off the working surface, which is not an issue with direct-touch. However, TM finds framing on an object difficult with Eden, because it often requires tapping on a small object, which is imprecise with the conjoined touch.

Adding objects: TM finds that adding objects to a set with Eden is more efficient than with Maya. Using the throw-and-catch technique, he can tap directly where a new object should roughly be positioned. The visual icons in the model catalog also help remind him what each model looks like. Maya does not provide preview icons.

Additional commands: TM considers quasimodes to be effective for accessing additional commands. Quasimodes permit the reuse of simple gestures, which makes the corresponding commands easy to invoke. The icons on the buttons help him remember which quasimodes are available. TM also finds strokes are as effective as keyboard shortcuts for executing simple commands such as undo and redo.

Repetitive stress injury: Over the years building sets, TM has developed repetitive stress injury (RSI) and currently wears a wrist protector on the hand he uses to control the mouse. To prevent his RSI from worsening, he takes regular breaks and finds other ways to exercise his wrist. TM finds that using two hands with Eden better balances the load between both hands. However, we do not have enough experience to know if different RSI problems will arise from multitouch interaction.

TM estimates that he is 20% faster building a set with Eden than with Maya. These results suggest that we have succeeded in providing an expert set construction artist a fully functioning multitouch application that is more efficient than Maya, an industry-approved application that has been refined over many years.

Nevertheless there is still room for improvement in both the interface and hardware of our system. According to TM, coarse placement is sufficient for the majority of the organic set construction task. But, if we can address the occlusion problem for x-y translation and the precision problem for selecting small objects with techniques such as Shift [128] or FingerGlass [64], then we can provide a better overall experience for TM. Our hardware also limits the effectiveness of Eden. Our multitouch sensor only runs at 30 Hz and our touch detection system has a small delay when responding to input, which makes Eden less responsive than Maya. Also, two nearby, but separate touches sometimes collapse into a conjoined touch due to noise in the sensor, so the application may at times interpret TM's intentions incorrectly.

5.3.3 New User Experience

We designed Eden using the input from one set construction artist. To gain a better understanding of Eden's potential, we asked TP, a set construction artist with two years of experience, to use Eden for three, 45-minute sessions.

In the first session, we introduced Eden to TP, explaining its operations and features. He spent the second half of the session exploring and familiarizing himself with the interface by constructing a few small sets. His biggest early frustration was camera control, as the sensitivity did not match the Maya controls he was used to.

At the start of the second session we asked TP to recall the object manipulation gestures and the camera control gestures. He was able to perform each one without help, with the exception of world z rotation and one-dimensional scale. These two operations tend to be the least frequently used for object manipulation. After spending 20 minutes warming up and refamiliarizing himself with the interface, he was ready to construct a set. In 15 minutes he was able to build the set shown in Figure 5.3. At this stage, TP claimed he was "having fun" and building organic sets with his hands "feels like gardening." By the end of session two TP felt he was over the initial hump of learning the gestures.

TP returned for the third session three days after session two. Despite the break, TP was able to recall all the object manipulation and camera control gestures. He remembered the quasimode functions as well as the stroke commands for loading models, performing undo,

and resetting the camera position. After ten minutes of practicing the various gestures, he spent the remaining time constructing a set.

Overall, TP found that Eden provided a more immersive experience than Maya, because he felt like he was "sculpting a space" with his hands and could "forget about the technology," which made him feel like he was sketching. In addition to enjoying the tactile quality of interacting with the objects, he found that using both hands to quickly transport objects in and out of the drawer was effective and efficient. We are encouraged that TP was able to learn and remember all of the object manipulation and camera control gestures after just two sessions, suggesting that our gestures are easy to learn and recall. Like TM, TP also discovered that he could perform fluid transitions between operations without lifting the selecting finger. He used fluid transitions frequently.

Although TP had a positive experience overall, he found certain operations difficult to perform with Eden. While he could control the camera, he was uncomfortable with the gestures. He found that camera roll mapped to two fingers was confusing as he would inadvertently roll the camera when he wanted to only perform a dolly. Although the dolly gesture has enough degrees of freedom to also specify roll, in future iterations of Eden we could separate the two operations or remove roll entirely. Also, his interpretation for the pinch motion to perform dolly was inverted from its intended use. When he spread two fingers apart he thought he was pushing the set away, so he expected the camera to dolly away from the set; instead, the camera dollied towards the set. Future iterations of Eden may resolve this difference in interpretation by giving TP a method to customize gestures. For the majority of the organic set construction process, TP did not find precision to be an issue. However, like TM, when TP wanted to fine-tune the positions of a few objects, he had to dolly in close, otherwise he found selecting and manipulating small or distant objects difficult.

As we observed with TM, our hardware has room for improvement. TP felt he had to apply heavy pressure on the screen when performing gestures, making them slow and possibly straining on the hand. If we improve the hardware to recognize lighter touches and be more responsive, then we can provide a more comfortable and seamless experience.

5.4 Lessons Learned

Based on our experiences designing a complete multitouch application and our interviews with professional set construction artists who used it, we summarize the following lessons:

- **Justify simultaneous interactions** – Determine how often users will use simultaneous interactions, if at all. If the benefits of simultaneous interactions do not outweigh the complexity of handling simultaneous interactions and the cognitive difficulty for a user to perform them, then support just one interaction at a time.

- **Balance gestures across both hands** – Split the touches across both hands in order to reduce the number of touches per hand and increase mobility. Fewer touches per

hand makes gestures faster and more comfortable to perform.

- **Reuse gestures via modes** – Gestures generally become more complicated as the number of operations increases. Although we sought to reduce modes, quasimodes allow reuseable gestures, which keep gestures simple.
- **Interpret gestures based on location** – Reduce conflicts by interpreting gestures made in one location (e.g., stroke pad) differently than gestures made in other locations.
- **Identify low precision tasks** – Evaluate the proposed application and consider whether precision will be a major factor. Techniques that compensate for touch imprecision [14] may slow the user's performance and limit the effectiveness of a multitouch interface.
- **Factor in occlusion** – Consider designing gestures that use indirect manipulation, so the user can perform manipulations away from the object and reduce hand occlusion, or allowing the user to release static touches once a gesture is recognized [143]. In addition, consider augmenting the interface to be occlusion-aware [127].
- **Throw objects** – A mouse cursor cannot be in two places at once, whereas a user's hands can. Pass objects between the hands to reduce hand movement times. Consider integrating a flick gesture to indicate a throw.
- **Design fluid transitions between gestures** – If two operations are often performed in sequence, design corresponding gestures that smoothly transition between the two. For example, a one-touch gesture can transition to a two-touch gesture with the application of a second touch.

5.5 Extensions

Eden primarily supports the rough placement of objects for organic set construction. For general set construction, we need to augment Eden with more precise interaction techniques. An artist should be able to adjust a single spatial parameter of an object without affecting the others, so we need additional gestures that control each spatial parameter separately. We could also incorporate existing techniques such as snap-dragging [16] to help the artist precisely align and position manmade objects found in general sets. In addition, a better hardware setup could improve precision by increasing touch resolution and reducing latency. Aside from object manipulation, we expect Eden's basic interface for camera control, adding objects, and setting modes to be sufficient for general set construction.

In addition to general set construction, our design decisions should transfer well to other single-user multitouch applications. By restricting applications to support only one operation at a time, developers can design simple, two-handed gestures that are easy to remember and comfortable to perform. Quasimodes allow the reuse of simple one-handed gestures, and when applicable, throw-and-catch eliminates the need for dragging.

5.6 Conclusion

We have demonstrated that with careful application and gesture design, multitouch input is a promising enhancement to the workflow for organic set construction artists. Multitouch displays for the desktop are becoming more widely available, providing more opportunity for users in different industries to benefit from professional multitouch applications. Many of our design guidelines and lessons learned are applicable to the development of new multitouch applications. Good developer tools can also facilitate the creation and adoption of these applications. Thus, it is important to consider the tools for developing multitouch applications. In the next chapter, we present a new multitouch framework to help developers build and manage multitouch gesture sets.

Chapter 6

Representing Multitouch Gestures as Regular Expressions

Designing and implementing Eden gave us significant experience with building a complete multitouch application. The Eden gesture set consists of variety of gestures that relies heavily on touch sequencing and hit-targets. We implemented these gestures from scratch by processing low-level touch events like the *touch-down*, *touch-move*, and *touch-up* events reported by current multitouch frameworks. These touch events are analogous to the *mouse-down*, *mouse-move*, and *mouse-up* events generated by mouse-based GUI frameworks. As with mouse events, multitouch frameworks deliver touch events to widgets or objects in the scene hit by the respective touches. A *multitouch gesture* is thus a sequence of these touch events on scene objects such as *touch1-down-on-object1*, *touch1-move*, *touch2-down-on-object2*, etc. Unlike mouse gestures which track the state of a single point of input, multitouch gestures often track the states of many points of contact in parallel as they each appear, move, and disappear. When building Eden, we found that writing robust recognition code with current frameworks for sets of multitouch gestures was challenging for two main reasons:

1) Multitouch gesture recognition code is split across many locations in the source. Today, developers must write separate callbacks to handle each low-level touch event for each scene object. Implementing a gesture requires tracking the sequence of events that comprise the gesture across disjoint callbacks. Consider a gesture in which the user must simultaneously touch two scene objects to connect them as nodes in a graph. The developer must maintain the state of the gesture across callbacks for each object and communicate this state via messages or global variables. Such *stack ripping* [3] results in complicated and hard-to-read "callback soup." Breaking up gesture code makes it difficult to not only express a gesture, but also to understand and modify the gesture later.

2) Multiple gestures may be based on the same initiating sequence of events. Consider two gestures, one for rotating and one for scaling objects in a 2D scene. Both gestures require the first touch to fall on an object to select it, while the motion of a second touch indicates the transformation. At the level of touch events, the two sequences are

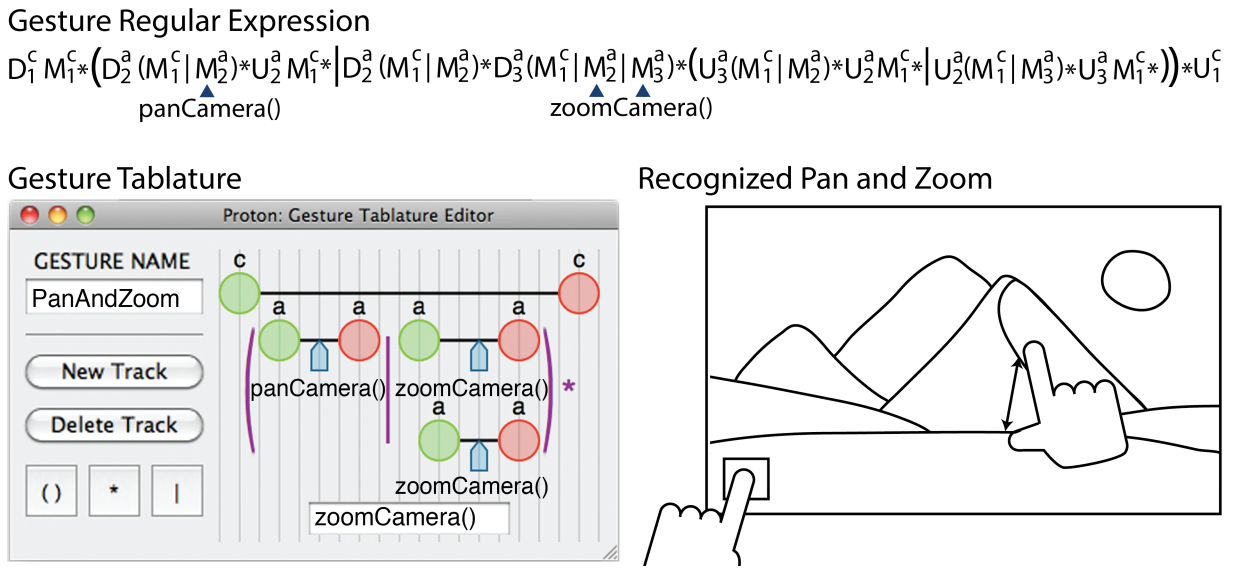


Figure 6.1: Proton represents a gesture as a regular expression describing a sequence of touch events. Using Proton's gesture tablature, developers can design a multitouch gesture graphically by arranging touch sequences on horizontal tracks. Proton converts the tablature into a regular expression. When Proton matches the expression with the touch event stream, it invokes callbacks associated with the expression.

identical and the developer must write disambiguation logic to resolve this conflict within the respective callbacks. Yet, such gesture conflicts may not even be apparent until a user performs one of the gestures at runtime and the application responds with an unintended operation. In large applications that support many gestures, identifying and managing such gesture conflicts can be extremely difficult.

These two sources of complexity – splitting gesture recognition code across callbacks and conflicts between similar gestures – make it especially difficult to maintain and extend multitouch applications.

To help alleviate these issues, we have developed *Proton*, a multitouch framework that allows developers to declaratively specify the sequence of touch events that comprise an entire gesture using a single regular expression. Proton automatically manages the underlying state of the gesture; the developer writes one callback function that is invoked whenever the stream of input events matches the gesture's regular expression. To provide visual feedback over the course of a gesture, the developer can optionally define additional callbacks that are invoked whenever the input stream matches expression prefixes. Our approach significantly reduces splitting of the interaction code as developers do not have to manually manage state or write a callback for every touch event.

Proton touch events are also customizable. Developers can define custom touch attributes and incorporate them into declarative gesture definitions. We demonstrate the benefits of customization with example implementations of five attributes: a *direction* attribute for

specifying touch trajectory, a *pinch* attribute for detecting when touches move towards one another, a *touch area* attribute for simulating pressure, a *finger orientation* attribute that provides an additional parameter for selecting menu items, and a *screen location* attribute for simulating hand ID and user ID.

Declaratively specifying gestures as regular expressions also allows Proton to statically analyze the gesture expressions to identify the conflicts between them. Instead of having to extensively test for such conflicts at runtime, our static analyzer tells the developer exactly which sets of gestures conflict with one another at compile time. The developer can then choose to write the appropriate disambiguation logic or modify the gestures to avoid conflicts.

Complex regular expressions can be difficult to author, read, and maintain [18]. In regular expressions that recognize multitouch events, the interleaving of multiple, simultaneous touch events in a single event stream exacerbates this complexity. To help developers build gesture expressions, Proton offers *gesture tablature*, a graphical notation for multitouch gestures (Figure 6.1). The tablature uses horizontal tracks to describe touch sequences of individual fingers. Using Proton's tablature editor, developers can author a gesture by spatially arranging touch tracks and graphically indicating when to execute callbacks. Proton automatically compiles the gesture tablature into the corresponding regular expression.

We demonstrate the expressivity of Proton with implementations of four proof-of-concept applications: a shape manipulation application, a sketching application, a unistroke text entry technique [140], and the game Pong. Using these examples, we show that Proton significantly reduces splitting of gesture recognition code and that it allows developers to quickly identify and resolve conflicts between gestures. Proton increases maintainability and extensibility of multitouch code by simplifying the process for adding new gestures to an existing application.

Finally, we present a user study that investigates how quickly and accurately developers recognize and reason about gestures described using gesture regular expressions, gesture tablatures, and iOS-style procedural event-handling pseudocode.

6.1 A Motivating Example

We begin with a motivating example that demonstrates the complexity of implementing a custom gesture using Apple's iOS [6], which is structurally similar to many commercial multitouch frameworks. We later show how writing the same example in Proton is significantly simpler, making it easier to maintain and extend the interaction code.

As the user interacts with a multitouch surface, iOS continuously generates a stream of low-level touch events corresponding to *touch-down, touch-move* and *touch-up*. To define a new gesture, the developer must implement one callback for each of these events, *touchesBegan()*, *touchesMoved()* and *touchesEnded()*, and register them with objects in the scene. For each touch event in the stream, iOS first applies hit-testing to compute the scene object under the touch point and then invokes that object's corresponding callback.

