Chapter 11
Imprecision, Inaccuracy, and Frustration: The Tale of Touch Input

Hrvoje Benko and Daniel Wigdor

Abstract Touch and multi-touch technologies have generated a great deal of excitement. In this chapter we focus on addressing the fundamental limitations associated with the use of touch as the primary input mechanism. We discuss seven problems facing the users of touch-based interfaces and offer a set of possible solutions from the available research so far. In particular, we address issues such as the lack of haptic feedback and hover, as well as problems with precision, occlusion, capture and physical constraints which plague this technology. We then describe two case studies from our own work, which provide complementary solutions to all of the issues discussed in this chapter. By discussing these projects in detail, we aim to expose the reader to the complexity of the issues at hand, to various design considerations, and to the intricate implementation details necessary for implementing such solutions.

Introduction

Supporting direct and natural interactions with the on-screen content is one of the most frequently cited benefits of touch-based interfaces. However, bypassing the input device and allowing input with the human finger to directly the screen comes with significant problems. These range from the issues of reduced precision, finger occlusion, as well as the lack of clear feedback about the state of interaction.

This can manifest itself in a lack of confidence that any given input is being accurately received by the application, caused by a general inability to properly attribute unexpected results to their actual causes. The reduction in confidence in the device often also manifests itself as an increase in user frustration and confusion across the entirety of the experience. This is exacerbated in multi-user systems, in which the actions of other users add additional variability to the system's response.

H. Benko (✉)
Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
e-mail: benko@microsoft.com

C. Müller-Tomfelde (ed.), *Tabletops – Horizontal Interactive Displays*,
Human-Computer Interaction Series, DOI 10.1007/978-1-84996-113-4_11,

In this chapter, we first describe the set of seven problems facing the users of touch-based interfaces. We discuss each problem in detail and offer a set of possible solutions from the available research so far. We then focus on two case studies that address some of these problems in detail. Our first case study explores the set of bimanual (or two-finger) solutions, called *Dual Finger Selections* [1], to the issues of finger occlusion and precision of touch selection. The second case study presents a comprehensive visualization solution, called *Ripples* [2], which provides visual feedback to reduce the ambiguities with touch-based input. Finally, we summarize the techniques discussed in this chapter in hope of providing concise guidance to researchers and practitioners who are interested in developing their own touch-based solutions.

Human Finger as an Input Device

When interacting with a touch system, there are a number of situations where the user's input will result in an unexpected behaviour. The user might place two fingers onto an object anticipating a particular response, and another is presented, or no response at all. Did the hardware fail to detect the touch? Did their fingers miss the target? Is the multi-finger response not what they believed it to be? Was there a simultaneous accidental activation elsewhere on the device that changed the state of the object? Is the object not enabled for touch interaction?

How the application reacts to the user's input determines how well the user will be equipped to understand the reasons for the unexpected behaviour. However, most applications do not provide an explicit feedback mechanism that can help users to understand why their action was not successful, and the application feedback is usually constrained to responses designed to signal the execution of successful actions only. The result is applications which *respond* to touch input, but do not provide information about the *causes* of those responses. We refer to this as *touch feedback ambiguity problem*, where, in the case of confusing or unsuccessful actions, the user is usually left to deduce the cause of error from very little or no application feedback. Previous research, as well as our own observations, have found that this ambiguity can lead to a disconnection from the system, and frustration, or a loss of sense of control [3–5].

To understand this problem, consider the difference between actuation of an on-screen object with a mouse or a touch screen: with the mouse, the user moves the pointer over an object and clicks the button, and with the touch screen, the user taps directly on the object on the screen. Now, consider what happens when the system does not react in an expected manner.

The user is left to interpret this response using the feedback which has been made available by the system. In the case of a mouse input, feedback provided by the operating system helps the user to quickly isolate the cause. Visual movement of the mouse reassures the user that the system is still working, the physical activation of the mouse button affirms that the input was delivered, and the position of the mouse

Table 11.1 Causes of an unexpected behaviour to input in a mouse-based system and the feedback given by the hardware or operating system (OS) in typical mouse and touch systems to each, or left to applications (app)

	Feedback refuting cause	
Cause of unexpected behaviour	Mouse	Touch
System is non-responsive	OS: pointer movement	(App)
Hardware failed to detect input	HW: activation of button	(App)
Input delivered to wrong location	OS: visible pointer location	(App)
Input does not map to expected function	(App)	(App)

pointer makes it apparent where the input was delivered. In touch-based systems, this is typically not the case [6, 7], and so it is left to the application to provide feedback for all of these potential causes. Table 11.1 describes various possible causes of unexpected behaviour, as well as the source and type of feedback available to dispel that cause in each of a mouse and direct-touch system.

It is possible for application designers to provide visual feedback distinguishing these sources of error or frustration. However, this makes it more difficult to produce touch-based applications than mouse-based ones, which do not require this feedback mechanism. Further, relying on individual applications to provide feedback decreases the likelihood of consistency across applications.

Sources of Touch Input Error and Frustration

To fully understand the issue with touch feedback ambiguity, we must first explain the potential sources of error that contribute to the ambiguity. In addition of defining the problems, we provide a detailed summary of all relevant solutions to these problems from the related research work.

Lack of Activation Notification and Haptic Feedback

When interacting with a WIMP system, users feel a physical "click" when they depress the mouse button. When working with a touch screen, users feel the moment of contact with the display. However, depending on the particular hardware, the moment of activation can vary: with some vision-based systems, for example, activation occurs before the finger reaches the display, which might result in a different initial position of the touch contact than where the user thinks the contact occurred. With some resistive technologies, a degree of pressure is required for activation [8]. There is no consistent physical sensation connected with this transition. A correct feedback should indicate the activation moment, and help the user to be accurate in their touches.

Most input devices provide some feedback to the user to signal the activation event (e.g., mouse clicks, keyboard keys presses, etc.). In many legacy touchscreen systems, the screen is used as a controller for a mouse pointer. The consequence is that, for every touch, the pointer moves to the point of contact providing some feedback of its location. However, providing a clear notification to the user that a touch event has occurred is challenging in a multi-touch or multi-user context. Poupyrev et al. investigated providing haptic feedback by embedding a vibro-tactile actuator underneath the touchscreen; however their method precludes scaling to multi-touch [9]. While not designed for finger input, Lee et al. designed a range of haptic sensations, by embedding a solenoid within a stylus for stylus-based interfaces [10]. Hancock et al. investigated the use of auditory feedback, but point out its limitations in a multi-user system; these problems would no-doubt be amplified in a multi-touch system [11].

An interesting alternative is to provide passive physical widgets on top of touch screens such as transparent knobs, buttons, or keypads (e.g., SLAP widgets [12]). While effective in providing direct feedback, such widgets impose a lot of constraints on the size, shape and layout of the interface elements and reducing the directness of the touch experience by requiring the use of additional objects. Physical widgets can also be integrated within the screen, as is the case with air-actuated deformable buttons [13]; however this rather static approach also suffers from the lack of flexibility when it comes to layout or size of the interface.

Lack of Hover State

Buxton stipulated that most modern interfaces depend on a three-state input model [14] (i.e., for a mouse those states are out-of-range, tracking and dragging). However, most touch-screens are only able to reliably sense two states (out-of-range and in-contact). The ability to sense hover is greatly missed as currently any contact with the interactive surface becomes an instant engagement without the ability to preview one's actions before committing. Figures 11.1 and 11.2 illustrate.