It is the responsibility of the developer to track the state of the gesture across the different callbacks. Consider a two-touch rotation gesture where both touches must lie on the same object with the following pseudocode.

iOS: rotation gesture

```
1:  shape.addRecognizer(new Rotation)

2:  class Rotation
3:  gestureState ← possible /*gesture state*/
4:  touchCount ← 0

5:  function touchesBegan()
6:  touchCount ← touchCount + 1
7:  if touchCount == 2 then
8:    gestureState ← began
9:  else if touchCount > 2 then
10:   gestureState ← failed

11: function touchesMoved()
12: if touchCount == 2 and gestureState != failed then
13:   gestureState ← continue
14:   /*compute rotation*/

15: function touchesEnded()
16: touchCount ← touchCount − 1
17: if touchCount == 0 and gestureState != failed then
18:   gestureState ← ended
19:   /*perform rotation cleanup*/
```

The gesture recognition code must ensure that the gesture begins with exactly two touches on the same object (lines 5-10), that rotation occurs when both touches are moving (lines 11-14) and that the gesture ends when both touches are lifted (lines 15-19). Counting touches and maintaining gesture state adds significant complexity to the recognition code, even for simple gestures. This state management complexity can make it especially difficult for new developers to decipher the recognition code.

Suppose a new developer decides to relax the rotation gesture, so that the second touch does not have to hit the object. The developer must first deduce that the gesture recognition code must be re-registered to the canvas containing the object in order to receive all of the touch events, including those outside the object. Next the developer must modify the *touchesBegan()* function to check that the first touch hits the object, and set the gesture state to failed if it does not. Although neither of these steps is difficult, the developer cannot make these changes without fully understanding how the different callback functions work together to recognize a single gesture. As the number of gestures grows, understanding how

they all work together and managing all the possible gesture states becomes more and more difficult.

6.2 Using Proton

Like iOS, Proton is an event-based framework. But, instead of writing callbacks for each low-level touch event, developers work at a higher level and declaratively define gestures as regular expressions comprised of sequences of touch events.

6.2.1 Representing Touch Events

A touch event contains three key pieces of information: the touch action (down, move, up), the touch ID (first, second, third, etc.), and a series of touch attribute values. These touch attribute values describe different characteristics of a touch. For example, one attribute value can indicate the hit-target of the touch, while a second attribute value can indicate the direction of the touch. The developer determines the number and types of touch attributes by creating attribute generators that process the touch event data and returns an attribute value for each touch event. Proton represents each event as a symbol with three components:

$$E_{T_{ID}}^{A_1:A_2:A_3...}$$

1) E is the touch action, either D, M, or U to represent touch-down, touch-move, or touch-up respectively. 2) T_{ID} is the touch ID that groups events belonging to the same touch. 3) $A_1 : A_2 : A_3...$, are the attribute values, where A_1 is the value corresponding to the first attribute, A_2 is the value corresponding to the second attribute, and so on.

For example, with a single attribute for hit-target, M_1^s represents *move-with-first-touch-on-shape-target*, where the 's' hit-target attribute value represents a shape object. We can add a second attribute to indicate the direction of the touch. $M_1^{s:W}$ represents *move-with-first-touch-on-shape-target-in-west-direction*, where the additional 'W' direction attribute value represents the west direction. As we explain in Section 6.3.1, Proton works with the multitouch hardware and attribute generators provided by the developer to create a stream of these touch event symbols.

6.2.2 Gestures as Regular Expressions

The developer can define a gesture as a regular expression over these touch event symbols. Proton supports the use of the three standard regular expression operators: parentheses to group symbols, Kleene stars ('*') to specify repetitions, and vertical bars ('|') to specify disjunctions. Figure 6.2 shows the regular expressions describing three shape manipulation operations using a hit-target attribute:

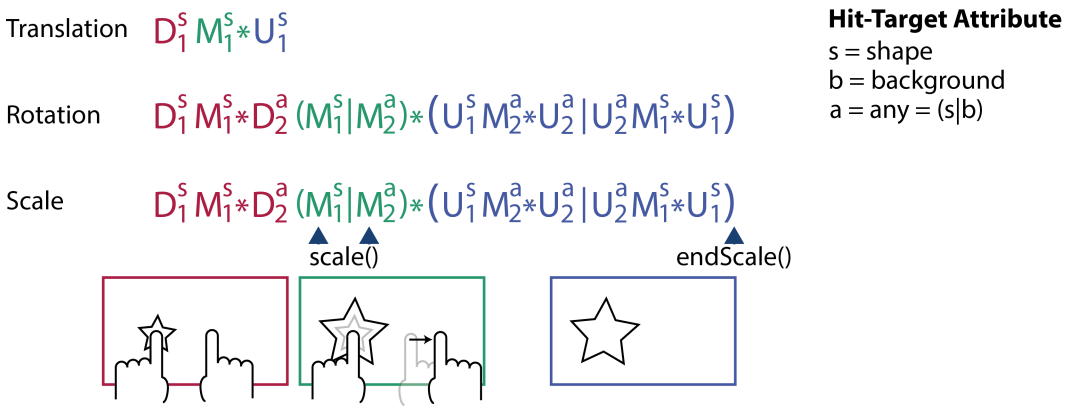


Figure 6.2: Regular expressions for translation, rotation and scale gestures. The thumbnails illustrate the user's actions corresponding to the colored symbols for the scale gesture.

Translation: First touch down on a shape to select it (red symbol). The touch then moves repeatedly (green symbol). Finally, the touch lifts up to release the gesture (blue symbol).

Rotation: First touch down on a shape followed by a second touch down on the shape or canvas (red symbols). Then both touches move repeatedly (green symbols). Finally, the touches lift up in either order (blue symbols).

Scale: First touch down on a shape followed by a second touch down on the shape or canvas (red symbols). Then both touches move repeatedly (green symbols). Finally, the touches lift up in either order (blue symbols).

Describing a gesture as a regular expression both simplifies and unifies the gesture recognition code. The Proton pseudocode required to implement the general rotation gesture (with a second touch starting on a shape or background) is:

Proton: rotation gesture

/*indices:1 2 3 4 5 6 7 8 9 10 11*/

1: $gest: D_1^s M_1^s{*}D_2^a\ (M_1^s|M_2^a)^*\ (U_1^s M_2^a{*}U_2^a|U_2^a M_1^s{*}U_1^s)$

2: $gest$.addTrigger($rotate(), 4$)

3: $gest$.addTrigger($rotate(), 5$)

/*compute rotation in $rotate()$ callback*/

4: $gest$.finalTrigger($endRotate()$)

/*perform rotation cleanup in $endRotate()$ callback*/

5: $gestureMatcher$.add($gest$)

Instead of counting touches and managing gesture state, the entire gesture is defined as a single regular expression on line 1. Unlike iOS, Proton handles all of the bookkeeping required to check that the stream of touch events matches the regular expression. The developer associates callbacks with *trigger* locations within the expression. The second parameters in

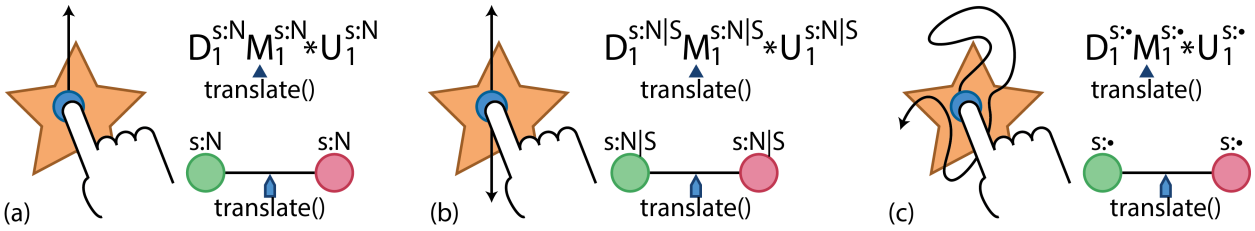


Figure 6.3: Combining the hit-target and direction attributes, the developer can specify a gesture to translate a shape (denoted as 's') with varying degrees of specificity: (a) north only, (b) north and south only, (c) in any direction.

lines 2 and 3 create triggers at the 4th and 5th symbols of the expression. The application invokes the *rotate()* callback each time the event stream matches the regular expression up to the trigger location. In this case, the match occurs when the two touches are moving; the callback can provide on-screen feedback. The location for the final trigger (line 4) is implicitly set to the end of the gesture expression.

To change a gesture's touch sequence, the developer simply modifies the regular expression. For example, to require users to place the second touch on a shape, the developer need only change the regular expression so that the second touch down must occur on a shape rather than on a shape or background. In iOS, making this change requires much deeper understanding of the state management in the gesture recognition code.

6.2.3 Gestures with Multiple Touch Attributes

By implementing several attribute generators, the developer can also create gestures that combine attributes. For example, the expression $D_1^{s:N} M_1^{s:N} * U_1^{s:N}$ describes a one-finger northward motion on the shape object (Figure 6.3a). Often a gesture allows for certain attributes to take on one of several values. The developer can use the '|' character to denote the logical *or* of attribute values. For example, the expression $D_1^{s:N|S} M_1^{s:N|S} * U_1^{s:N|S}$ extends the previous gesture to allow both north and south motions (Figure 6.3b). Proton expands the '|' shorthand into the full regular expression $(D_1^{s:N}|D_1^{s:S})(M_1^{s:N}|M_1^{s:S})*(U_1^{s:N}|U_1^{s:S})$. Proton also allows developers to use the '•' character to denote a wildcard which specifies that an attribute can take any value, effectively ignoring the attribute during matching. For example, if the direction attribute A_2 can take the set of values $\{N, S, E, W\}$, the expression $D_1^{s:\bullet} M_1^{s:\bullet} * U_1^{s:\bullet}$ describes any one-finger trajectory on the shape object (Figure 6.3c). In this expression, the symbol $M_1^{s:\bullet}$ expands to $M_1^{s:N}|M_1^{s:S}|M_1^{s:E}|M_1^{s:W}$.

6.2.4 Gesture Tablature

When a gesture includes multiple fingers each with its own sequence of touch-down, touch-move and touch-up events, the developer may have to carefully interleave the parallel events in the regular expression. To facilitate authoring of such expressions, Proton introduces

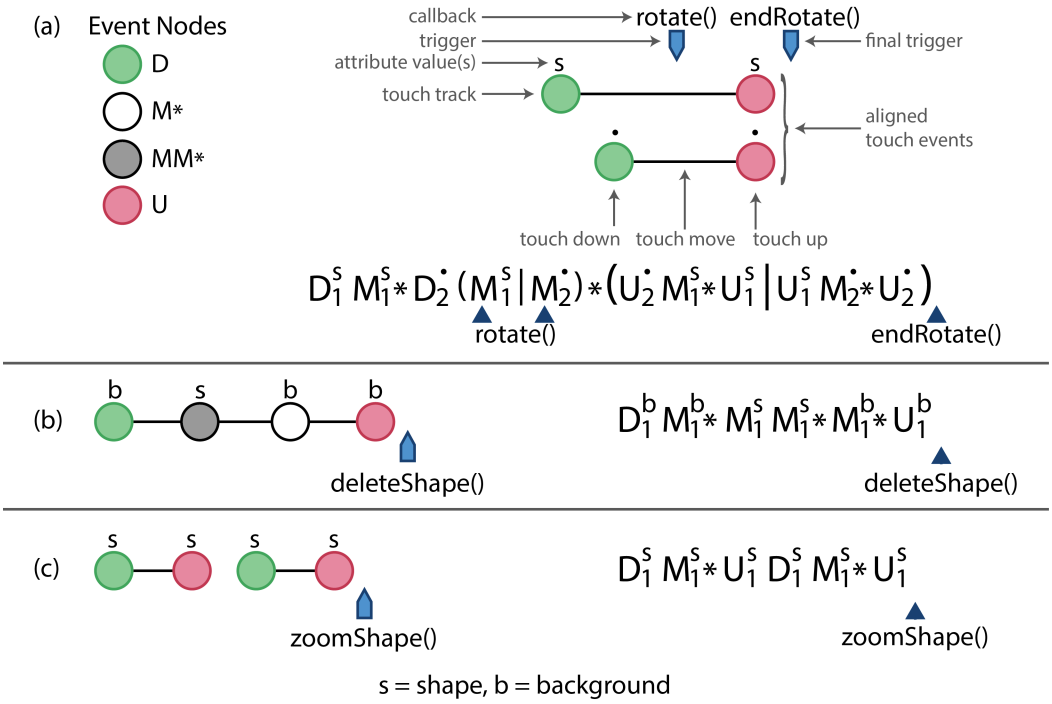


Figure 6.4: (a) Tablature for a two-touch rotation gesture. (b) Tablature for a strikethrough delete gesture. (c) Tablature for double tap zoom.

gesture tablature (Figure 6.4). This notation is inspired by musical notations such as guitar tablature and step sequencer matrices. Proton converts the graphical notation into a regular expression, properly interleaving parallel touch events. Proton ensures that the resulting regular expressions are well formed.

Using Proton's interactive tablature editor, developers can graphically indicate a touch event sequence using horizontal *touch tracks*. Within each track, a green node represents a touch-down event and a red node represents a touch-up event. The attribute values associated with each touch event are listed above the corresponding nodes. The black line connecting the nodes represent an arbitrary number of touch-move events and inherit the attributes of the preceding node. Vertical positions of nodes specify the ordering of events between touch tracks: event nodes to the left must occur before event nodes to the right, and when two or more event nodes are vertically aligned the corresponding events can occur in any order. This separation of concerns facilitates authoring as developers can first design the event sequence for each finger on a separate track and then consider how the fingers must interact with one another.

For example, Figure 6.4a shows the tablature for a two-touch rotation gesture with a hit-target attribute where the first touch must hit some shape, the second touch can hit anywhere, and the touches can be released in any order. Proton converts the vertically aligned touch-up nodes into a disjunction of the two possible event sequences: $(U_1^s M_2^{s*} U_2^s | U_2^s M_1^{s*} U_1^s)$.

Thus, Proton saves the developer the work of writing out all possible touch-up orderings. In addition, tablature uses the same shorthand ('|' characters and '•' wildcards) as regular expressions for specifying multiple attribute values (Figure 6.3). We describe the algorithm for converting tablatures into regular expressions in Section 6.3.5.

Proton inserts touch-move events between touch-down and touch-up events when converting tablature into regular expressions. To indicate that attributes associated with a touch can change during a move, the developer can insert explicit touch-move nodes. Consider a strikethrough gesture to delete shapes that starts with a touch-down on the background, then moves over a shape, before terminating on the background again. The corresponding tablature (Figure 6.4b) includes a gray node with hit-target attribute value s indicating that at least one move event must occur on a shape and a white node with hit-target attribute value b, indicating that the touch may move onto the background before the final touch-up on the background. Developers can also express multiple recurring touches (e.g., a double tap), by arranging multiple touch tracks on a single horizontal line (Figure 6.4c).

Developers can also graphically specify trigger locations and callbacks. A *local trigger* arrow placed directly on a touch track associates a callback only with a symbol from that track (Figure 6.1). A *global trigger* arrow placed on its own track (e.g., *rotate()* in Figure 6.4a) associates the callback with all aligned events (down, move or up). A *final trigger* is always invoked when the entire gesture matches (Figure 6.4b,c). To increase expressivity our tablature notation borrows elements from regular expression notation. The developer can use parentheses to group touches, Kleene stars to specify repetitions, and vertical bars to specify disjunctions. Figure 6.1 shows an example where the user can place one touch on a button and perform repeated actions with one or two additional touches.

6.2.5 Static Analysis of Gesture Conflicts

Gesture conflicts arise when two gestures begin with the same sequence of touch events. Current multitouch frameworks provide little support for identifying such conflicts and developers often rely on runtime testing. However, exhaustive runtime testing of all gestures in all application states can be prohibitively difficult. Adding or modifying a gesture can lead to new conflicts, which then requires retesting of all gestures for conflicts.

Proton's static analysis tool identifies conflicts between gesture expressions at compile time. Given the regular expressions of any two gestures, this tool returns the extent to which the gestures conflict, in the form of a longest prefix expression that matches both gestures. For example, when comparing the translation and rotation gestures (Figure 6.2), it returns the expression $D_1^s M_1^s *$, indicating that both gestures will match the input stream whenever the first touch lands on a shape and moves. When the second touch appears, the conflict is resolved as translation is no longer possible.

Once the conflict has been identified the developer can either modify one of the gestures to eliminate the conflict or write disambiguation code that assigns a confidence score to each interpretation of the gesture as we describe in Section 6.3.3.