Most interactive surfaces have been designed to disregard the activity in the hover zone above the surface in order to reduce the amount of noise in the touch sensing. However, various methods have been proposed to simulate the hover state. Potter et al. proposed sending an activation on contact *take-off* which effectively made the

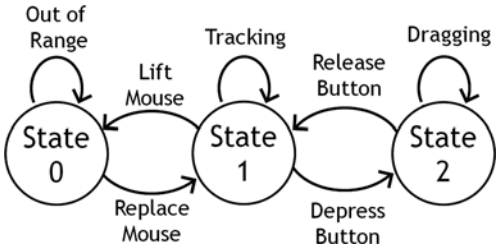

Fig. 11.1 The 3-state model of input as expressed for the mouse (adapted from Buxton [14])

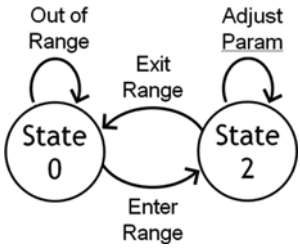

Fig. 11.2 The 2-state model is more common for touch devices (adapted from Buxton [14]). The lack of state 1 means that the user receives no feedback before the system is "engaged"

entire contact state a hover state [8]. This behaviour has also been used in more recent techniques that explored adding more precision to selection (e.g., [1, 15]). Benko et al. used the change on the contact area to simulate the hover state for a touchscreen in a technique called SimPress [1]. In principle their technique is equal to the concept of light and heavy touch as discussed in the Glimpse interaction [16] or to the behaviour of mapping different levels of pressure on a pressure sensitive touch screen device [17].

The Fat Finger Problem: Occlusion and Precision Issues

Two fundamental problems with direct-touch finger input are that the user's finger occludes the target in the critical moment before touching the display (the *occlusion problem*) and that the touch area of the finger is many times larger than a pixel of the display (the *precision problem*).Those two issues are most commonly referred to together as the *fat finger problem* (e.g., [18]).

Touch screens offer good pointing performance for large user interface targets [19], and save device space while maximizing the visual display area by superimposing the control surface over the screen. This has allowed the creation of devices such as the Sony Cyber-shot DSC-N1[1] digital camera and, more recently, the Apple iPhone,[2] in which nearly the entire front surface is occupied by the screen and the size of the device is limited only by its display.

Because of these two issues, a user is often unable to accurately specify the point of contact with the display [8]. For small mobile devices, the occlusion problem is especially drastic because the hand often covers the majority of the display. Previous solutions to these problems have attempted to provide software or hardware aids.

The first such approach is the offset cursor, described by Potter et al. [8] (Fig. 11.3a). This technique involves offsetting the interaction point a few millimetres from the point of contact with the display, and visualized with a crosshair. Users make selections within the user interface by positioning the crosshair over an item, then lifting their finger from the device. The authors note that the result is that users feel put-off that the system does not respond beneath their finger.

[1] http://www.sony.com

[2] http://www.apple.com/iphone

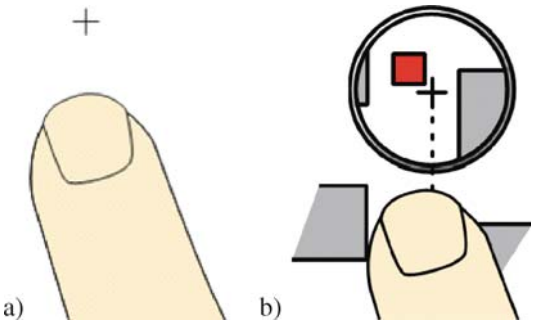

Fig. 11.3 Two offset cursor techniques: (**a**) Potter et al.'s original offset cursor [8]; (**b**) shift offsets both the cursor and the data beneath the finger (adapted from [15])

A variation on the offset cursor solution is Vogel and Baudisch's Shift technique [15]. In Shift, the primary interaction is direct touch. When touching on or near small targets, however, the view of the area beneath the finger, along with the touch location, are automatically offset from the user's finger (Fig. 11.3b).

An alternative to offsetting cursors is the use of more complex tools. Albinsson and Zhai [20] presented a pair of widgets to overcome the fat finger problem and to increase precision. Their widgets (e.g., the *cross-keys* widget and the *precision-handle* widget) are designed for single finger operation, and require first that the user coarsely positions the widget, followed by finer-grained movements by touching a separate buttons or areas of the widget.

Another design is the use of touch cursors. This approach was described in the LucidTouch project [21]. In that project, input to the device is moved to the back of the device to remove the effects of finger occlusion (similar to [22–24]), and the device simulates to the user the experience of looking through the device, while maintaining direct touch interactions (Fig. 11.4). When a finger is hovering over the device, the point used for hit testing is shown on a shadow of the user's hands. When the user actually touches the display, the touch cursor changes colour. This technique addresses the issues of both occlusion and touch precision.

Furthermore, several techniques investigated mapping multiple contact points to reduce the fat finger problem. Each relies on mapping multiple input locations onto a

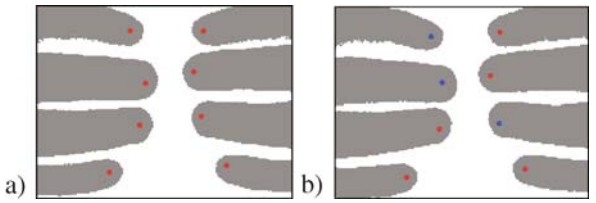

Fig. 11.4 The Lucid touch [21] solution puts the interaction to the back of the device. The images are showing the "virtual shadows" of the user's fingers active in the back of the device: (**a**) all touch-cursors are *red*: no fingers are touching the device, (**b**) the three fingers with *blue* touch-cursors are touching the display.

single selection. Later in this chapter, we describe a set of five two finger techniques, called *Dual Finger Selections* [1], in detail in the first case study below. Similarly, the *DTMouse* solution [25], enabled placing two fingers on the table, positioning the mouse pointer between them, and then make selections by tapping a third finger. It is also possible to treat multiple contacts as approximations of an area cursor such as envisioned by Moscovich et al. [26].

Accidental Activation and Tabletop Debris

As Ryall et al. point out, with a multi-touch system, "every touch counts", especially with horizontal touch systems, accidental activations are common [3]. Such accidental touches can be due to inadvertent touches by the same or different user (e.g., resting an elbow on the tabletop) or by physical objects left on the surface that trigger accidental activations. When this occurs, users are able to observe only the consequence to the application. As they also point out, some accidental inputs are not noticed by the user, and so sudden changes in the state of the system cannot be properly linked to their cause. A meaningful feedback would make the causes of accidental activations clear to the user.

Accidental touches by additional fingers or parts of the user's arm often cause unsuspected behaviours in touchscreen interfaces. Such unwanted actions can be reduced by placing the interaction on the underside of the surface (e.g., [21–24]) or by specifically modelling the user's touch behaviour and discarding the unwanted touches [27]. An interesting alternative is to create user interface elements that respond not only to touch, but to a touch with a particular shape (e.g., ShapeTouch [28]). This requirement makes such elements harder to invoke accidentally since only touching with a particular hand shape will permit activation.

In addition, users of tabletop systems have been observed to place objects on the surface of the screen (e.g., a long term tabletop use study by Wigdor et al. [29]) and many touch-sensing technologies can and do track objects on the surface. The result can be unexpected behaviour when the system responds to these unintended inputs. In our own internal observations of users, we found that this was particularly problematic when an object would act as an additional contact for an object being manipulated by the user.

When scrolling a list, for example, the Microsoft Surface[3] multi-touch tabletop uses the average distance travelled of all contacts on the list to compute its movement. Because it is interpreted as a stationary contact, a beverage placed on the surface of the table, as reported in [29], has the effect of halving the speed of scrolling of a list. A visualization framework should visualize both that debris on the table is being interpreted as an input, as well as when stationary contacts are placing additional constraints on movement.

[3] http://www.microsoft.com/surface

Non-responsive and Captured Content

Invariably, applications will sometimes include elements that are not intended to respond to touch: deactivated controls, background images, etc. Furthermore, elements that are supposed to respond to touch input (such as buttons, etc.) can be unresponsive if they are already *captured* by some other contact.

In a standard WIMP system, user interface controls have two capture states: captured (typically entered when the mouse is clicked on a control), and un-captured. In a multi-touch system, multiple fingers may attempt to capture a control simultaneously. How to deal with multiple, possibly contradictory touches to the same control is an issue decide by framework designers. In the DiamondSpin SDK, "click" events are generated every time a user taps a button, even if another finger is "holding it down" [6]. On Microsoft Surface, "tap" events (equivalent to "click") are generated for buttons only when the last captured contact is lifted from the control. While both approaches have merit, a consequence of the latter is that buttons can be "held down" (captured) by a user.

However, some multi-touch controls can be captured by more than a single contact simultaneously, i.e., *over-captured*. For example, selecting the thumb of a slider with two fingers can mean that it will not track directly under a single finger when moved. When too many contacts have captured a control, its behaviour can be well defined, but inconsistent with the direct-touch paradigm, leading to confusion. Although visual cues should afford inactivation to the user, this state nonetheless adds another source of error in which the user will receive no reaction, requiring correct feedback.

To understand why non-responsive content is particularly problematic for touch systems, consider the difference between actuation of an on-screen object with a mouse or a touch screen. With the mouse, the user moves the pointer over an object and clicks the button, and with the touch screen, the user taps directly on the object on the screen. Now, consider what happens when the system does not react in an expected manner. The user is left to interpret this response using the feedback which has been made available by the system. In the case of a mouse input, feedback provided by the operating system helps the user to quickly isolate the cause. Visual movement of the mouse reassures the user that the system is still working, the physical activation of the mouse button affirms that the input was delivered, and the position of the mouse pointer makes it apparent where the input was delivered. In touch-based systems, this is typically not the case, and so it is left to the application to provide feedback for all of these potential causes.

We provide solutions to the issues of non-responsive content and capture visualization in our second case study, described later in this chapter.

Feedback of Physical Manipulation Constraints

Touch and multi-touch input often rely on the principle of direct manipulation. The principal is that the user places their finger(s) onto an object, moves their finger, and

the object changes its position, orientation, and size to maintain the contact points. The direct-touch paradigm is broken when movement constraints are reached. For example, this can occur when attempting to move an object past the bounds of its container, or to resize an object past its size limit.

A rather clever solution to this problem has been seen in some commercial products. In some cases, this has been addressed by "elastic" effects, where the finger is seen to "slip" along the content, which springs-back to its maximum value when the finger is released. This is demonstrated on the Apple iPhone when attempting to scroll past the end of a list.

As manipulation constraints become more complex, it becomes more difficult to illustrate to the user that a constraint has been reached – because so much is moving at the same time, the user simply doesn't notice the content "slipping".

The *metaDESK* system famously addressed rotation constraints between two physical objects with the placement of a physical constraint on their manipulation: a bar that attached to both and prevented each from rotating relative to the other [30].

The use of a physical constraint relies on the presence of physical objects to constrain the interactions. When these are not available, other solutions are needed to address this problem. Nacenta et al. explored the issue of dynamically constraining the degrees of freedom when manipulating on-screen objects [31], e.g., limiting the scaling of an object when rotating it. Such interactions, while providing clear benefits, have an effect similar to the "snapping" behaviour which tends to break the direct manipulation metaphor and tends to cause finger slipping over content (Fig. 11.5).

A problem with snapping is that it does not allow the user to feel as if they are in control of objects, and prevents them from accessing certain values. The authors addressed this with "catch-up zones", as the user of their system slides towards a desired value, the object accelerates towards the "snap" value, and remains there as the finger slides along the screen. When the finger has travelled a sufficient distance, the object begins to slide once again, at an accelerated rate, until it catches-up with the finger. The result is that objects feel as if they "stick" in desirable locations.

Use of controls can also extend beyond the bounds of those controls. For example, in a traditional GUI, the scrollbar can be captured by selecting it with the mouse. At that point, vertical movements of the mouse are applied to the position of the thumb, and horizontal movements are ignored. The result is that the mouse pointer can be moved away from the slider while still controlling it. This is equally necessary in a touch system, but mapping fingers to their controls is a potential

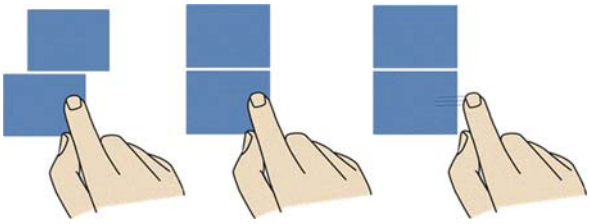

Fig. 11.5 An example physical manipulation constraint: snapping. The lower block stops moving once it aligns with the upper one. The consequence is that the finger slips off the block, giving the user the impression that it was "dropped"

source of confusion with multiple touch-points, controls, and users all interacting simultaneously.

Direct-touch systems rely on the use of a fixed mapping between input location and displayed result. This creates a fun and "natural" feeling, but also significantly limits the potential for "reaching" interaction. *HybridPointing* project did this dynamically, to a great effect, allowing the user to dynamically switch between direct and indirect pointing mode [32].

If not done carefully, mixing direct and indirect touch in the same system can result in user dissatisfaction. In fact, according to Potter et al., the use of indirect in a touch system is undesirable [8]. Additionally, multiple users controlling multiple objects from a distance may become confused as to which objects are under which users' control. Thus, it is important to make it clear which object is responding to which touch at all times.

Case Study #1: Addressing the Fat Finger Problem with Dual Finger Selections

Addressing the fat finger problem implies solving two problems simultaneously: (1) the finger used for selection will partially or completely occlude the target making it difficult to see what is being selected, and (2) it is difficult to perform precise selections since the finger contact is often large when compared with the target. The discrepancy between the finger size and the small target in the standard user interface is best illustrated by the chart in Fig. 11.6. Note that the mean value of 18.3 mm for the right index finger width is derived from the anthropometric data provided by Feeney [33].