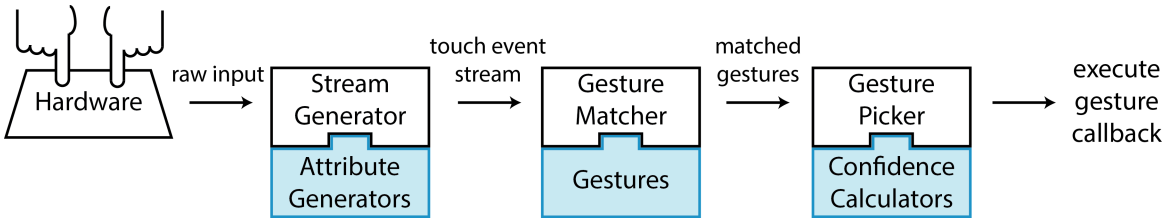


Figure 6.5: The Proton architecture. The responsibilities of the application developer are shown in blue.

6.3 Implementation

The Proton runtime system includes three main components (Figure 6.5). The *stream generator* converts raw input data from the hardware into a stream of touch events. The *gesture matcher* compares this stream to the set of gesture expressions defined by the developer and emits a set of candidate gestures that match the stream. The *gesture picker* then chooses amongst the matching gestures and executes any corresponding callback. Proton optionally *splits the touch stream* to support recognition of multiple simultaneous gestures. Proton also includes two compile-time tools. The *tablature conversion algorithm* generates regular expressions from tablatures and the *static analysis tool* identifies gesture conflicts.

6.3.1 Stream Generator

Multitouch hardware provides a sequence of time-stamped touch points. Proton converts this sequence into a stream of touch event symbols (Figure 6.6 Top). It groups touches based on proximity in space and time, and assigns the same T_{ID} for touch events that likely describe the path of a single finger. The stream generator increases the T_{ID} by one for each new touch down. Proton also appends a series of touch attribute values for each touch event. It is the responsibility of the developer to provide attribute generators that process touch events and return attribute values. For example, the developer can implement an attribute generator that performs hit-testing and returns the hit-target of the touch event. We describe attribute generation in more detail in Section 6.4.

When the user lifts up all touches, the stream generator flushes the stream to restart matching for subsequent gestures. Some gestures may require all touches to temporarily lift up (e.g., double tap, Figure 6.4c). To enable such gestures, developers can specify a timeout parameter to delay the flush and wait for subsequent input. To minimize latency, Proton only uses timeouts if at least one gesture prefix is matching the current input stream at the time of the touch release.

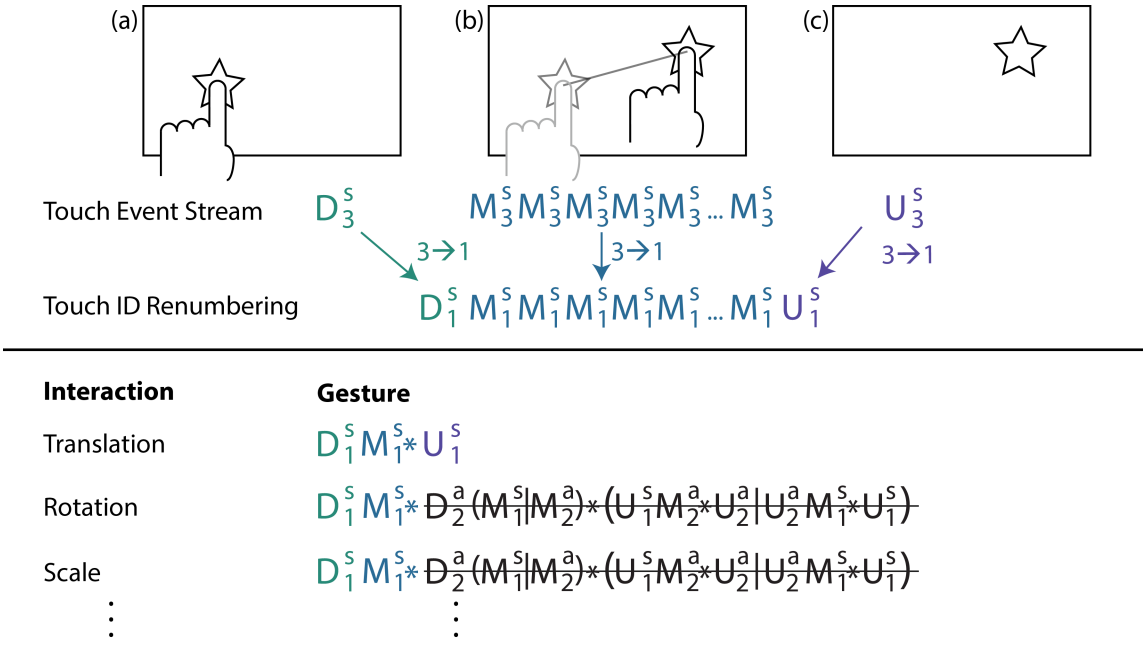


Figure 6.6: Top: Proton generates a touch event stream from a raw sequence of touch points given by the hardware. (a) The user touches a shape, (b) moves the touch and (c) lifts the touch. The gesture matcher renumbers unique T_{ID}s produced by the stream generator to match the gesture expressions. Bottom: The gesture matcher then sequentially matches each symbol in the stream to the set of gesture expressions. Translation, rotation, and scale all match when only a single finger is active, (a) and (b), but once the touch is lifted only translation continues to match, (c).

6.3.2 Gesture Matcher

The gesture matcher keeps track of the set of gestures that can match the input stream. Initially, when no touches are present, it considers all gestures to be possible. As it receives new input events, the matcher compares the current stream against the regular expression of each candidate gesture. When a gesture no longer matches the current stream the matcher removes it from the candidate set. At each iteration the matcher sends the candidate set to the gesture picker.

In a gesture regular expression, T_{ID} denotes the touch by the order in which it appears relative to the other touches in the gesture (i.e., first, second, third, ... touch within the gesture). In contrast, the T_{ID}s in the input event stream are globally unique. To properly match the input stream with gesture expressions, the matcher first renumbers T_{ID}s in the input stream, starting from one (Figure 6.6 Top). However, simple renumbering cannot handle touch sequences within a Kleene star group, e.g., $(D_1^a M_1^a * U_1^a)*$, because such groups use the same T_{ID} for multiple touches. Instead we create a priority queue of T_{ID}s and assign them in ascending order to touch-down symbols in the input stream. Subsequent touch-

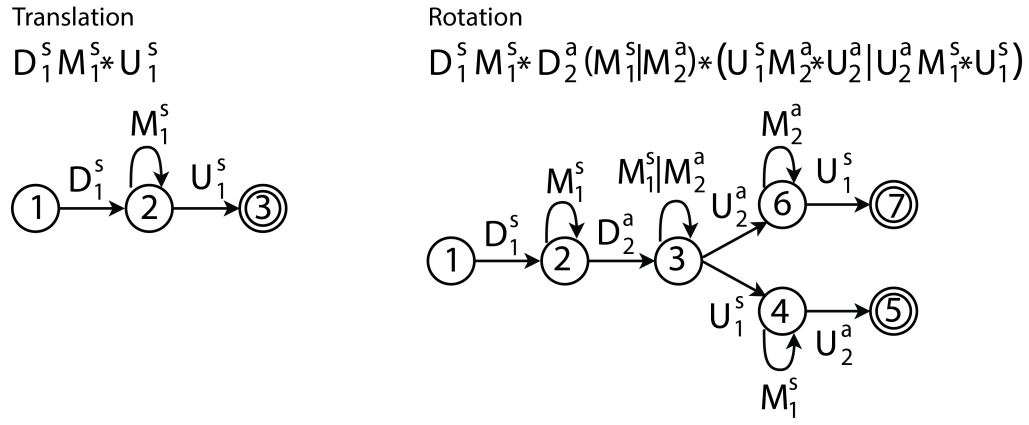


Figure 6.7: Proton converts the developer-defined regular expressions to finite-state machines for gesture matching.

move and touch-up symbols receive the same T_{ID} as their associated touch-down. Whenever we encounter a touch-up, we return its T_{ID} to the priority queue so it can be reused by subsequent touch-downs.

The gesture matcher precomputes the finite-state machine (FSM) for each gesture expression (Figure 6.7). Before the first touch event, all gestures are candidates for matching and their respective FSMs begin in the start state. For each touch event, the gesture matcher traverses the FSMs of all candidate gestures. At any iteration, if traversal leads to a fail state in an FSM, the respective gesture is removed from the candidate set (Figure 6.6 Bottom). If traversal reaches an accept state, the gesture fully matches the input stream and the gesture callback is forwarded to the gesture picker where it is considered for execution. If traversal reaches a state with trigger callback, the gesture matcher fowards the trigger callback to the gesture picker. Finally, whenever the stream generator flushes the event stream, Proton reinitializes the set of possible gestures and matching begins anew.

6.3.3 Gesture Picker

The gesture picker receives a set of candidate gestures and any associated callbacks. In applications with many gestures it is common for multiple gesture prefixes to match the event stream, forming a large candidate set. In such cases, additional information, beyond the sequence of touch events, is required to decide which gesture the user intended to perform.

The developer can provide the additional information by writing a *confidence calculator* function for each gesture that computes a likelihood score between 0.0 and 1.0. In computing this score, confidence calculators can consider additional touch features not described in the matching sequence of touch event symbols. For example, a confidence calculator may analyze the timing between touch events or the trajectory of touch positions across move events. Consider the conflicting rotation and scale gestures shown in Figure 6.2. The confidence

calculator for scale might check if the touches are moving away from one another while the confidence calculator for rotation might check if one finger is circling the other. The calculator can prevent the execution of a callback by returning a score of zero.

The gesture picker executes confidence calculators for all of the gestures in the candidate set and then invokes the associated callback for the gesture with the highest confidence score. In our current implementation it is the responsibility of the developer to ensure that exactly one confidence score is highest. We leave it to future work to build more sophisticated logic into the gesture picker for disambiguating amongst conflicting gestures. Schwarz et al.'s [118] probabilistic disambiguation technique may be one fruitful direction to explore.

One common use of trigger callbacks is to provide visual feedback over the course of a gesture. However, as confidence calculators receive more information over time, the gesture with the highest confidence may change. To prevent errors due to premature commitment to the wrong gesture, developers should ensure that any effects of trigger callbacks on application state are reversible. Developers may choose to write trigger callbacks so that they do not affect global application state or they may create an undo function for each trigger callback to restore the state. Alternatively, Proton supports a *parallel worlds* approach. The developer provides a copy of all relevant state variables in the application. Proton executes each valid callback regardless of confidence score in a parallel version of the application but only displays the feedback corresponding to the gesture with the highest confidence score. When the input stream flushes, Proton commits the the application state corresponding to the gesture with the highest confidence score.

6.3.4 Splitting the Touch Event Stream

Proton recognizes a gesture when the entire touch event stream matches a gesture regular expression. Each time a match is found, Proton executes the callback associated with the gesture expression and flushes the stream. With a single stream, Proton can recognize at most one gesture at a time.

To enable the recognition of simultaneous gestures, the developer can set Proton to split the touch event stream by any custom attribute. The developer registers a split-stream attribute generator with the stream generator, and the stream generator splits the touch event stream by attribute value into substreams. Thus, each substream only contains events with the same split-stream attribute value. Proton then runs a separate gesture matcher on each substream, which allows Proton to recognize multiple simultaneous gestures, as each matcher can detect a gesture. Note that once a touch is assigned to a gesture matcher on a touch-down event, all successive touch events with the same T_{ID} will be sent to the same gesture matcher, ensuring that touch events from the same finger do not get split amongst different gesture matchers. In Section 6.6 we discuss the challenges of detecting multiple simultaneous gestures without stream splitting. In Section 6.7.4 we demonstrate stream splitting with a user ID attribute to enable a multiplayer Pong game.

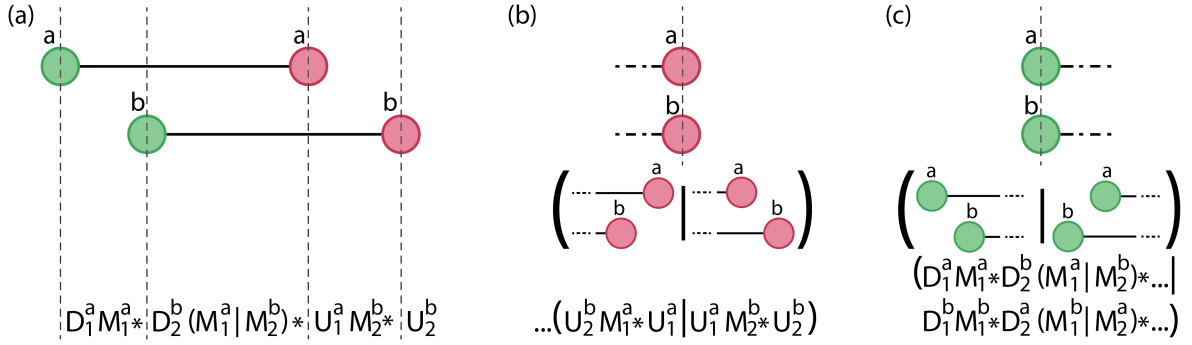


Figure 6.8: Our tablature conversion algorithm sweeps left-to-right and emits symbols each time it encounters a touch-down or touch-up node (vertical dotted lines). We distinguish three cases: (a) non-aligned nodes; (b) aligned touch-up nodes; (c) aligned touch-down nodes.

6.3.5 Tablature to Expression Conversion

To convert a gesture tablature into a regular expression, we process the tablature from left to right. As we encounter a touch-down node, we assign the next available T_{ID} from the priority queue (Section 6.3.2) to the entire touch track. To emit symbols we sweep from left to right and distinguish three cases. When none of the nodes are vertically aligned we output the corresponding touch symbol for each node followed by a repeating disjunction of move events for all active touch tracks (Figure 6.8a). When touch-up nodes are vertically aligned we emit a disjunction of the possible touch-up orderings with interleaved move events (Figure 6.8b). When touch-down nodes are vertically aligned we first compute the remainder of the expressions for the aligned touch tracks. We then emit a disjunction of all permutations of T_{ID} assignments to these aligned tracks (Figure 6.8c). We output the regular expression symbols (,), |, or $*$ as we encounter them in the tablature. If we encounter a local trigger, an arrow placed directly on a touch track, we associate it with only the symbols emitted for its track. If we encounter a global trigger, an arrow placed on its own track, we associate it with all symbols emitted at that step of the sweep.

6.3.6 Static Analysis Algorithm

Two gestures conflict when a string of touch event symbols matches a prefix of both gesture expressions. We call such a string a *common prefix*. We define the regular expression that describes all such common prefixes as the *longest common prefix expression*. Our static analyzer computes the longest common prefix expression for any pair of gesture expressions.

To compute the longest common prefix expression we first compute the *intersection* of two regular expressions. This intersection is a third expression that matches all strings that are matched by both original expressions. A common way to compute the intersection is to first convert each regular expression into a non-deterministic finite automata (NFA) using Thompson's Algorithm [126] and then compute the intersection of the two NFAs.

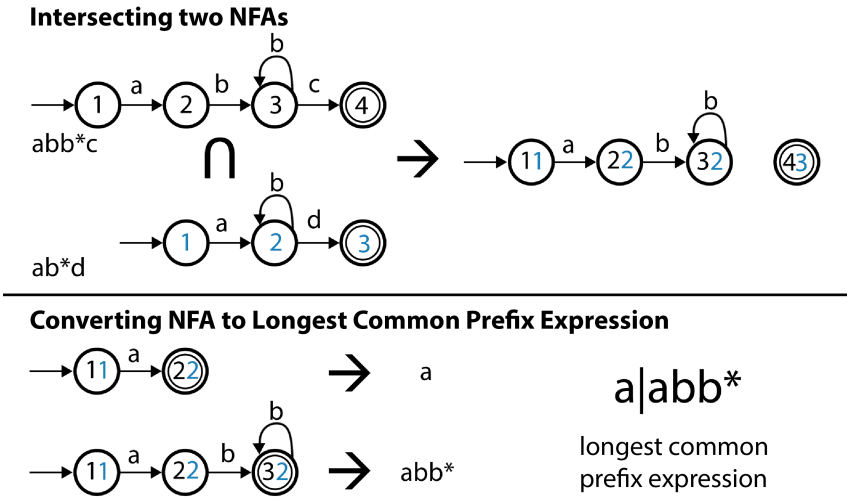


Figure 6.9: Top: The intersection of NFAs for the expressions $abb*c$ and $ab*d$ does not exist because the start state 11 cannot reach the end state 43. Bottom: Treating states 22 and 32 each as end states, converting the NFAs to regular expressions yields a and $abb*$. The longest common prefix expression is the union of the two regular expressions.