	Target Width		
Target Element	**Device Space**	**Visual Space 17'' screen 1024×768**	**Visual Space 39'' screen 1024×768**
Close Button 	18 pixels	6 mm (32.8 % of finger)	10.8 mm (59.0 % of finger)
Resize Handle 	4 pixels	1.34 mm (7.3 % of finger)	2.4 mm (13.1 % of finger)

Fig. 11.6 Comparison of small target widths to an average right hand index finger width at the fingertip (estimated as 18.3 mm for an adult*) (adapted from Benko [34]). Note: Values for the 30″ diagonal screen correspond to the display characteristics used in this case study

We now discuss five different interaction techniques that use two contacts in order to resolve the fat finger problem. These techniques, called *Dual Finger Selections* are part of a larger body of work, previously published by Benko et al. [1].

These techniques assume that the selection does not happen implicitly when landing on or taking off, but rather explicitly. The explicit click event can be facilitated in numerous ways; for example, by using a pressure sensitive device or by using the touch contact area to approximate the hover state (light touch is a hover and heavy touch is click). The adaptation of the later technique called *SimPress* was used in design of Dual Finger Selections [1]. Also, since most touch screens and tabletops cannot identify which of the individual user's fingers or hands is touching the surface without special gloves, for the purpose of these Dual Finger interactions, the first contact with a tabletop surface is always referred to as a *primary finger*, while the second contact is referred to a *secondary finger*.

Dual Finger Offset

The initial and simplest Dual Finger Selection technique, called *Dual Finger Offset*, provides a user triggered cursor offset. The cursor offset is not enabled by default as in [8], but rather invoked on demand. By placing a secondary finger anywhere on the surface, the cursor is subsequently offset with respect to the primary finger by predefined fixed amount. This offset always places the cursor above the primary finger. To accommodate both left- and right-handed users the cursor is placed to the left or to the right of the primary finger based on the relative position of the secondary finger. For example, by placing the secondary finger to the left of the secondary finger to the left of the primary, the cursor appears to the left of and above the primary finger. While providing an on-demand offset helps resolve the occlusion problem, this technique does not assist the user in making more precise selections.

Dual Finger Midpoint

Dual Finger Midpoint technique provides variable offset and enables finer control of the cursor speed (Fig. 11.7). This technique is triggered by placing the secondary finger on the surface. The cursor is then offset to the midpoint between the primary and the secondary finger. DTMouse technique [25] is based on a similar premise; however, it required a third finger to actually trigger a selection. A similar behaviour occurs on any resistive touchpad that places the pointer at the midpoint of all touches (e.g., SMART Board Interactive Whiteboard[4]).

While both fingers are in contact, moving either or both fingers controls the movement of the cursor. This technique allows for variable reductions in cursor

[4]SMART Technologies: www.smarttech.com

Fig. 11.7 Dual finger midpoint technique positions the cursor at exactly the halfway point between the *two fingers*, giving the user both a *cursor* offset as well as a variable reduction of cursor speed.

speed: when both fingers are moving in the same direction and the same speed, the cursor follows with the same speed, while when only one finger is moving, the cursor moves with half the speed of that finger.

At best, this method reduces the finger speed by a factor of 2 which yields good results for most targets; but it does not provide enough control for the smallest targets. An additional shortcoming of this technique is that not all locations on the screen are equally accessible. For example, screen corners are not accessible using midpoint selection. Consequently, the utility of this technique is somewhat limited by the fact that in today's user interfaces small targets often are located in the corners of the screen.

Dual Finger Stretch

Dual Finger Stretch allows the user to adaptively scale a portion of the screen with the secondary finger while the primary finger performs the selection. To allow for simultaneous "stretching" and selection, the primary finger provides the initial anchor location around which the user interface is scaled, while the secondary finger identifies the corner of the square area which will be scaled. By moving the secondary finger closer or further away from the primary finger, the square stretching area is reduced or expanded as illustrated in (Fig. 11.8). Lifting the secondary finger from the table resets the interface to its default un-stretched state. Upon this reset, the cursor is offset with respect to the primary finger and is placed where it was located in the stretched state. The cursor offset is reset when all fingers are removed from the table. The extent of control-display ratio manipulation depends on two physical limits: (1) the closest perceptible distance between user's fingers and (2) the largest diagonal of the screen. For most common mid-screen manipulations, Dual Finger Stretch enables control display ratios roughly up to 10. By allowing clutching and repeated zooming, it may be possible to further increase this ratio.

The Dual Finger Stretch technique has several advantages over the related single finger ZoomPointing technique (designed by Albinsson and Zhai [20]) primarily due to the dual finger design. First, zooming and selection are not decoupled into two separate actions. Instead they can happen concurrently which results in a fluid interaction. Second, the interface scales in all directions from the original primary finger's location. This provides an important advantage over traditional rectangle

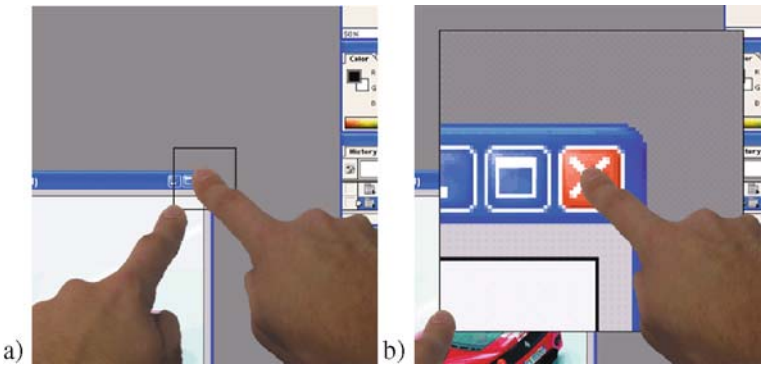

Fig. 11.8 Dual finger stretch technique adaptively scales the user interface: (**a**) the *secondary finger* specifies the *square zooming area* centred at the primary finger's location, (**b**) *primary finger* performs precise selection while, simultaneously, the secondary finger adjusts the level of magnification. © 2009 ACM, Inc. Included here by permission

selection where the two points specify the diagonal corners of the zooming rectangle (also known as bounding box zoom).

With the rectangle selection, the user tends to place the primary finger off target in order to "capture" the target in the zoomed area, while with Dual Finger Stretch, the user places the primary finger directly on target and the interfaces scales underneath in all directions. Placing the finger off-target requires the user's primary finger to traverse an increased distance to perform final selection because the target will appear to move away from the finger as the zoom level increases. By encouraging placement of the primary finger as close to the target as possible, the eventual distance that this finger will need to traverse to acquire the target is minimized.

Dual Finger X-Menu

To allow users to adaptively adjust the control-display ratio as well as obtain cursor offset while looking at an un-zoomed user interface, we have designed the *Dual Finger X-Menu* widget. This circular menu is invoked whenever the secondary finger establishes contact with the surface. It is positioned so that the finger is located at its centre. The user can select a particular assistance mode by moving the secondary finger to any of the desired regions of the menu (Fig. 11.9b). Dual Finger X-Menu has six selection areas shown in Fig. 11.9a. Four areas control the relative speed of the cursor: *normal*, *slow 4x*, *slow 10x*, and *freeze*. Normal mode moves the cursor with the same speed as the primary finger; the two slow modes reduce the speed of the cursor by a factor of 4 and 10 respectively, while freeze mode "freezes" the cursor in place, disabling any cursor movement.