To construct the longest common prefix expression we mark all reachable states in the intersection NFA as accept states and then convert this NFA back into a regular expression.

We compute the intersection of two NFAs [123] as follows. Given an NFA M with states m_1 to m_k and an NFA N with states n_1 to n_l, we construct the NFA P with the cross product of states $m_i n_j$ for $i = [1, k]$ and $j = [1, l]$. We add an edge between $m_i n_j$ and $m_{i'} n_{j'}$ with transition symbol r if there exists an edge between m_i and $m_{i'}$ in M and an edge between n_j and $n_{j'}$ in N, both with the transition symbol r. Since we only care about states in P that are reachable from its start state $m_1 n_1$, we only add edges to states reachable from the start state. Unreachable states are discarded. Suppose that m_k and n_l were the original end states in M and N respectively, but that it is impossible to reach the cross product end state $m_k n_l$. In this case the intersection does not exist (Figure 6.9 Top). Nevertheless we can compute the longest common prefix expression. We sequentially treat each state reachable from P's start state $m_1 n_1$ as the end state, convert the NFA back into a regular expression, and take the disjunction of all resulting expressions (Figure 6.9 Bottom). The NFA to regular expression conversion is detailed in Sipser [123].

6.4 Custom Attributes

To add a new custom attribute in Proton, the developer must write and register an attribute generator. On each touch event, the attribute generator receives the touch data reported by the hardware sensors, along with the entire sequence of previous touch symbols from the stream generator. It then computes an attribute value based on this information and

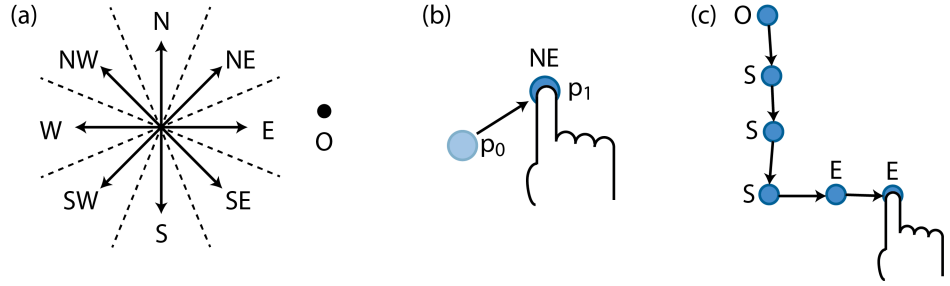


Figure 6.10: (a) The space of directions is divided into eight ranges representing the four cardinal and four ordinal directions. (b) The vector formed by the last two positions is binned to the closest direction. (c) An L-shaped gesture generates south ('S') symbols then east ('E') symbols.

appends it to the current touch event symbol. Since Proton is based on regular expressions composed of discrete touch symbols, the primary constraint on the attributes is that they must take discrete values. Thus, attribute generators are often responsible for quantizing continuous-valued parameters to convert them into attribute values suitable for Proton.

We have implemented five example attribute generators that produce such discrete attributes and demonstrate the flexibility of our approach: 1) a direction attribute for describing touch trajectory, 2) a pinch attribute for detecting when touches move towards one another, 3) a touch area attribute for simulating pressure, 4) a finger orientation attribute for selecting menu items, and 5) a screen location attribute for simulating hand ID and user ID. The direction and screen location attributes are based on touch position and can work with any multitouch device. Our implementations of the touch area and finger orientation attributes require additional touch information which we obtain from a Fingerworks iGesture Pad [133].

6.4.1 Direction Attribute

The direction attribute allows developers to describe a touch trajectory within a Proton gesture expression. Since attribute values must be discrete, we bin the space of all directions into eight ranges representing the four cardinal and four ordinal directions seen in a compass (Figure 6.10a). To generate this attribute we compute the vector $p_1 - p_0$, between the previous touch position p_0 (as given by the previous touch event with the same touch ID) and the current touch position p_1. Our direction attribute generator then returns the direction bin containing the vector (Figure 6.10b). If the touch has not moved beyond a distance threshold, the generator outputs an 'O' value. Our implementation uses a threshold of five pixels.

Trajectory. A sequence of direction attribute values describes a gesture trajectory. For example, in an L-shaped gesture (Figure 6.10c), such as the one used in Eden to load models, the touch first moves in the S direction and then in the E direction. The expression to detect this trajectory is thus: $D_1^O M_1^S M_1^{S*} M_1^E M_1^{E*} U_1^E$. In practice we have found that users often

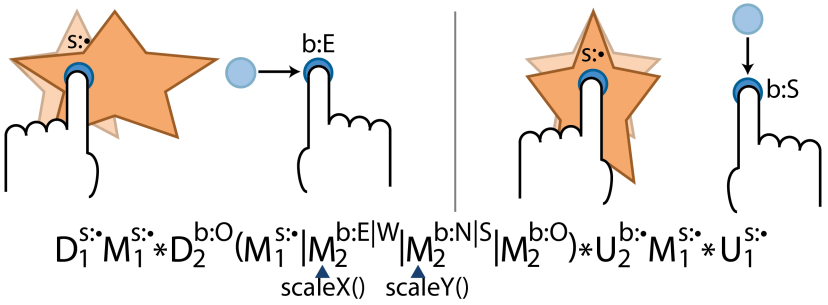


Figure 6.11: Proton continuously tracks the trajectory of the second touch, allowing the developer to provide continuous feedback depending on if the touch moves east-west (scale in x-axis) or north-south (scale in y-axis).

slow down at the beginning of the trajectory or when making the turn from S to E, and the gesture expression fails if the user hesitates in this manner. To allow the user to hold a touch position at any point along the trajectory, we modify the expression to include symbols with direction attribute value 'O': $D_1^O M_1^O * M_1^S M_1^{O|S} * M_1^E M_1^{O|E} * U_1^{O|E}$. Note that this gesture expression requires the user to execute a perfect right-angle turn. We describe how we can use timing to extend such trajectory-based gestures to allow imprecise turns in Section 6.5.

Unlike many recognition systems that detect trajectory at the end of the gesture [116, 139], Proton continuously tracks the trajectory as the user performs the gesture. Thus, developers can provide continuous feedback. For example, a shape manipulation application might include a gesture where one touch selects the shape, and a second touch must move E-W to scale the shape along the x-axis or move N-S to scale the shape along the y-axis with continuous feedback (Figure 6.11). Providing such immediate feedback is an essential feature for direct manipulation interfaces [122].

6.4.2 Pinch Attribute

A pinch in which two or more touches move towards each other is a commonly used gesture in multitouch applications. While the previous direction attribute evaluated the movement of an individual touch, the pinch gesture is based on the relative movements of multiple touches. Our pinch attribute generator computes the average distance between each touch and the centroid of all the touches. It compares this average distance to the average distance computed for the previous touch event and if it decreases the generator assigns the touch the attribute value 'P' for pinching. If the average distance increases it assigns the touch the value 'S' for spreading, and if there is no change in average distance it assigns the value 'N' (Figure 6.12a). We use the pinch attribute to describe a two-touch zoom gesture as shown in Figure 6.12b. Our approach considers all touches together as a whole and cannot distinguish when only a subset of touches are pinching. For example, if two touches are locally pinching, but moving away from a stationary third touch, this attribute may report the touches as

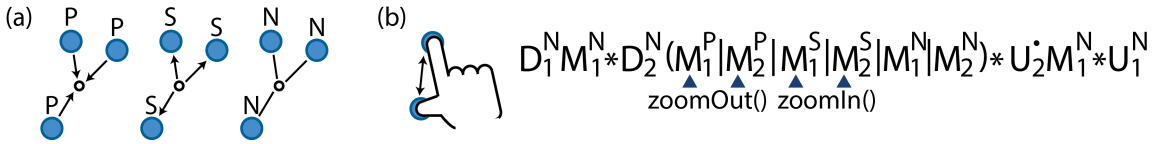


Figure 6.12: (a) Touches are assigned a 'P' when on average the touches move towards the centroid, an 'S' when the touches move away from the centroid, and an 'N' when they stay stationary. (b) A two-touch gesture that zooms out on a pinch and zooms in on a spread.

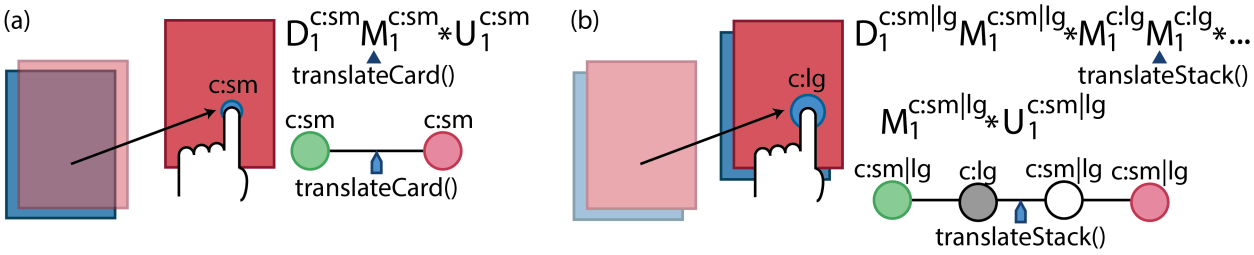


Figure 6.13: (a) A touch with small ('sm') area translates only the topmost card of a stack. (b) A touch with large ('lg') area translates the entire stack.

spreading. An alternative approach is to generate an attribute that encodes the pairwise pinch relationships of all possible touches, so the developer can then specify which pairs of touches must be involved in the pinch.

6.4.3 Touch Area Attribute

Many multitouch devices such as the Fingerworks iGesture Pad [133] report the touch area, the contact area between a finger or hand and the device, as a continuous value. Our attribute generator quantizes the size of the touch area to two discrete values, *small* and *large*. Precisely regulating touch area can be difficult. In practice we found that consistently generating more than two distinct levels of touch area was challenging and therefore limited this attribute to two levels.

Simulating Pressure. Although the iGesture Pad cannot detect pressure, we can use touch area to simulate force, using the approach of ShapeTouch [23]: smaller touch area corresponds to lower pressure and larger area corresponds to stronger pressure. As shown in Figure 6.13a, a touch on a card ('c') with a small ('sm') area $M_1^{c:sm}$ translates the topmost card of a stack, while in Figure 6.13b, a touch on a card with a large ('lg') area $M_1^{c:lg}$ translates the entire stack of cards. One limitation of touch area is that a user's initial touch area starts small before it grows into a large area. Thus, when using the large touch attribute value, the developer should allow the touch to begin and end on a small area. In general, the developer must carefully consider attributes such as area, where a touch must go through lower attribute values to reach higher attribute values.

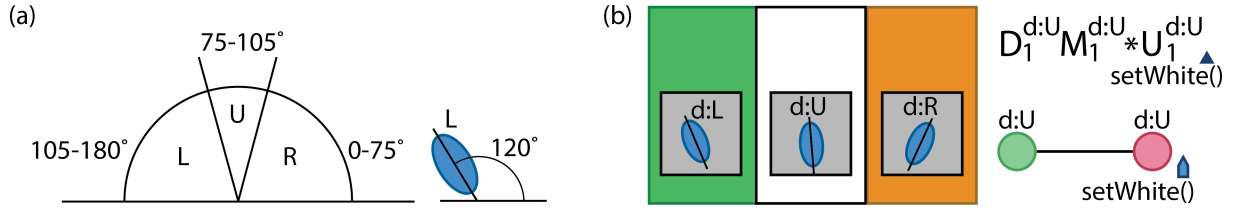


Figure 6.14: (a) The angle of the major axis of a touch is binned into three orientation values. (b) The dial menu ('d') uses orientation to choose the background color of an application.

6.4.4 Finger Orientation Attribute

The Fingerworks iGesture Pad provides a continuous orientation value for each touch in the range 0-180°. We bin the orientations to three levels as shown in Figure 6.14a: up (75-105°), left (>105°), and right (<75°). We define a narrow range for the up bin so users do not have to awkwardly rotate their wrist or fingers from their natural positions to reach the left and right bins. We also minimize the number of orientation levels so that users can easily perform them.

Selecting Menu Items. We use the finger orientation attribute to select from a three-state dial menu (Figure 6.14b). In this example, the dial sets the background screen color: an up orientation assigns a white background, a left orientation assigns a green background, and a right orientation assigns an orange background. This dial menu is very similar to a marking menu [76], but uses finger orientation instead of stroke direction.

6.4.5 Screen Location Attribute

All multitouch devices provide touch position as a continuous value with each touch event. The screen location attribute assigns discrete attribute values to touch positions. Hit-testing is one approach for assigning such discrete values; the attribute value is set to the label of the hit-target, the object directly under the touch point. In addition to using scene objects for hit-testing, we can also define other screen regions to generate attribute values.

Hand Identification. Most multitouch devices cannot detect which hand (left or right) generated a touch event. However, we can simulate hand ID using the screen location attribute as a proxy for hand ID. We divide the screen in half and assign the attribute value 'L' (for left hand) to touches originating from the left side of the screen. Similarly we assign the value 'R' (for right hand) to touches originating from the right side.

We can combine this simulated hand ID attribute with the direction attribute (Figure 6.15) to create an ordered two-handed marking menu (Chapter 4), in which users must start a stroke with the left hand and then start a second stroke with the right hand to select between a large number of menu items. For example, the expression and tablature for the left hand drawing a stroke in the E direction and the right hand drawing a stroke in the W direction are given in Figure 6.15.

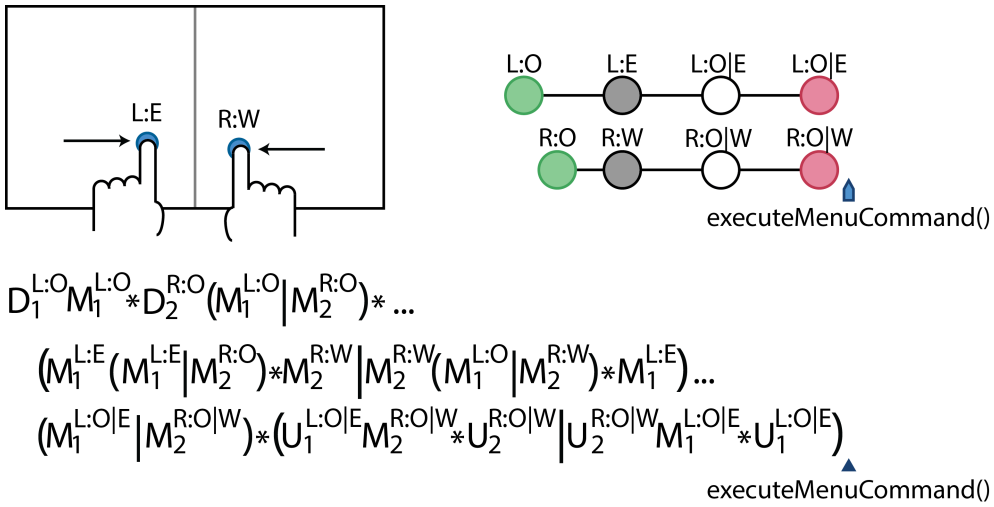


Figure 6.15: To simulate hand identification, touches beginning on the left side belong to the left hand and touches beginning on the right side belong to the right hand. An ordered two-handed marking menu can be described by adding the direction attribute.

6.4.6 Designing Custom Attributes

We have implemented example applications for the direction, pinch, touch area, finger orientation, and screen location attributes. Based on our experience building these applications, we distill several design considerations for creating custom attributes.