The ability to completely stop the cursor from moving has two benefits. First, by freezing the cursor, the user can quickly and easily establish a desired cursor

Fig. 11.9 Dual finger X-menu contains four selection areas for cursor speed control (normal, slow 4x, slow 10x and freeze), and two toggle areas (snap and magnify). Magnify mode presents an integrated magnification widget in the middle of the menu, while snap mode removes the current cursor offset. When freeze mode is selected, the cursor is completely immobile. © 2009 ACM, Inc. Included here by permission

offset. This is accomplished by freezing the cursor temporarily, moving the finger to achieve the desired offset, and then unfreezing the cursor again. Second, when selecting very small targets, even small amounts of noise can cause an error. Such noise can be due to device tracking errors, tremor in the user's hand, or noise due to the clicking motion. By freezing the cursor in place, the user can ensure that the desired selection is successful even in very noisy conditions.

The left two areas on the crossing menu invoke two helper modes: "snap" and "magnify". When snapping is triggered, the cursor offset (if any) is removed and the cursor snaps back to the current location of the primary finger. This mode is useful in repositioning the cursor in the slow movement modes because it is easy to run out of tracked screen space when using the slow cursor modes. Magnify mode presents a small magnification area in the middle of the crossing menu that shows the enlarged area under the cursor. The magnification factor is fixed at 2x.

This mode is particularly useful when the primary finger overlaps the cursor. In this case the magnified image acts as a lens showing the portion of the interface obstructed by the primary finger. A simple cursor notification widget displays which cursor speed level is currently selected, without requiring the user to refer back to the menu. The behaviour of this notification widget can be seen in Fig. 11.10.

Dual Finger X-Menu is not operated by clicking, but rather by "crossing" the finger into a particular area, which enables more experienced users to activate modes by simply performing quick strokes in a particular direction. With practice, this

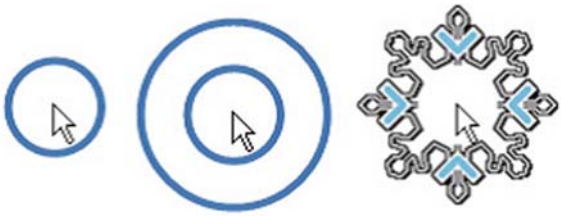

Fig. 11.10 Cursor notification widget signals the current amount of cursor speed reduction: (**a**) 4x reduction, (**b**) 10x reduction, and (**c**) frozen cursor. © 2009 ACM, Inc. Included here by permission

selection can be made without looking, and could therefore allow for an expert mode in which the menu could be completely hidden from the user. Removing the secondary finger from the surface will cause the menu to disappear.

Dual Finger Slider

The different interaction modes of Dual Finger X-Menu offer a palette of options, but tend to distract the user from the task at hand. The *Dual Finger Slider* technique incorporates the menu's most useful features, but simplifies and streamlines the overall interaction (Fig. 11.11). Given that two finger interactions are a very natural way of specifying distance, this interaction uses the distance between fingers to switch between cursor speed reduction modes. This technique does not present an on-screen widget to the user. Instead, it relies completely on the user's ability to gauge the spatial relationship between their fingers. The same cursor notification widget (Fig. 11.10) is used to signal the cursor speed to the user.

Moving the secondary finger towards the primary finger reduces the cursor speed in 3 discrete steps. This allows for the same reductions in cursor speed that is available in Dual Finger X-Menu: *normal*, *slow 4x*, *slow 10x*, and *freeze*. Moving the secondary finger away from the primary increases the speed up to the normal speed. Continuing to move the fingers apart triggers a "snap" which warps the cursor back to the primary finger's location. Snapping is signalled by a distinct sound effect. The distance that the secondary finger traverses in switching speed reduction modes is predefined and is not dependent on the distance between the fingers. The modes are remembered even after the user lifts the secondary finger which allows for clutching in the interaction.

Summary of Experimental Results

In a qualitative comparative study which asked the users to repetitively select small targets (1, 2, 4, 8 pixels wide on a $30''$ 1024×768 pixel screen), the Dual Finger Stretch technique outperformed the others and was also the most preferred technique. Interestingly, Stretch was the only one that did not provide a cursor offset. This clearly demonstrated that the benefit of increased target size successfully compensated for the fingertip occlusion factor. The data from this experiment is consistent with the results from a study by Albinsson and Zhai [20] which also showed that their zooming technique outperformed on-screen widgets that provided cursor speed control.

However, in many applications, scaling may have an undesired effect of losing overview of the interface and there Slider and X-Menu offer appealing alternatives. Both Slider and X-Menu techniques performed comparatively to Stretch in terms of error rate (all three were under 5%), but imposed a 1s penalty in performance.

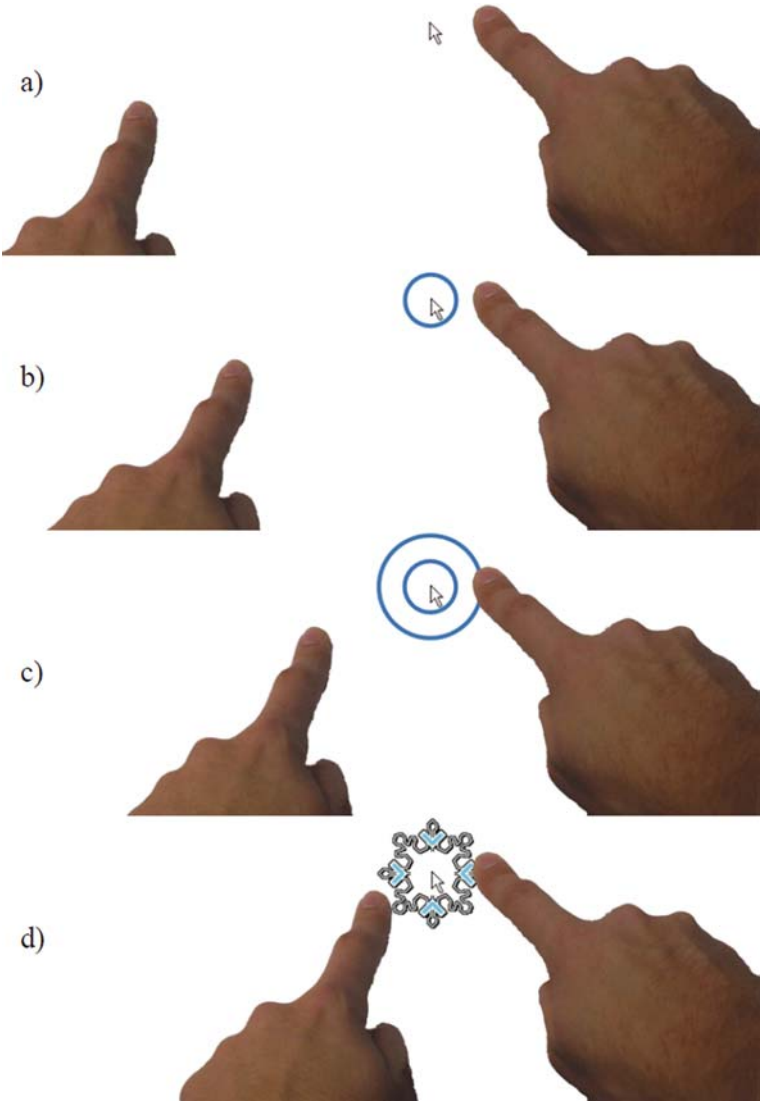

Fig. 11.11 Dual finger slider – the *right finger* (primary) controls the cursor, the *left finger* (secondary) is invoking the invisible slider; speed reductions modes are achieved by moving the fingers closer together: (**a**) normal, (**b**) slow 4x, (**c**) slow 10x, (**d**) frozen cursor mode. © 2009 ACM, Inc. Included here by permission

Selecting really small targets might not be a frequent task in a touch-driven interface.

These five techniques present a useful set of interactions that effectively resolve the fat finger problem. While simple and effective, they require the use of at least two

fingers which can be a significant limitation, particularly on a small display where a single finger use would be preferred. Following this work, Vogel and Baudisch, incorporated the offset zoomed region (similar to the X-Menu solution) into the *Shift* interaction technique designed for a single finger use [15]. Shift showed the zoomed offset region with a delay proportional to the size of the target underneath, which makes it appear only when the target was small enough to warrant the zoom and the improved precision.

Case Study #2: Addressing the Feedback Ambiguity Problem

Currently there is no generalized visual language for conveying various error conditions to the user. It is left to the application designer to reflect the state of the contact in some visual property of the application. Even worse, there has been no description of a set of visual states and transitions which, when taken together, address each of these conditions. There are two possible outcomes: either the application fails to provide a visualization, or each application provides its own, independent (and therefore inconsistent) visualization. In either case, the result is that, when system responses do not map to users' expectations, the ambiguity of this feedback makes it impossible to decipher the causes of unexpected behaviour. Further, the burden on designers of applications for direct and multi-touch systems is much greater than for mouse systems, making it difficult to transition from the mouse to the touch world.

Per-contact visuals were selected over other techniques in an attempt to minimize visual overhead: ensuring consistent, unambiguous feedback was available, but not overwhelming. Of course, it would be possible for a designer to fully instrument an application so that every element conveys an individual response that is appropriate to the particular element, rather than having a generalized visualization tool. Were they to do this, a requisite first step would be to identify those states and transitions requiring visualization, ensuring that all sources of unexpected behaviour are identifiable by the user. The contribution of our work, therefore, is threefold. First, we describe the need for this feedback, and describe the various error conditions requiring disambiguation. Second, we describe a spanning set of states and transitions which, if uniquely visualized, results in a touch system free of ambiguity. Finally, we provide a set of application independent, user-tested, and iteratively designed visualizations which illustrate those states and transitions. These designs are ready to be taken-up by system designers, in order to ease the development of effective direct and multi-touch applications.

Ripples, a contact visualization system, was designed to provide a feedback information channel to users, helping them to be more accurate and efficient in their interactions with the system. Ripples consists of a set of 6 visualizations spanning 14 states and transitions that place the information beneath and around users' fingertips. The design of effective contact visualizations is critical as these are intended to be constantly present, sitting atop all applications. In designing contact visualizations, we faced several design challenges:

1. Visualize action sources, alleviate feedback ambiguity
2. Provide clear visual encodings of multiple parameters
3. Maintain visual integrity of underlying applications
4. Build a framework requiring little work from application developers to leverage

Balancing these requirements required careful blending and balancing of interaction, visual, and system design.

Visual States and Transitions

Ripples' states and transitions of can be divided into two categories: those that apply to any direct-touch system, and those which apply primarily to multi-touch.

Basic Contact Visualization States

Basic states provide helpful feedback for any touch-system, and address many of the problems described previously. We determined a need for visualization of several states, and of the transitions between those states. The aim was to establish a minimum spanning set, providing visualizations only where needed to combat specific problems (Fig. 11.12).

State 0 cannot be visualized in most systems, as it precedes detection. The visualizations of transition A and state 1 address the problem of clearly indicating the *activation event*. They also help to note *accidental activations*, as unintended contacts receive an individual response, allowing the user to correct their posture. To help the user to differentiate between *fat fingers* and *non-responsive content*, and to visualize *selection,* the visual provided for transition A differentiates between contacts which have successfully captured an object, and those which have not (Fig. 11.13).

To address the fat finger problem, we also included an animation for transition D (Fig. 11.14). This animation emphasizes the hit-testing point, similar to [8, 15, 21]. Unlike this past research, the point is not offset from the contact, maintaining direct-touch. To overcome occlusion, transition D employs hysteresis, so that it will

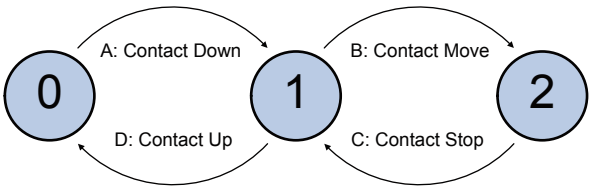

Fig. 11.12 Ripples states and transitions. 0: Not yet touching, 1: stationary contact, 2: moving contact. © 2009 ACM, Inc. Included here by permission

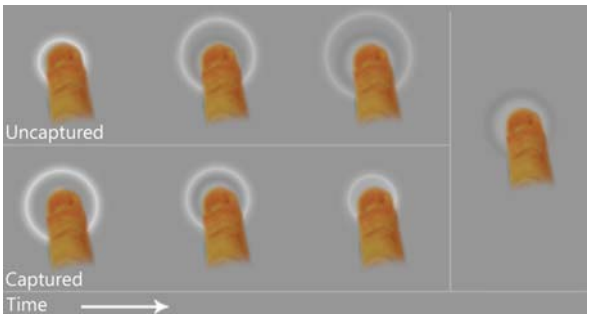

Fig. 11.13 *Left*: 1 of 2 animations is shown for transition A. If an object is captured, a *circle* shrinks around the contact. If not, it "splashes" outward. *Right*: state 1 is identical for both captured and uncaptured. © 2009 ACM, Inc. Included here by permission

Fig. 11.14 Transition D (see Fig. 11.12): when contact is lifted, the visualization shrinks to the hit testing point. © 2009 ACM, Inc. Included here by permission

continue to be visible for a moment after the user lifts their finger. This effect is similar to the afterglow concept presented in the Phosphor work [35] and it serves to explain the user what action just occurred.

Furthermore, as the contact visualization disappears, it contracts to the hit-test point, so that this point is the last thing seen by the user (Fig. 11.14). Unlike previous work, the goal is not to assist the user in making the current selection, but rather to improve accuracy over time by helping them to learn the point/finger mapping.

The addition of State 2 was made to allow us to address the issue of lag. In our visual rendering, the contact is seen to transition to a trail shown behind the finger, making lag appear to be a design element. State 2 and transitions B and C are shown in Fig. 11.15.

Fig. 11.15 State 2 is shown as a trail, which reduces the perception of lag. 1: Contact is static (state 1), 2: begins to move (transition B), 3: moving (state 2). © 2009 ACM, Inc. Included here by permission

Multi-touch and Advanced Contact Visualization States

In addition to the basic contact visualization, additional states were added to address issues which arise primarily with multi-touch systems. These issues are *multiple capture states*, *physical manipulation constraints*, *interaction at a distance*, and *stolen capture*.

In examining these problems, we found that all could be addressed by adding just two states and their associated transitions. These are shown in Fig. 11.16.