Levels and Ranges of Attribute Values. Developers should base the number of levels of an attribute on the specificity needed by the application. More levels provide developers with finer-grain control over gesture specifications. However, more levels also require extra effort to author: developers may have to carefully write complex disjunctions of attribute values in the expressions.

The range of input values binned to each attribute value affects users' ability to perform actions that correspond to each attribute value. For example, if the range of a direction value is small, users may find it difficult to accurately draw a stroke in that particular direction. Noise or variation in user performance may cause matching to fail for attribute values with narrow ranges. Developers should choose ranges such that users can reliably perform actions for each attribute value.

Attribute Value Traversal. Certain attributes such as touch area will require traversal through lower attribute values to reach higher attribute values. Developers should be cognizant of this possibility and design gestures that allow for such traversal. Additional developer tools could aid in integrating these attributes in gestures.

No Asynchronous Attribute Values. Developers must assign an attribute value to each touch event as soon as it is emitted, within the time interval defined by the stream generator's reporting rate. While the history of a touch is available, developers must not

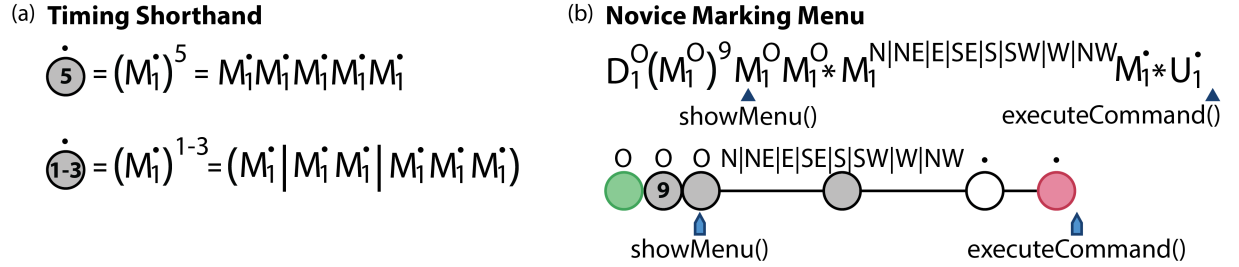


Figure 6.16: (a) Shorthand for specifying timing in tablature. (b) Novice marking menu using timing notation to specify a touch hold.

wait for future information to determine an attribute value. Relying too much on touch history reduces the immediacy in which Proton can distinguish gestures from each other and detect conflicts.

Application Independent Attributes. To create attributes that can be reused across multiple applications, attribute generators should only rely on information contained in the touch stream received from a hardware device and should not access application-specific information. For example, once defined, the direction attribute can be used in any application with gestures that require directional specification.

6.5 Timing

While taps, holds and flicks are common multitouch gestures, the basic touch event sequence for all three is exactly the same. To distinguish among these three gestures, the gesture recognizer must have access to timing information. We introduce timing to Proton by adding the constraint that the stream generator reports touch events at a fixed time interval; we use $\frac{1}{30}s$. Thus each touch symbol $M_1^\bullet$ also represents a unit of time t and a sequence of k such symbols represents a time duration of kt. To express gesture timing, the developer can replace $M_1^\bullet*$, which matches a touch movement of any duration, with a fixed-length sequence of $M_1^\bullet$ symbols. This sequence matches only when the touch movement lasts for the corresponding length of time.

Writing out a fixed-length sequence of move symbols can be tedious, so we introduce a shorthand for specifying the number of successive touch-move events using the notation

$$\left(M_{T_{ID}}^{A_1:A_2:A_3\cdots}\right)^{t_1-t_2}$$

which generates the expression that matches t_1 to t_2 successive $M_{T_{ID}}^{A_1:A_2:A_3\cdots}$ events. The t_2 parameter is optional. Proton expands the shorthand into t_1 consecutive move symbols if t_2 is not specified. It generates the disjunction of t_1 consecutive move symbols to t_2 move symbols if t_2 is specified. For example, a touch and hold that lasts at least five consecutive move events is expressed as $D_1^\bullet(M_1^\bullet)^5 M_1^\bullet* U_1^\bullet$, which expands to $D_1^\bullet M_1^\bullet M_1^\bullet M_1^\bullet M_1^\bullet M_1^\bullet M_1^\bullet* U_1^\bullet$. A tap

of one to five move events is expressed as $D_1^\bullet(M_1^\bullet)^{1-5}U_1^\bullet$, which expands to $D_1^\bullet(M_1^\bullet|M_1^\bullet M_1^\bullet|...$ $M_1^\bullet M_1^\bullet M_1^\bullet M_1^\bullet M_1^\bullet)U_1^\bullet$. We also update the tablature with timing notation as shown in Figure 6.16a. The developer can specify a range t_1 to t_2 within the gray move nodes.

Using timing to detect a hold, we can design a marking menu [78] for novice users that visually displays the menu items if the user holds down a touch for $\frac{1}{3}$ of a second. We use the direction attribute described in Section 6.4.1 and a sequence of 10 touch-move symbols to specify the $\frac{1}{3}s$ duration of the hold with the expression shown in Figure 6.16b. We associate a menu drawing callback with the tenth M_1^O.

Our previous L-shaped trajectory example requires the user to make a perfect right-angle turn from the S direction to the E direction (Section 6.4.1). We can use timing to allow the user to momentarily move in any direction during the turn using the expression $D_1^O M_1^{O*} M_1^S M_1^{O|S*}(M_1^\bullet)^{1-5} M_1^E M_1^{O|E*} U_1^{O|E}$. The timing is specified in the symbol, $(M_1^\bullet)^{1-5}$, which gives the user up to a $\frac{1}{6}s$ window to make a less precise turn.

To capture timing between taps, Proton utilizes a user-definable timeout. The timing mechanism could be extended to capture timing between taps, by emitting a new event symbol that represents zero touches at the system framerate. Similar to specifying the duration of touch-move events, the developer could use this new symbol to specify the duration between taps.

6.6 Touch Group Permutations

To handle multiple simultaneous gestures without stream splitting via an attribute generator, the gesture matcher would need to consider how to assign each distinct touch to a gesture. This space of possible assignments grows exponentially by the number of touches and gestures. The developer then must decide which assignment best interprets the intention of the users.

The number of possible touch groups depends on the number of ways the set of touches can be partitioned. An integer partition is a way to break up an integer into the sums of positive integers. A single user can apply up to ten touches to a multitouch device, so for the integers 1 through 10, the number of integer partitions are 1, 2, 3, 5, 7, 11, 15, 22, 30, and 42 respectively. For example, the number 4 has five integer partitions: (1,1,1,1), (1,1,2), (1,3), (2,2), and (4).

With respect to touches, the partition (1,1,2) corresponds to dividing four touches into three groups: two groups of one touches and one group of two touches. However, each touch is distinct, so there is more than one way to construct these three groups. There are six in fact. Take the four touches t_1, t_2, t_3, and t_4. They can be grouped into the partition (1,1,2) in the following ways:

$$(t_1, t_2, t_3t_4), (t_1, t_3, t_2t_4), (t_1, t_4, t_2t_3), (t_2, t_3, t_1t_4), (t_2, t_4, t_1t_3), (t_3, t_4, t_1t_2)$$

The equation for computing the number of permutations per partition is:

$$\frac{n!}{(x_1!...x_g!)(y_1!...y_d!)}$$

where n is the number of touches, x_i is the size of each of the g groups in a partition, and y_j is the count of each of the d distinct group sizes.

For the previous example, there are four touches so $n = 4$. The partition has three groups 1, 1, and 2, so $x_1 = 1$, $x_2 = 1$, and $x_3 = 2$. Finally there are two groups of size 1 and one group of size 2. Thus, $y_1 = 2$ and $y_2 = 1$ respectively.

$$\frac{n!}{(x_1!...x_g!)(y_1!...y_d!)} = \frac{4!}{(1!1!2!)(2!1!)} = 6$$

These touch group permutations do not take into account the gesture set. It is likely that a touch grouping within permutations can match multiple gestures. For example a grouping of two touches is a possible match for both the two-touch rotation and scale example gestures. Thus, a permutation also must consider which gesture to match for each group of touches. However, if none of the gestures uses n touches, then permutations that contain a group of n touches is not a possible grouping.

6.7 Applications

To demonstrate the expressivity of our framework, we implemented four proof-of-concept applications, each with a variety of gestures. They all use the hit-target attribute.

6.7.1 Application 1: Shape Manipulation

Our first application is inspired by Eden (Chapter 5). It allows users to manipulate and lay out shapes in 2D (Figure 6.17). The user can translate, rotate, scale and reflect the shapes. To control the canvas, the user holds a quasimode [112] button and applies two additional touches to adjust pan and zoom. To add a shape, the user touches and holds a shape icon in a shape catalog and indicates its destination with a second touch on the canvas. To delete a shape, the user holds a quasimode button and selects a shape with a second touch. To undo and redo actions, the user draws strokes in a command area below the shape catalog.

Succinct Gesture Definitions

We created eight gesture tablatures, leaving Proton to generate the expressions and handle the recognition and management of the gesture set. We then implemented the appropriate gesture callbacks and confidence calculators. We did not need to count touches or track gesture state across event handlers.

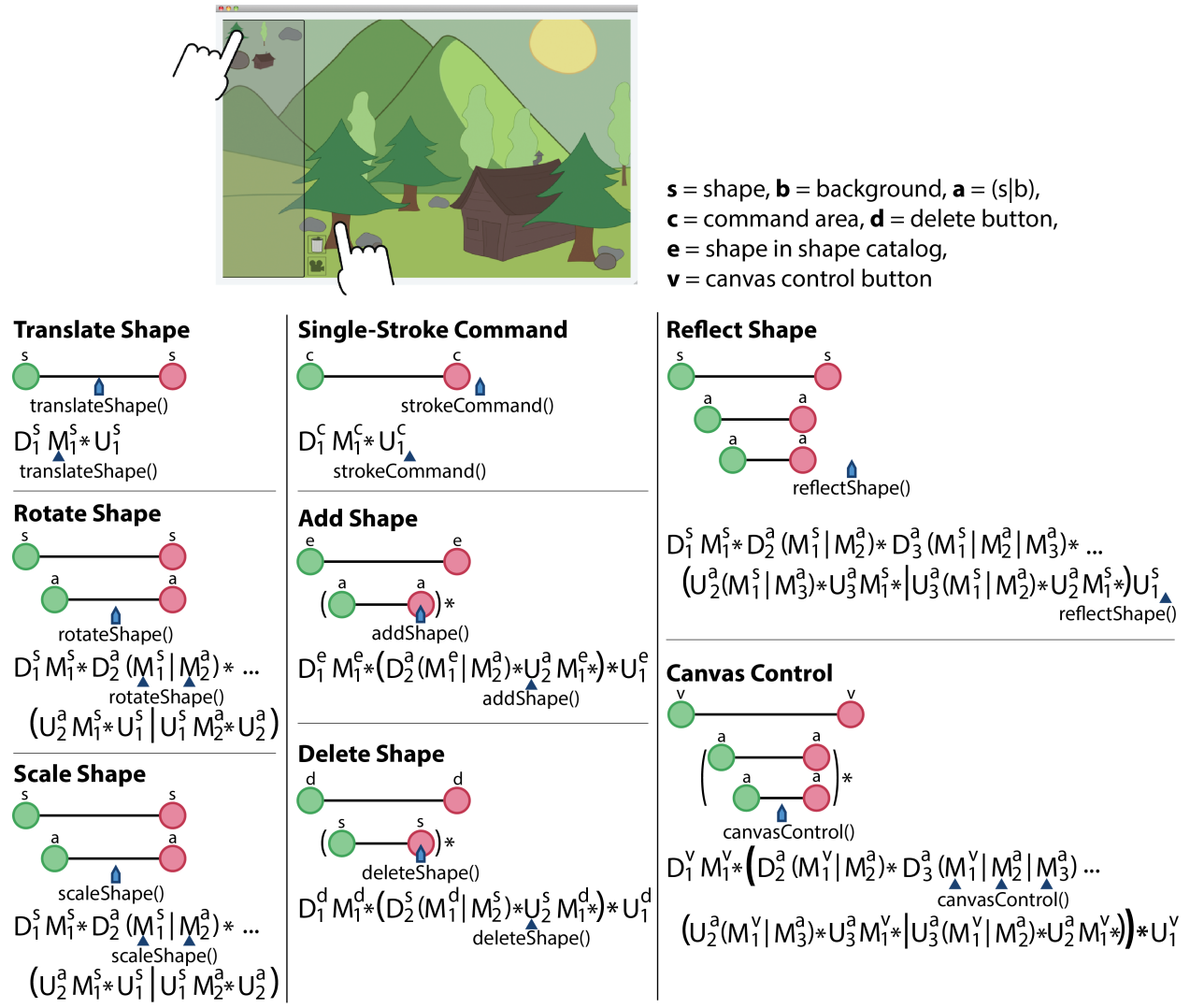


Figure 6.17: The shape manipulation application includes gestures for 2D layout, canvas control, and shape addition and deletion through quasimodes.

Many of our tablatures specify different hit-targets for different touches in a gesture. For example, the Delete gesture requires the first touch to land on the delete button and the second touch to land on a shape. Proton enables such quasimodes without burdening the developer with maintaining application state. A hit-target attribute value does not have to correspond to single target objects: the Rotate, Scale, and Reflect gestures allow the second touch to land on any object including the background. We also specified the order in which touches should lift up at the end of a gesture. While the Rotate and Scale gestures permit the user to release touches in any order, modal commands require that the first, mode-initiating touch lift up last. Finally, we specified repetitions within a gesture using Kleene stars. For

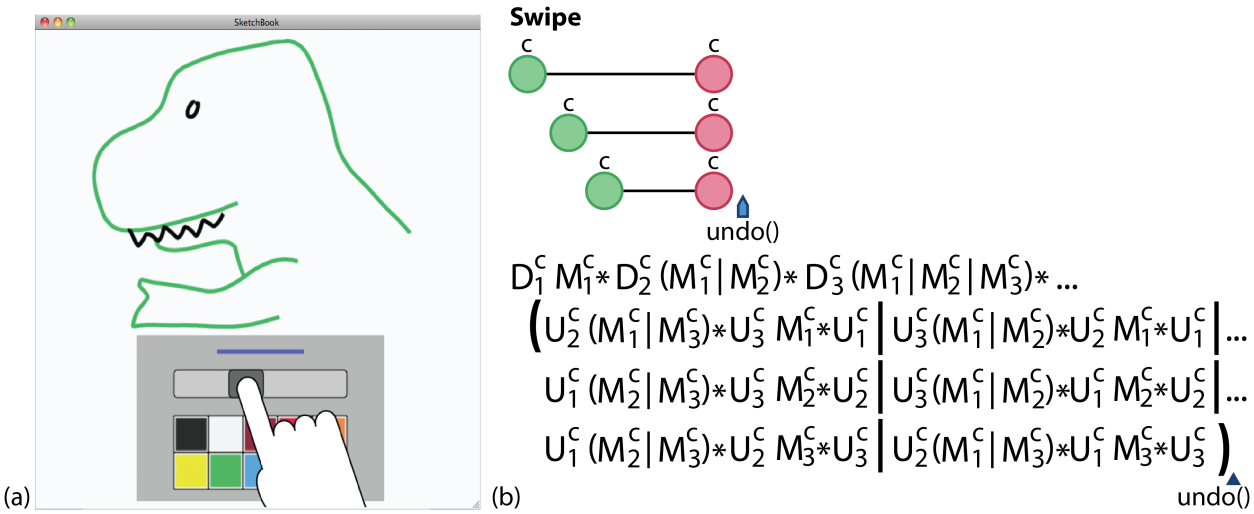


Figure 6.18: (a) In the sketching application's palette, the user adjusts brush parameters through predefined widgets. (b) Aligned touch-up nodes for the swipe tablature generate all six touch-up sequences.

example, the second touch in the Delete gesture is grouped with a Kleene star, which allows the expression to match any number of taps made by the second touch.

Static Analysis of Gesture Conflicts

Proton's static analyzer reported that all six pairs of the shape manipulation gestures (Translate, Rotate, Scale, and Reflect) conflicted with one another. Five of the conflicts involved prefix expressions only, while the sixth conflict, between Rotate and Scale, showed that those two gestures are identical. We wrote confidence calculators to resolve all six conflicts.