State 3 is described earlier as *over-captured*: when the number of contacts captured to a control exceeds the available degrees of freedom of that control, necessitating breaking the direct-touch input paradigm. For example, if two fingers have captured the thumb of a slider, or if three have captured an object enabled for two-finger rotate/translate/scale. As in the basic contact visualizations, this difference is conveyed through the transitions. Transitions F receives the same visual treatment as transition A for an uncaptured contact, and transition G the same as a captured contact. To differentiate these, however, transitions F and G are applied to *all contacts* captured to a control, clearly differentiating states 3 and 1.

State 4 is a condition under which the user has met a constraint on translation, scaling, or rotation of an object. In the Microsoft Surface SDK, these contacts remain captured to the object even though they are no longer touching it. An alternative capture model might cause the contact to lose capture of the object once the finger is no longer touching it. Whatever model is employed, it is critical that a visual be provided to explain why the object is no longer under the user's finger – this addresses the problems of *physical manipulation constraints* and the *interaction at a distance*. To visualize these constraints, we employed a visualization similar to the trails seen in state 2 (see Fig. 11.15). In state 4, the trails become "tethered" to the point at which the constraint was reached, illustrating that the contacts are now "slipping" from their last point of direct-touch (Fig. 11.17).

A purist's interpretation of state 4 would yield tethers when interacting with the majority of controls, since most map multiple degrees of freedom to a single dimension or cannot be moved. What we found, however, was that this could produce what we termed the *Freddy Kruger effect*, where tethers were appearing regularly all over the display. We reduced the frequency of the tethers to the minimal set needed to address specific as sources of error (see above).

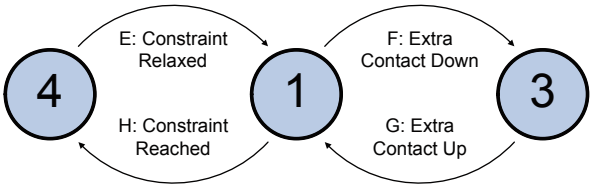

Fig. 11.16 Additional ripples states and transitions for multi-touch. 1: (see Fig. 11.12), 3: object is over-captured, 4: contact operating beyond constraints. © 2009 ACM, Inc. Included here by permission

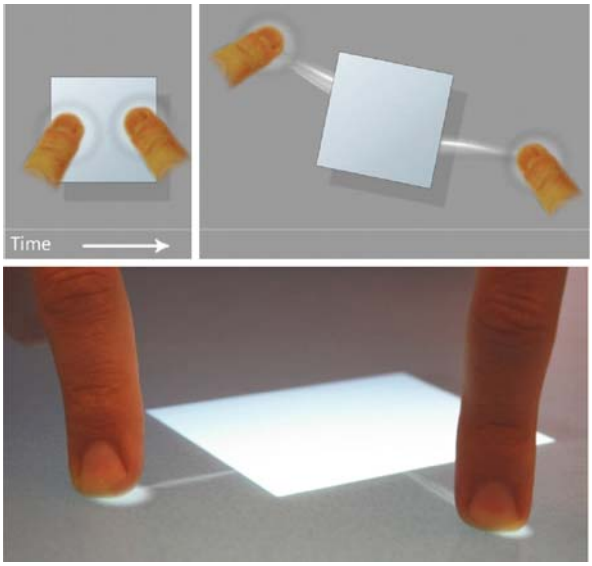

Fig. 11.17 Ripple tethers indicate that a size constraint has been reached on an item being scaled. © 2009 ACM, Inc. Included here by permission

The first such situation was the over-constrained scrolling of a list. It was determined through iterative design that, in most cases, the reaction of the list itself matched user intent, and thus did not require visualization of constraints. The remaining case involves tabletop debris, which can cause slower than expected scrolling of a list. In this situation, determined by the presence of a stationary contact, tethers are rendered to demonstrate that the list is scrolling slowly because of that contact (Fig. 11.18).

The final state 4 visualization visually tethers contacts which have slid off of, but are still captured to, controls. Again, to reduce unnecessary visuals, we split

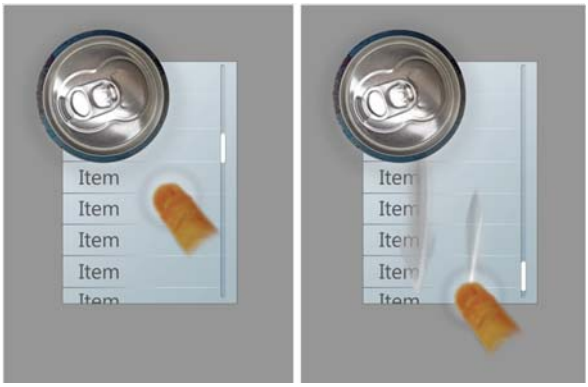

Fig. 11.18 Tethers indicate that slow scrolling of the list is due to the presence of the stationary contact. © 2009 ACM, Inc. Included here by permission

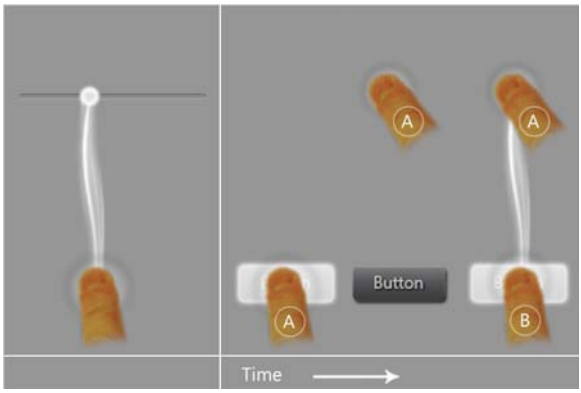

Fig. 11.19 *Left*: contact controlling the slider is visually tethered to it at all times. *Right*: for stationary controls, such as buttons, the tether is shown only when another contact attempts to actuate the control. © 2009 ACM, Inc. Included here by permission

these into two classes: for controls which can be manipulated from a distance, the visualization is shown from the moment the contact slides off the control. For stationary controls, the tether is shown only when another contact attempts to actuate the control, addressing stolen capture (Fig. 11.19).

System Design

Key to Ripples' success as a cross-application standardized visualization is its integration into a platform. Those seeking to implement Ripples for their platforms can do so as a system-level framework which renders the visuals above all applications running on the system. Extensions for various UI toolkits can then be created which pass contextual information to the rendering layer. The contextual information includes which contacts are captured, whether and to where tether lines should be rendered for each captured contact, and which colours should be used to provide the optimal appearance over the application. Once a UI toolkit has integrated with the Ripples rendering framework, applications using this toolkit automatically get the benefits of Ripples without any burden on application authors.

Designers of platforms who wish to support application development without a UI toolkit can provide API's to allow application developers to request the rendering of Ripples above their application in real time.

We recommend that such a framework include mechanisms to allow modification of Ripples, an extensibility mechanism. This allows UI control developers and application designers to fine-tune parts of the visualization provided for contacts captured to their controls and applications. Our system is implemented for modification on a per-control and container basis, similar to pointer customization for mice in most software development kits. Our goal was to make it easy to modify the visual appearance without changing the underlying behaviour. An application can choose to disable visuals entirely, to maintain the functionality but replace the visualization, to select different colours to better match their application, or leave Ripples unmodified.

Design Recommendations

Three sets of design recommendations can be gleaned from this work. The first is for system designers: the states and transitions we have identified all require visualization in order to address the litany of problems affecting user confidence in their inputs to touch and multi-touch systems. In some cases, these visualizations can be reused in order to reduce visual overhead. Because of this reuse, we reintroduce moments of ambiguity which we found to be minimal. Also, it should also be noted that the capture models may vary in other toolkits, requiring a different set of visuals.