The static analyzer also found that all shape manipulation gestures conflict with Translate when only one touch is down. We implemented a threshold confidence calculator that returns a zero confidence score for Translate if the first touch has not moved beyond some distance. Similarly, Rotate and Scale only return a non-zero confidence score once the second touch has moved beyond some distance threshold. After crossing the threshold, the confidence score is based on the touch trajectory.

6.7.2 Application 2: Sketching

Our second application replicates a subset of the gestures used in Autodesk's SketchBook [8] application for the iPad. The user can draw using one touch, manipulate the canvas with two touches, and control additional commands with three touches. A three-touch tap loads a palette (Figure 6.18a) for changing brush attributes and a three-touch swipe executes undo and redo commands, depending on the direction.

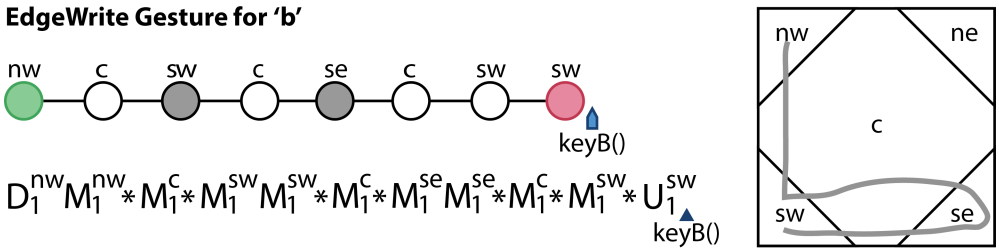


Figure 6.19: EdgeWrite gestures change hit-targets multiple times within a touch track. The gesture for the letter 'b' is shown.

In this application, the generated expressions for Load Palette and Swipe are particularly long because these three-touch gestures allow touches to release in any order. We created tablatures for these gestures by vertically aligning the touch-up nodes and Proton automatically generates expressions containing all possible sequences of touch-up events (Figure 6.18b).

Proton includes a library of predefined widgets such as sliders and buttons, in which the expressions have already been defined, and the developer need only associate the target widget with the gesture expression. We used this library to create the brush attribute palette. For example, to add color buttons into the palette, we created new instances of the button press widget, which consists of a button and corresponding button press gesture. We gave each button instance a unique name, e.g., *redButton*, and then assigned *redButton* as the hit-target attribute value for the touch events within the expression for the button press gesture. We also defined a change color callback to be executed on a successful button press.

We also implemented a soft keyboard for text entry. The keyboard is a container where each key is a button subclassed from Proton's built-in button press widget. We associated each key with its own instances of the default button press gesture and callback. Thus, we created 26 different buttons and button press gestures, one for each lower-case letter. Adding a shift key for entering capital letters required creating a gesture for every shift key combination, adding 26 additional gestures.

6.7.3 Application 3: EdgeWrite

Our third application re-implements EdgeWrite [140], a unistroke text entry technique where the user draws a stroke through corners of a square to generate a letter. For example, to generate the letter 'b', the user starts with a touch down in the NW corner, moves the finger down to the SW corner, over to the SE corner, and finally back to the SW corner. Between each corner, the touch moves through the center area c (Figure 6.19 Left). We inserted explicit touch-move nodes (gray and white circles) with new hit-target attribute values into the gesture tablature to express that a touch must change target corner objects as it moves. The gesture tablature for the letter 'b' and its corresponding regular expression are shown in Figure 6.19 Right.

Our EdgeWrite implementation includes 36 expressions, one for each letter and numbers

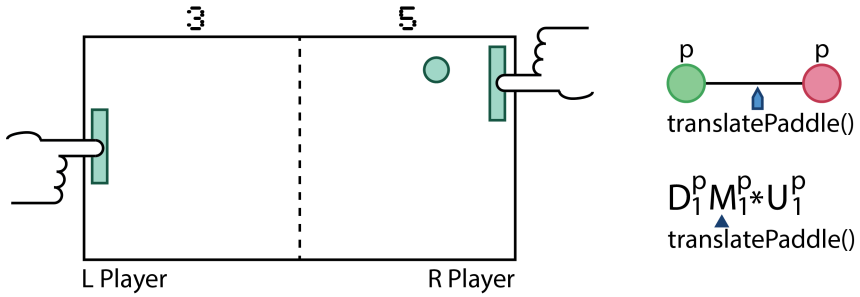


Figure 6.20: In this Pong game, the touch stream is split so one gesture matcher can process touches from the left player and a second gesture matcher can process touches from the right player. Both gesture matchers use the same gesture for controlling the paddle ('p').

0-9. Our static analyzer found that 18 gestures conflict because they all start in the *NW* corner. The number of conflicts drops as soon as these gestures enter a second corner: 9 conflicts for the *SW* corner, 2 for the *SE* corner, and 7 for the *NE* corner. Our analyzer found that none of the gestures that started in the *NW* corner immediately returned to the *NW* corner which suggests that it would be possible to add a new short *NW*–*NW* gesture for a frequently used command such as delete. Similarly the analyzer reported conflicts for strokes starting in the other corners. We did not have to resolve these conflicts because the callbacks execute only when the gestures are completed and none of the gestures are identical. However, such analysis could be useful when designing a new unistroke command to check that each gesture is unique.

6.7.4 Application 4: Pong

Some multitouch devices, such as the DiamondTouch [30, 89], directly provide a different user ID for each person interacting with the device. For multitouch devices that do not provide such identification, we can use screen location to simulate a user ID attribute. For example in a two-player Pong game (Figure 6.20), touches originating on the left side of the screen correspond to one user, and touches originating on the right correspond to a second user. Using a single stream and gesture matcher would restrict the players so that only one of them could move their paddle at any time. Splitting the stream by the location-based user ID removes this restriction. Since each stream has its own gesture matcher, the system can recognize paddle control gestures from both players at the same time. In this example the developer could provide the same set of attributes and gesture expressions to both gesture matchers. However, Proton also allows developers to register different sets of attributes and gesture expressions to each gesture matcher.

6.8 User Study

To help us understand whether developers can benefit from the gesture matching and conflict detection provided by Proton, we conducted a user study evaluating how quickly and accurately developers comprehend gestures described using regular expressions, tablatures, and iOS-style [6] event-handling code. We divided the study into two parts, with the first part focusing on basic gestures that involved only the sequence of touch events and the second part including trajectory-based gestures.

We recruited 12 participants (10 male, 2 female, ages between 20 and 51) who were all experienced programmers. Each participant performed both parts of the study and each part contained three blocks. Each block focused on one of the three gesture representations: *tablature*, *expression*, and *iOS*. We counterbalanced the orderings of the gesture representations so that each of the six possible orderings was performed by two participants.

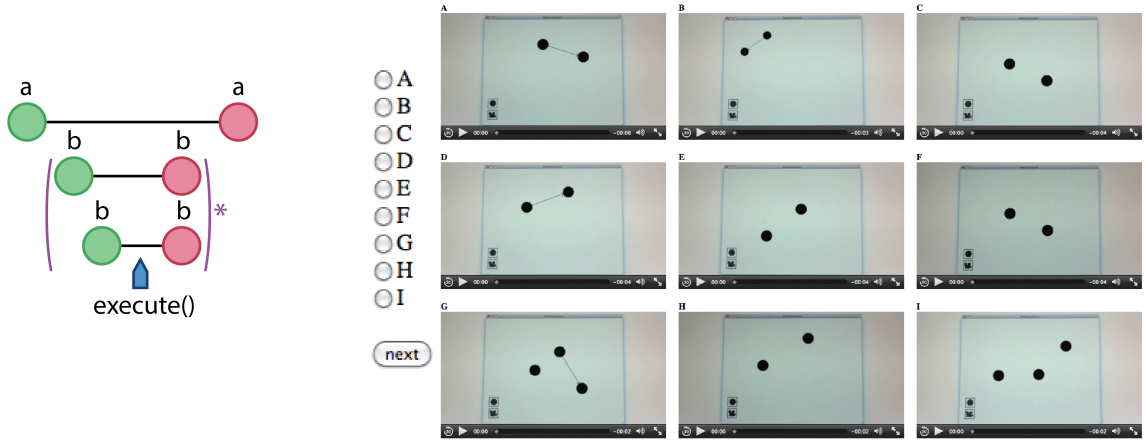
6.8.1 Part 1: Basic Touch Event Sequences

The first part of the study tested how each gesture representation affects the participant's understanding of basic touch event sequences in a gesture. Gestures are often dependent on the target of the touches, so we also included a single hit-target attribute: the type of target hit by the touch.

At the start of each block, we gave the participant a tutorial on how to interpret a multitouch gesture using the block's gesture representation. We then asked the participant to perform five gesture identification trials. The gestures were chosen such that each block covered a range of different gestures with one, two, and three touches. For each trial, we presented the participant with a gesture written in the block's gesture representation and a set of nine videos of a user performing gestures (Figure 6.21). We asked the participant to identify which video matched the gesture. To mitigate learning effects, we reordered the nine videos between blocks. Figure 6.22 contains screenshots of a node-linking gesture used in the study.

For each trial, we were interested in only the time spent understanding the gesture. However, participants would often spend significant time rewatching and searching videos for the correct one, after having already understood the gesture. Thus, we measured time to completion of a trial as the total trial time minus the video-playing time. We also checked whether the participant chose the correct video.

Results. The average times to completion (Figure 6.23 Left) for identifying a gesture were 23.50 seconds for tablature, 49.25 seconds for expression, and 110.99 seconds for iOS event-handling (one way ANOVA $F_{2,22}$=55.37, p<.001; all pairwise comparisons with Bonferroni correction, p<.05, were also significant). Tablature was 2.1 times faster than expressions and 4.7 times faster than event-handling. The average accuracies were 100% for tablature, 93.3% for expression, and 95% for iOS event-handling, but the differences were not significant ($F_{2,22}$=1.20, p=.320).



$$D_1^a\, M_1^a * \left(D_2^b\,(M_1^a \mid M_2^b) * D_3^b (M_1^a \mid M_2^b \mid M_3^b) * \left(U_2^b (M_1^a \mid M_3^b) * U_3^b \mid U_3^b (M_1^a \mid M_2^b) * U_2^b\right) M_1^a *\right) * U_1^a$$

execute()

```
_state = GesturePossible;

touchesDown(Array *touches, Array *allTouches)
  if(allTouches->count() > 3)
    _state = GestureFailed;
  else if(allTouches->count() == 1)
    if(touches[0]->target() != 'a')
      _state = GestureFailed;
  else
    if(touches[0]->target() != 'b')
      _state = GestureFailed;

touchesMove(Array *touches, Array *allTouches)
  for(i = 0; i < touches->count(); i++)
    if(touches[i]->touchId() == 0 && touches[i]->target() != 'a')
      _state = GestureFailed;
      return;
    else if(touches[i]->touchId() != 0 && touches[i]->target() != 'b')
      _state = GestureFailed;
      return;
  if(allTouches->count() == 3)
    execute();

touchesUp(Array *touches, Array *allTouches)
  if(allTouches->count() == 1)
    if(touches[0]->touchId() == 0 && touches[0]->target() == 'a')
      _state = GestureRecognized;
    else
      _state = GestureFailed;
  else
    if(touches[0]->touchId() != 0 && touches[0]->target() == 'b');
    else
      _state = GestureFailed;
```

Figure 6.21: In Part 1, the participant is shown a gesture and the participant must identify the matching video.

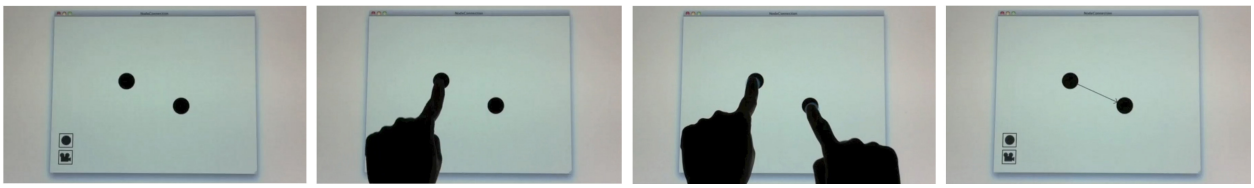


Figure 6.22: Screenshots of a video depicting a two-touch gesture for linking two nodes.

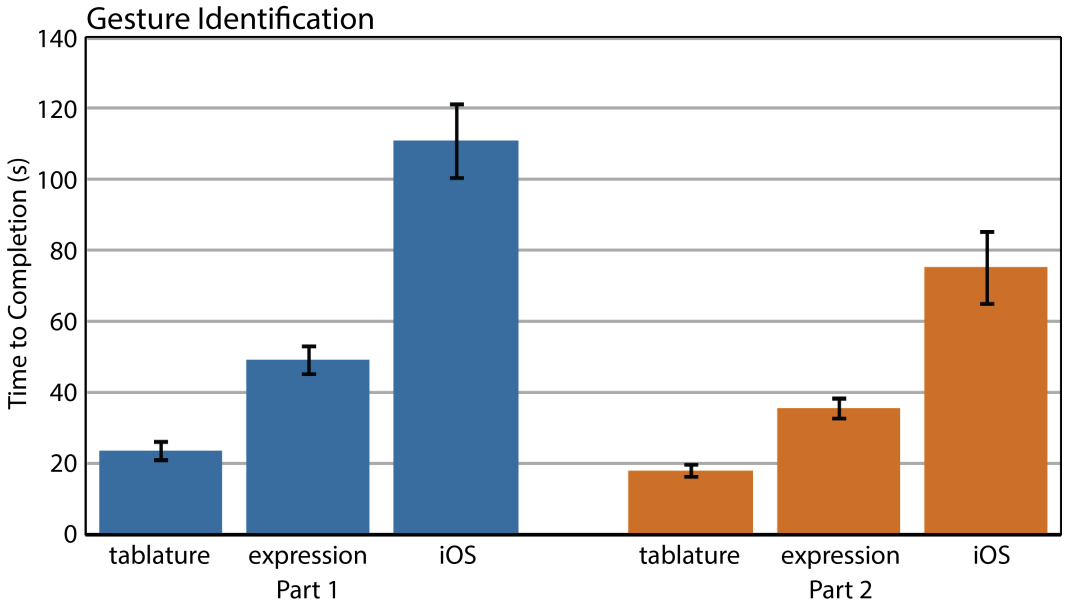


Figure 6.23: The average time to completion for identifying a gesture in Part 1 and Part 2. Standard error bars are shown.

6.8.2 Part 2: Trajectory Gestures

In the second part of the study we asked participants to identify gestures that use both hit-target attribute and the direction attribute for specifying trajectory. For each block, we asked participants to perform three trials of gesture identification. In each trial, we presented a gesture and a set of four images, each depicting the gesture trajectory as red directed paths drawn on a target. Participants chose which trajectory would be recognized by the given gesture (Figure 6.24). As in the first task, for each trial we checked for correctness and measured the time of completion.

Results. The average times to completion (Figure 6.23 Right) for identifying a gesture were 17.82 seconds for tablature, 35.49 seconds for expression, and 75.29 seconds for iOS event-handling ($F_{2,22}$=21.30, p<.001; all pairwise comparisons were also significant). Tablature was 2.0 times faster than expressions and 4.2 times faster than event-handling. All participants had 100% accuracy rate in identifying the gestures for all three representations.

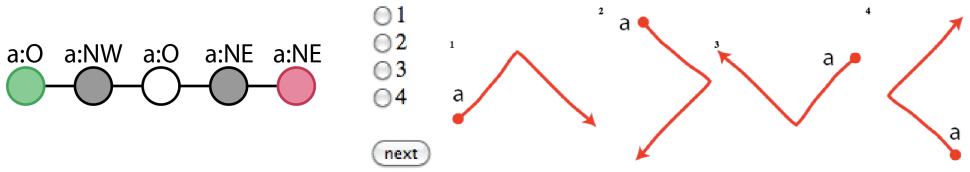


Figure 6.24: In Part 2, the participant is shown a gesture and must identify the matching trajectory.

6.8.3 Qualitative Results

In a post-study survey we asked participants to rate the representations for ease of comprehension. On a Likert scale of 1 (easiest) to 5 (hardest), the average score was 1.33 for tablature, 2.92 for expression, and 4.13 iOS event-handling. A Kruskal Wallis test revealed a significant effect of condition on Likert ratings (H=26.4, 2df, p<.001). Mann-Whitney tests showed that all pairwise comparisons were also significant. When asked which representation they would most like to design multitouch gestures with, 11 of the 12 participants preferred tablature and the remaining participant preferred expression.

6.8.4 Discussion

Our results show that users are faster at identifying gestures in tablature form than in expression or event-handling form. These results indicate that users can quickly learn and understand gesture tablatures, which suggests users can also more quickly build and maintain multitouch gestures written in tablature. Our results confirm that tablature is an effective graphical representation for the underlying regular expression representation of gestures. Our post-study survey also suggests that users prefer implementing multitouch gestures with tablature over standard touch event-handling.

Participants generally preferred tablature as they found it obvious how "to express temporal order" of the touches. They could mime the touch actions as they read the tablature. They saw similar benefits with regular expressions, but were concerned that the "complexity of the expressions could easily explode." In contrast with tablature and expressions, participants found it difficult to keep track of the gesture state in disparate event-handlers, which required "too much jumping around the code" and "mental book-keeping." However, participants felt by having direct access to touch events, event-handling is "ostensibly more flexible."

6.9 Conclusion

As developers continue to build more multitouch applications, and users spend more time interacting with them, our understanding of multitouch input will continue to improve. We developed Proton from the recognition that current event-handling methods work well for

mouse input, but not for multitouch. Proton directly addresses the challenges of building and managing multitouch gesture sets. As researchers and application developers continue to evolve the way we use multitouch input, multitouch developer tools should also evolve.

Chapter 7

Conclusions and Future Work

In this dissertation we investigated the benefits of multitouch input and the design and development of multitouch applications. We conducted a pair of user studies that contribute to the understanding of direct-touch, bimanual, and multifinger input. We then leveraged multitouch benefits to design a fully functional multitouch application for a professional content-creation task. From our experience building a complete multitouch application, we identified the challenges of building multitouch gesture sets and developed a new multitouch framework to aid in the creation and management of these gesture sets.

We summarize our contributions in more detail in Section 7.1 and we present several avenues of future work in Section 7.2.

7.1 Contributions

In our multitarget selection study, we showed that users can select targets two times faster with multitouch input than with mouse input. Since we compared direct-touch, bimanual, and multifinger input in a single study, we were also able to separate the performance benefit of each of these input techniques. We found that direct-touch input provides the majority of the speed improvement (83%), and bimanual interaction using one finger on each hand accounts for the remaining speedup. Users prefer using the index and middle fingers for selection. Multifinger input provided no additional benefit and may in fact reduce targeting accuracy. Tasks that require the selection of many targets such as file or photo reorganization and the grouping and arrangement of graphical objects can benefit from the performance advantage of multitouch target selection.

Our second study continued the investigation of bimanual interaction using one finger per hand, but for drawing directional strokes in our novel two-handed marking menus. We showed that for the two-handed simultaneous marking menu, users can draw pairs of strokes simultaneously 10-15% faster than drawing one stroke at a time. Our two-handed ordered menu exhibit less motion overlap between hands, but provide twice as many menu options as the two-handed simultaneous marking menu with the same number of strokes. Our

marking menus are an effective menu selection technique and also show the value of bimanual interaction supported by multitouch input.

We then demonstrated that a complete multitouch application can outperform a mouse and keyboard application for a professional content-creation task. We developed Eden, a multitouch application for building sets for computer-animated films, which an expert set construction artist found to be more effective than Maya, a mouse and keyboard application currently used by artists. We found that focusing on supporting one operation at a time allowed us to design simple, memorable gestures that split the workload across two hands. We presented a set of design guidelines and lessons learned from developing Eden, which multitouch application developers can apply to building multitouch applications for other professional workflows.

As multitouch hardware becomes more readily available and the demand for multitouch applications increases, developers would benefit from improved tools to build such applications. We designed and developed Proton, a declarative multitouch framework that allows developers to specify gestures as regular expressions, which provide automatic gesture matching and static analysis of conflicts. In addition, we introduced gesture tablature, a graphical notation that simplifies the creation of multitouch gestures. We showed that users can interpret gesture tablature four times faster than traditional event-handling pseudocode.

Our dissertation contributes to the understanding of multitouch input, demonstrates the viability of multitouch applications for professional users, and facilitates their development through a novel declarative multitouch framework.

7.2 Future Work

Touch Precision: Eden demonstrated the viability of multitouch input for the specific workflow of organic set construction. However, Eden also demonstrated the drawbacks of touch input, including the fat finger problem [110] and occlusion [127]. Even organic set construction, which requires less precision than general set construction, was not immune to these drawbacks. A critical research path in pushing the use of multitouch into more application areas is addressing these precision drawbacks. If touch precision continues to be problematic, then it would be constructive to identify general attributes of workflows that make them conducive to improved user performance through multitouch. Understanding when multitouch is appropriate as input instead of the mouse can inform the design of hybrid multitouch and mouse workstations, as multitouch and mouse input are not mutually exclusive.

Automatic Relayout for Multitouch: To best leverage multitouch input, we designed Eden from the ground up. This process required much creativity and many iterations while consulting with a target expert user. A quicker path to gaining the benefits of direct-touch input and bimanual interaction is to automatically relayout mouse-based interfaces for use on a multitouch workstation. Gajos et al. [42] automatically adapted mouse interfaces for various devices including single-touch screens. Small widgets designed for mouse input

were enlarged for touch input. Future work can investigate techniques for adapting interfaces for full multitouch devices of varying display sizes. For example, widgets located at the top of a mouse-based interface can be placed towards the user on a large horizontal multitouch display, to reduce the distance required for the hands to reach them. Although a converted interface may not approach the sophistication of a completely redesigned multitouch interface, it can be a cheap way to bridge users from mouse-based workstation to multitouch workstations.

Multitouch Ergonomics: Including multitouch input as part of a user's workflow has the potential to relieve users of repetitive strain injury (RSI) issues from using the mouse. However, if multitouch becomes a significant component to a user's workflow, then it is important to understand the long-term physical effects and potential RSI issues of multitouch. Working with multitouch displays requires a different set of ergonomic considerations than with traditional mouse and keyboard interfaces. Along with wrist motions, using multitouch also involves both small finger motions and large arm motions. In addition, the form factor of the device affects both the posture and the muscle groups required of the user. Multitouch devices come in varying sizes, from handheld mobile devices to monitors for the desktop to large collaborative screens. Large displays also come in varying orientations. Long term user studies will help illuminate the RSI implications of multitouch.

Other Gestural Input Techniques: Proton focused on multitouch interaction, but there are other gestural devices that still rely on standard event-handling to process multiple streams of parallel events. Multitouch devices report the contact of multiple fingers on 2D surfaces, while devices like the Kinect [145] reports multiple joint positions of the entire human body in 3D space. In-the-air 3D hand gestures [130] require the tracking and processing of multiple hand joint positions. Even smartphones report data in addition to touches, such as the acceleration and orientation of the device. With the right custom attributes and extensions, Proton might be able to support the detection of gestures for these input techniques.

Bibliography

[1] 3Dconnexion. *SpaceNavigator*. `http://www.3dconnexion.com`.

[2] 3M. *Multi-Touch Displays*. `http://solutions.3m.com/wps/portal/3M/en_US/TouchSystems/TouchScreen/Solutions/MultiTouch`.

[3] A. Adya, J. Howell, M. Theimer, W. J. Bolosky, and J. R. Douceur. "Cooperative task management without manual stack management". In: *Proceedings of the USENIX Annual Technical Conference*. USENIX Association, 2002, pp. 289–302.

[4] C. Appert and M. Beaudouin-Lafon. "SwingStates: adding state machines to the swing toolkit". In: *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2006, pp. 319–322.

[5] C. Appert and S. Zhai. "Using strokes as command shortcuts: cognitive benefits and toolkit support". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2009, pp. 2289–2298.

[6] Apple. *iOS*. `http://developer.apple.com/technologies/ios`.

[7] Autodesk. *Maya*. `http://www.autodesk.com/`.

[8] Autodesk. *SketchBook Pro*. `http://usa.autodesk.com/adsk/servlet/pc/item?siteID=123112&id=15119465`.

[9] G. Bailly, E. Lecolinet, and L. Nigay. "Flower menus: a new type of marking menu with large menu breadth, within groups and efficient expert mode memorization". In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. ACM, 2008, pp. 15–22.

[10] R. Balakrishnan and P. Patel. "The PadMouse: facilitating selection and spatial positioning for the non-dominant hand". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1998, pp. 9–16.

[11] R. Balakrishnan and K. Hinckley. "Symmetric bimanual interaction". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2000, pp. 33–40.

[12] R. Balakrishnan and K. Hinckley. "The role of kinesthetic reference frames in two-handed input performance". In: *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology*. ACM, 1999, pp. 171–178.

[13] W. C. Barnert. *A comparison of one-handed and two-handed direct and indirect computer interaction*. Tech. rep. Medford, Mass.: Department of Computer Science, Tufts University, Nov. 2005.

[14] H. Benko, A. Wilson, and P. Baudisch. "Precise selection techniques for multitouch screens". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2006, pp. 1263–1272.

[15] F. Bevilacqua, B. Zamborlin, A. Sypniewski, N. Schnell, F. Guédy, and N. Rasamimanana. "Continuous realtime gesture following and recognition". In: *Gesture in Embodied Communication and Human-Computer Interaction*. Vol. 5934. Lecture Notes in Computer Science. 2010, pp. 73–84.

[16] E. Bier. "Snap-dragging in three dimensions". In: *Symposium on Interactive 3D Graphics* 24 (2 1990), pp. 193–204.

[17] E. Bier, M. Stone, K. Pier, W. Buxton, and T. DeRose. "Toolglass and magic lenses: the see-through interface". In: *Proceedings of ACM SIGGRAPH*. ACM, 1993, pp. 73–80.

[18] A. Blackwell. "Swyn: a visual representation for regular expressions". In: *Your Wish is My Command*. Ed. by H. Lieberman. Morgan Kauffman, 2000, pp. 245–270.

[19] T. Bleser and J. D. Foley. "Towards specifying and evaluating the human factors of user-computer interfaces". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1982, pp. 309–314.

[20] P. Brandl, C. Forlines, D. Wigdor, M. Haller, and C. Shen. "Combining and measuring the benefits of bimanual pen and direct-touch interaction on horizontal interfaces". In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. ACM, 2008, pp. 154–161.

[21] W. Buxton. "Chunking and phrasing and the design of human-computer dialogues". In: *Proceedings of IFIP World Computer Congress* (1986), pp. 475–480.

[22] W. Buxton and B. Myers. "A study in two-handed input". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1986, pp. 321–326.

[23] X. Cao, A. Wilson, R. Balakrishnan, K. Hinckley, and S. Hudson. "ShapeTouch: leveraging contact shape on interactive surfaces". In: *Proceedings of the 3rd IEEE International Workshop on Horizontal Interactive Human-Computer Systems*. IEEE, 2008, pp. 129–136.

[24] M. Cardinaels, K. Frederix, J. Nulens, D. Van Rijsselbergen, M. Verwaest, and P. Bekaert. "A multi-touch 3D set modeler for drama production". In: *Proceedings of International Broadcasting Convention*. 2008, pp. 330–335.

[25] D. Casalta, Y. Guiard, and M. Beaudouin-Lafon. "Evaluating two-handed input techniques: rectangle editing and navigation". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (Extended Abstracts)*. ACM, 1999, pp. 236–237.

[26] A. Cohé, F. Dècle, and M. Hachet. "tBox: a 3D transformation widget designed for touch-screens". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 3005–3008.

[27] C. T. Dang and E. André. "Usage and recognition of finger orientation for multi-touch tabletop interaction". In: *Proceedings of the 13th IFIP TC13 International Conference on Human-Computer Interaction*. Springer, 2011, pp. 409–426.

[28] A. De Nardi. "Grafiti: Gesture Recognition mAnagement Framework for Interactive Tabletop Interfaces". MA thesis. University of Pisa, Italy, 2008.

[29] J. Diedrichsen, E. Hazeltine, S. Kennerley, and R. B. Ivry. "Moving to directly cued locations abolishes spatial interference during bimanual actions". In: *Psychological Science* 12(6) (2001), pp. 493–498.

[30] P. Dietz and D. Leigh. "DiamondTouch: a multi-user touch technology". In: *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2001, pp. 219–226.

[31] R. F. Dillon, J. D. Edey, and J. W. Tombaugh. "Measuring the true cost of command selection: techniques and results". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1990, pp. 19–25.

[32] Doug Engelbart Institute. *Father of the mouse*. `http://www.dougengelbart.org/firsts/mouse.html`.

[33] F. Echtler, M. Huber, and G. Klinker. *Hand tracking for enhanced gesture recognition on interactive multi-touch surfaces*. Technical Report , Technische Universitat Munchen - Institut fur Informatik. 2007.

[34] F. Echtler and G. Klinker. "A multitouch software architecture". In: *Proceedings of NordiCHI 2008*. ACM, 2008, pp. 463–466.

[35] A. Esenther and K. Wittenburg. "Multi-user multi-touch games on DiamondTouch with the DTFlash toolkit". In: *Intelligent Technologies for Interactive Entertainment*. Vol. 3814. 2005, pp. 315–319.

[36] Fingerworks. *iGesture*. `http://www.fingerworks.com`.

[37] C. Forlines, D. Wigdor, C. Shen, and R. Balakrishnan. "Direct-touch vs. mouse input for tabletop displays". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2007, pp. 647–656.

[38] E. Franz, H. Zelaznik, S. Swinnen, and C. Walter. "Spatial conceptual influences on the coordination of bimanual actions: when a dual task becomes a single task". In: *Journal of Motor Behavior* 33(1) (2001), pp. 103–112.

[39] Fraunhofer-Institute for Industrial Engineering. *MT4j - Multitouch for Java*. `http://www.mt4j.org`.

[40] B. Froehlich, J. Hochstrate, V. Skuk, and A. Huckauf. "The GlobeFish and the GlobeMouse: two new six degree of freedom input devices for graphics applications". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2006, pp. 191–199.

[41] S. Gabrielli, S. Bellutti, A. Jameson, C. Leonardi, and M. Zancanaro. "A single-user tabletop card game system for older persons: general lessons learned from an in-situ study". In: *Proceedings of the 3rd IEEE International Workshop on Horizontal Interactive Human-Computer Systems*. IEEE, 2008, pp. 85–88.

[42] K. Gajos and D. S. Weld. "SUPPLE: automatically generating user interfaces". In: *Proceedings of the 9th International Conference on Intelligent User Interfaces*. ACM, 2004, pp. 93–100.

[43] D. Gibbon, U. Gut, B. Hell, K. Looks, A. Thies, and T. Trippel. "A computational model of arm gestures in conversation". In: *Proceedings of Eurospeech*. 2003, pp. 813–816.

[44] Google. *Android*. `http://www.android.com`.

[45] Y. Guiard. "Asymmetric division of labor in human skilled bimanual action: the kinematic chain as a model". In: *Journal of Motor Behavior* 19(4) (1987), pp. 486–517.

[46] Y. Guiard and T. Ferrand. "Asymmetry in bimanual skills". In: *Manual Asymmetries in Motor Performance*, CRC Press., 1995.

[47] J. Han. "Low-cost multi–touch sensing through frustrated total internal reflection". In: *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2005, pp. 115–118.

[48] M. Hancock, S. Carpendale, and A. Cockburn. "Shallow depth 3D interaction: design and evaluation of one–, two–, and three–touch techniques". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2007, pp. 1147–1156.

[49] T. E. Hansen, J. P. Hourcade, M. Virbel, S. Patali, and T. Serra. "PyMT: a post-WIMP multi-touch user interface toolkit". In: *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*. ACM, 2009, pp. 17–24.

[50] T. R. Henry, S. E. Hudson, and G. L. Newell. "Integrating gesture and snapping into a user interface toolkit". In: *Proceedings of the 3rd Annual ACM Symposium on User Interface Software and Technology*. ACM, 1990, pp. 112–122.

[51] K. Hinckley, M. Czerwinski, and M. Sinclair. "Interaction and modeling techniques for desktop two–handed input". In: *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*. ACM, 1998, pp. 49–58.

[52] K. Hinckley, R. Pausch, D. Proffitt, and N. Kassell. "Two–handed virtual manipulation". In: *ACM Transactions on Computer-Human Interaction* 5(3) (1998), pp. 260–302.

[53] K. Hinckley, R. Pausch, D. Proffitt, J Patten, and N. Kassell. "Cooperative bimanual action". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1997, pp. 27–34.

[54] S. E. Hudson, J. Mankoff, and I. Smith. "Extensible input handling in the subArctic toolkit". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2005, pp. 381–390.

[55] P. Isokoski and M. Käki. "Comparison of two touchpad-based methods for numeric entry". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2002, pp. 25–32.

[56] R. J. K. Jacob. "A specification language for direct-manipulation user interfaces". In: *ACM Transactions on Graphics* 5.4 (Oct. 1986), pp. 283–317.

[57] R. J. K. Jacob. "Executable specifications for a human-computer interface". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1983, pp. 28–34.

[58] R. J. K. Jacob, L. Deligiannidis, and S. Morrison. "A software model and specification language for non-WIMP user interfaces". In: *ACM Transactions on Computer-Human Interaction* 6.1 (1999), pp. 1–46.

[59] P. Kabbash, W. Buxton, and A. Sellen. "Two–handed input in a compound task". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1994, pp. 417–423.

[60] D. Kammer, G. Freitag, M. Keck, and M. Wacker. "Taxonomy and overview of multi-touch frameworks: architecture, scope and features". In: *Workshop on Engineering Patterns for Multitouch Interfaces* (2010).

[61] D. Kammer, J. Wojdziak, M. Keck, R. Groh, and S. Taranko. "Towards a formalization of multi-touch gestures". In: *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*. ACM, 2010, pp. 49–58.

[62] J. Karat, J. McDonald, and M. Anderson. "A comparison of selection techniques: touch panel, mouse keyboard". In: *International Journal of Man-Machine Studies* 25(1) (1986), pp. 73–92.

[63] A. K. Karlson, B. B. Bederson, and J. SanGiovanni. "AppLens and LaunchTile: two designs for one-handed thumb use on small devices". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2005, pp. 201–210.

[64] D. Käser, M. Agrawala, and M. Pauly. "FingerGlass: efficient multiscale interaction on multitouch screens". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 1601–1610.

[65] J. A. S. Kelso. "Phase transitions and critical behavior in human bimanual coordination". In: *American Journal of Physiology* 15 (1984), R1000–R1004.

[66] J. A. S. Kelso, D. L. Southard, and D. Goodman. "On the coordination of two-handed movements". In: *Journal of Experimental Psychology* 5(2) (1979), pp. 229–238.

[67] J. A. S. Kelso, D. L. Southard, and D. Goodman. "On the nature of human interlimb coordination". In: *Journal of Experimental Psychology* 203(4384) (1979), pp. 1029–1031.

[68] S. H. Khandkar and F. Maurer. "A domain specific language to define gestures for multi-touch applications". In: *10th Workshop on Domain-Specific Modeling* (2010).

[69] K. Kin, M. Agrawala, and T. DeRose. "Determining the benefits of direct-touch, bimanual, and multifinger input on a multitouch workstation". In: *Proceedings of Graphics Interface 2009*. CHCCS, 2009, pp. 119–124.

[70] K. Kin, B. Hartmann, and M. Agrawala. "Two-handed marking menus for multitouch devices". In: *ACM Transactions on Computer-Human Interaction* 18.3 (July 2011), 16:1–16:23.

[71] K. Kin, B. Hartmann, T. DeRose, and M. Agrawala. "Proton++: a customizable declarative multitouch framework". In: *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2012, pp. 477–486.

[72] K. Kin, B. Hartmann, T. DeRose, and M. Agrawala. "Proton: multitouch gestures as regular expressions". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2012, pp. 2885–2894.

[73] K. Kin, T. Miller, B. Bollensdorff, T. DeRose, B. Hartmann, and M. Agrawala. "Eden: a professional multitouch tool for constructing virtual organic environments". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 1343–1352.

[74] M. Kobayashi and T. Igarashi. "Boomerang: suspendable drag-and-drop interactions based on a throw-and-catch metaphor". In: *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2007, pp. 187–190.

[75] T. Költringer, P. Isokoski, and T. Grechenig. "TwoStick: writing with a game controller". In: *Proceedings of Graphics Interface 2007*. CHCCS, 2007, pp. 103–108.

[76] G. Kurtenbach and W. Buxton. "The limits of expert performance using hierarchic marking menus". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1993, pp. 258–264.

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

[78] G. Kurtenbach. "The design and evaluation of marking menus". PhD thesis. Toronto, Ontario Canada: University of Toronto, 1993.

[79] S. Lao, X. Heng, G. Zhang, Y. Ling, and P. Wang. "A gestural interaction design model for multi-touch displays". In: *Proceedings of British Computer Society Conference on Human-Computer Interaction*. 2009, pp. 440–446.

[80] C. Latulipe, C. Kaplan, and C. Clarke. "Bimanual and unimanual image alignment: an evaluation of mouse-based techniques". In: *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2005, pp. 123–131.

[81] C. Latulipe, S. Mann, C. Kaplan, and C. Clarke. "SymSpline: symmetric two-handed spline manipulation". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2006, pp. 349–358.

[82] G. J. Lepinski, T. Grossman, and G. Fitzmaurice. "The design and evaluation of multitouch marking menus". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 2233–2242.

[83] H. Lü and Y. Li. "Gesture Coder: a tool for programming multi-touch gestures by demonstration". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2012, pp. 2875–2884.

[84] A. Martinet, G. Casiez, and L. Grisoni. "The design and evaluation of 3D positioning techniques for multi-touch displays". In: *IEEE Symposium on 3D User Interfaces* (Mar. 2010), pp. 115–118.

[85] M. Masliah and P. Milgram. "Measuring the allocation of control in a 6 degree-of-freedom docking experiment". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2000, pp. 25–32.

[86] M. J. McGuffin, N. Burtnyk, and G. Kurtenbach. "FaST Sliders: integrating marking menus and the adjustment of continuous values". In: *Proceedings of Graphics Interface 2002*. CHCCS, 2002, pp. 35–41.

[87] F. Mechsner, D. Kerzel, G. Knoblich, and W. Prinz. "Perceptual basis of bimanual coordination". In: *Nature* 414 (2001), pp. 69–73.

[88] N. Mehta. "A Flexible Machine Interface". MA thesis. Toronto, Ontario Canada: University of Toronto, 1982.

[89] MERL. *DiamondTouch*. `http://www.merl.com/projects/DiamondTouch/`.

[90] Microsoft. *Pixelsense*. `http://www.microsoft.com/en-us/pixelsense/default.aspx`.

[91] Microsoft. *Surface 1.0*. `http://technet.microsoft.com/en-us/library/ee692114(v=surface.10)`.

[92] Microsoft. *Windows 7*. `http://www.microsoft.com/en-US/windows7/products/home`.

[93] Microsoft. *Windows phone*. `http://www.microsoft.com/windowsphone`.

[94] J. Moeller and A. Kerne. "ZeroTouch: an optical multi-touch and free-air interaction architecture". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2012, pp. 2165–2174.

[95] M. Morris, J. Wobbrock, and A. Wilson. "Understanding users' preferences for surface gestures". In: *Proceedings of Graphics Interface 2010*. CHCCS, 2010, pp. 261–268.

[96] M. Morris. "Supporting effective interaction with tabletop groupware". PhD thesis. Stanford University, 2006.

[97] T. Moscovich. "Contact area interaction with sliding widgets". In: *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*. ACM, 2009, pp. 13–22.

[98] M. Moyle and A. Cockburn. "Analysing mouse and pen flick gestures". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2002, pp. 19–24.

[99] B. A. Myers. "A new model for handling input". In: *ACM Transactions on Information Systems* 8.3 (1990), pp. 289–320.

[100] W. M. Newman. "A system for interactive graphical programming". In: *Proceedings of AFIPS 1968 (Spring)* (1968), pp. 47–54.

[101] NUI Group. *Touchlib*. `http://nuigroup.com/touchlib`.

[102] D. L. Odell, R. C. Davis, A. Smith, and P. K. Wright. "Toolglasses, marking menus, and hotkeys: a comparison of one and two-handed command selection techniques". In: *Proceedings of Graphics Interface 2004*. CHCCS, 2004, pp. 17–24.

[103] D. Olsen. "Building interactive systems: principles for human-computer interaction". In: Boston, MA, United States: Course Technology Press, 2009, pp. 43–66.

[104] D. R. Olsen Jr. and E. P. Dempsey. "Syngraph: a graphical user interface generator". In: *Proceedings of ACM SIGGRAPH*. ACM, 1983, pp. 43–50.

[105] D. Ostroff and B. Schneiderman. "Selection devices for users of an electronic encyclopedia: an empirical comparison of four possibilities". In: *Information Processing and Management* 24(6) (1988), pp. 665–680.

[106] R. Owen, G. Kurtenbach, G. Fitzmaurice, T. Baudel, and W. Buxton. "When it gets more difficult, use both hands: exploring bimanual curve manipulation". In: *Proceedings of Graphics Interface 2005*. CHCCS, 2005, pp. 17–24.

[107] P. Peltonen, E. Kurvinen, A. Salovaara, G. Jacucci, T. Ilmonen, J. Evans, A. Oulasvirta, and P. Saarikko. "It's mine, don't touch!: interactions at a large multi-touch display in a city centre". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2008, pp. 1285–1294.

[108] *Perceptive Pixel*. `http://www.perceptivepixel.com/`.

[109] K. Perlin. "Quikwriting: continuous stylus-based text entry". In: *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*. ACM, 1998, pp. 215–217.

[110] R. L. Potter, L. J. Weldon, and B. Shneiderman. "Improving the accuracy of touch screens: an experimental evaluation of three strategies". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1988, pp. 27–32.

[111] G. Ramos, M. Boulos, and R. Balakrishnan. "Pressure widgets". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2004, pp. 487–494.

[112] J. Raskin. *The Humane Interface*. Addison Wesley, 2000.

[113] J. Reisman, P. Davidson, and J. Han. "A screen-space formulation for 2D and 3D direct manipulation". In: *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*. ACM, 2009, pp. 69–78.

[114] J. Rekimoto. "SmartSkin: an infrastructure for freehand manipulation on interactive surfaces". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2002, pp. 113–120.

[115] I. Rosenberg and K. Perlin. "The UnMousePad: an interpolating multi-touch force-sensing input pad". In: *Proceedings of ACM SIGGRAPH*. ACM, 2009, 65:1–65:9.

[116] D. Rubine. "Specifying gestures by example". In: *Proceedings of ACM SIGGRAPH*. ACM, 1991, pp. 329–337.

[117] C. Scholliers, L. Hoste, B. Signer, and W. De Meuter. "Midas: a declarative multi-touch interaction framework". In: *Proceedings of the 5th International Conference on Tangible, Embedded, and Embodied Interaction*. ACM, 2011, pp. 49–56.

[118] J. Schwarz, S. E. Hudson, J. Mankoff, and A. D. Wilson. "A framework for robust and flexible handling of inputs with uncertainty". In: *Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology*. ACM, 2010, pp. 47–56.

[119] S. D. Scott, K. D. Grant, and R. L. Mandryk. "System guidelines for co-located, collaborative work on a tabletop display". In: *Proceedings of the 8th European Conference on Computer-Supported Cooperative Work*. Kluwer Academic Publishers, 2003, pp. 159–178.

[120] A. Sears and B. Shneiderman. "High precision touchscreens: design strategies and comparisons with a mouse". In: *International Journal of Man-Machine Studies* 34(4) (1991), pp. 593–613.

[121] O. Shaer and R. J. K. Jacob. "A specification paradigm for the design and implementation of tangible user interfaces". In: *ACM Transactions on Computer-Human Interaction* 16.4 (2009).

[122] B. Shneiderman. "Direct manipulation: a step beyond programming languages". In: *Computer*. Vol. 16(8). Aug. 1983, pp. 57–69.

[123] M. Sipser. In: *Introduction to the Theory of Computation*. 1st. International Thomson Publishing, 1996, pp. 46,70–76.

[124] *Sparsh UI*. `http://code.google.com/p/sparsh-ui`.

[125] S. Swigart. *Easily write custom gesture recognizers for your tablet PC applications*. Microsoft Technical Report. Nov. 2005.

[126] K. Thompson. "Regular expression search algorithm". In: *Communications of the ACM* 11.6 (1968), pp. 419–422.

[127] D. Vogel and R. Balakrishnan. "Occlusion-aware interfaces". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 263–272.

[128] D. Vogel and P. Baudisch. "Shift: a technique for operating pen-based interfaces using touch". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2007, pp. 657–666.

[129] F. Wang, X. Cao, X. Ren, and P. 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*. ACM, 2009, pp. 23–32.

[130] R. Wang, S. Paris, and J. Popović. "6D hands: markerless hand-tracking for computer aided design". In: *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2011, pp. 549–558.

[131] C. Ware and D. Jessom. "Using the Bat: a six-dimensional mouse for object placement". In: *IEEE Computer Graphics & Applications* 8(6) (1988), pp. 65–70.

[132] Weegie. `http://weegie.sourceforge.net`.

[133] W. Westerman. "Hand tracking, finger identification, and chordic manipulation on a multi-touch surface". PhD thesis. University of Delaware, 1999.

[134] D. Wigdor, G. Perm, K. Ryall, A. Esenther, and C. Shen. "Living with a tabletop: analysis and observations of long term office use of a multi-touch table". In: *Proceedings of the 2nd IEEE International Workshop on Horizontal Interactive Human-Computer Systems*. IEEE, 2007, pp. 60–67.

[135] D. Wigdor, H. Benko, J. Pella, J. Lombardo, and S. Williams. "Rock & rails: extending multi-touch interactions with shape gestures to enable precise spatial manipulations". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 1581–1590.

[136] A. Wilson and M. Agrawala. "Text entry using a dual joystick game controller". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2006, pp. 475–478.

[137] A. Wilson, S. Izadi, O. Hilliges, A. Garcia-Mendoza, and D. Kirk. "Bringing physics to the surface". In: *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*. ACM, 2008, pp. 67–76.

[138] J. Wobbrock, M. Morris, and A. Wilson. "User-defined gestures for surface computing". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2009, pp. 1083–1092.

[139] J. Wobbrock, A. Wilson, and Y. 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*. ACM, 2007, pp. 159–168.

[140] J. O. Wobbrock, B. A. Myers, and J. A. Kembel. "Edgewrite: a stylus-based text entry method designed for high accuracy and stability of motion". In: *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2003, pp. 61–70.

[141] C. D. Worth. *xstroke*. `http://pandora.east.isi.edu/xstroke/usenix_2003`.

[142] M. Wu and R. Balakrishnan. "Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays". In: *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2003, pp. 193–202.

[143] M. Wu, C. Shen, K. Ryall, C. Forlines, and R. Balakrishnan. "Gesture registration, relaxation, and reuse for multi-point direct-touch surfaces". In: *Proceedings of the 1st IEEE International Workshop on Horizontal Interactive Human-Computer Systems 2006*. IEEE, 2006, pp. 185–192.

[144] Xbox. `http://www.xbox.com/`.

[145] Xbox. *Kinect*. `http://www.xbox.com/kinect`.

[146] K. Yatani, K. Partridge, M. Bern, and M. W. Newman. "Escape: a target selection technique using visually-cued gestures". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2008, pp. 285–294.

[147] S. Zhao, M. Agrawala, and K. Hinckley. "Zone and polygon menus: using relative position to increase the breadth of multi-stroke marking menus". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2006, pp. 1077–1086.

[148] S. Zhao and R. Balakrishnan. "Simple vs. compound mark hierarchical marking menus". In: *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2004, pp. 33–44.

[149] S. Zhao, P. Dragicevic, M. Chignell, R. Balakrishnan, and P. Baudisch. "Earpod: eyes-free menu selection using touch input and reactive audio feedback". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2007, pp. 1395–1404.

[150] T. Zimmerman, J. Lanier, C. Blanchard, S. Bryson, and Y. Harvill. "A hand gesture interface device". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1987, pp. 189–192.