The second set of recommendations is for application designers. A contribution of this work is that the Ripples system is application independent, and is intended to be generic and equally well suited for use in any platform. As one of our study participants noted, however, not providing visualizations atop applications can be seen as "more like reality". The challenge to an application designer, therefore, is to provide contextual visualizations which provide unambiguous visual representation of the states and transitions we have described, while maintaining visual consistency with their overall design.

The final set of design concerns relate to those seeking to implement a Ripples-like system. Providing visualizations for the states and transitions described above is sufficient to address feedback ambiguity. However, careful consideration must still be made with respect to visual design issues.

Summary and Conclusions

A preponderance of issues affects the user experience with touch displays. System designers have, for the most part, relied on interaction paradigms designed for the mouse, replacing the pointer with the finger. As we have seen, this simple replacement fails to meet the burden of creating a good user experience, leaving users with diminished precision, feedback, increased accidental input, and a variety of other issues. We have also examined a number of techniques intended to address several of these issues.

A challenge for those seeking to implement a platform to enable touch input is understanding the plethora of interaction techniques described by researchers. While each has merit in addressing some aspect of touch, none successfully addresses all of the issues we have described. As an aid to the reader, we have summarized the challenges we have described, plotted against the various techniques intended to address them (Table 11.2).

When implementing a tabletop system, it is essential that interaction techniques be selected which address all of the issues we have described in this chapter. Upon examination, Table 11.2 makes it apparent that none of the techniques described in the literature fully addresses fundamental interaction problems introduced when using a touch-only system. A combination of these techniques,

Table 11.2 This is a summary of different interaction techniques discussed in this chapter. Issues related to touch input that are solved with each technique are denoted with a checkmark. Two sets of techniques highlighted on the bottom were discussed in detail as case studies in this chapter

		Issues						
		Activation notification and haptic feedback	Lack of hover	Fat finger: occlusion	Fat finger: precision	Accidental activation and debris	Non-responsive and captured content	Physical manipulation constraints
Interaction techniques	Offset cursor [8]		✓	✓	✓			
	Precision handle [20]		✓	✓	✓	✓		
	Cross-keys [20]		✓	✓	✓	✓		
	SimPress [1]		✓					
	Glimpse [16]		✓					
	DTMouse [25]		✓	✓	✓			
	Shift [15]		✓	✓	✓			
	Lucid touch [21]	✓	✓	✓	✓			
	Area cursors [26]		✓	✓	✓			
	SLAP widgets[12]	✓			✓	✓		
	VibroTactile feedback [9]	✓	✓					✓
	HybridPointing [32]		✓	✓	✓			✓
	ShapeTouch [28]					✓		
	Dual finger selections [1]		✓	✓	✓			
	Ripples [2]	✓				✓	✓	✓

therefore, is required in order to create an excellent user experience with touch displays.

Ultimately, the scale and scope of the issues we have described will vary depending on context. While each is important, variations in the conditions of use will affect how severely these issues impact the user experience. Those seeking to build effective tabletop systems should carefully consider the context of use of their system, and prioritize the development of solutions those problems that are likely to have a great impact.

Acknowledgments The authors would like to thank various members of Microsoft Research and Microsoft Surface groups for valuable feedback. In particular, we thank Andrew D. Wilson, Patrick Baudisch, Bryan Beatty, Meredith Ringel Morris, and Jarrod Lombardo for their assistance.

References

1. Benko H, Wilson AD, Baudisch P (2006) Precise selection techniques for multi-touch screens. In: Grinter R, Rodden T, Aoki P, Cutrell E, Jeffries R, Olson G (eds) Proceedings of the SIGCHI conference on human factors in computing systems (Montréal, QC, April 22–27, 2006). CHI '06, ACM Press, New York, pp 1263–1272, doi: 10.1145/1124772.1124963
2. Wigdor D, Williams S, Cronin M, White K, Levy R, Mazeev M, Benko H (2009) Ripples: Utilizing per-contact visualizations to improve user interaction with touch displays. In: Proceedings of the ACM symposium on user interface software and technology (UIST '09), ACM Press, New York, pp 1–10
3. Ryall K, Forlines C, Shen C, Morris MR, Everitt K (2006) Experiences with and observations of direct-touch tabletops. In: Proceedings of the first IEEE international workshop on horizontal interactive human-computer systems (January 05–07, 2006). TABLETOP, IEEE Computer Society, Washington, DC, pp 89–96, doi: 10.1109/TABLETOP.2006.12
4. Shen C, Ryall K, Forlines C, Esenther A, Vernier FD, Everitt K, Wu M, Wigdor D, Morris MR, Hancock M, Tse E (2006) Informing the design of direct-touch tabletops. IEEE Computer Graphics and Applications 26(5) (September 2006):36–46, doi: 10.1109/MCG.2006.109
5. Wu M, Shen C, Ryall K, Forlines C, Balakrishnan R (2006) Gesture registration, relaxation, and reuse for multi-point direct-touch surfaces. In: Proceedings of the first IEEE international workshop on horizontal interactive human-computer systems (January 05–07, 2006). TABLETOP, IEEE Computer Society, Washington, DC, pp 185–192, doi: 10.1109/TABLETOP.2006.19
6. Shen C, Vernier FD, Forlines C, Ringel M (2004) DiamondSpin: An extensible toolkit for around-the-table interaction. In: Proceedings of the SIGCHI conference on human factors in computing systems (Vienna, Austria, April 24–29, 2004). CHI '04, ACM Press, New York, pp 167–174, doi: 10.1145/985692.985714
7. Touchlib, A multi-touch development kit, http://nuigroup.com/touchlib/, Accessed 23.09.2009
8. Potter RL, Weldon LJ, Shneiderman B (1988) Improving the accuracy of touch screens: An experimental evaluation of three strategies. In: O'Hare JJ (ed) Proceedings of the SIGCHI conference on human factors in computing systems (Washington, DC, May 15–19, 1988). CHI '88, ACM Press, New York, pp 27–32, doi: 10.1145/57167.57171
9. Poupyrev I, Maruyama S, Rekimoto J (2002) Ambient touch: Designing tactile interfaces for handheld devices. In: Proceedings of the 15th annual ACM symposium on user interface software and technology (Paris, France, October 27–30, 2002). UIST '02, ACM Press, New York, pp 51–60, doi: 10.1145/571985.571993
10. Lee JC, Dietz PH, Leigh D, Yerazunis WS, Hudson SE (2004) Haptic pen: A tactile feedback stylus for touch screens. In: Proceedings of the 17th annual ACM symposium on user interface software and technology (Santa Fe, NM, October 24–27, 2004). UIST '04, ACM Press, New York, pp 291–294, doi: 10.1145/1029632.1029682
11. Hancock MS, Shen C, Forlines C, Ryall K (2005) Exploring non-speech auditory feedback at an interactive multi-user tabletop. In: Proceedings of graphics interface 2005 (Victoria, BC, May 09–11, 2005). GI, vol 112, Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, ON, pp 41–50
12. Weiss M, Wagner J, Jansen Y, Jennings R, Khoshabeh R, Hollan JD, Borchers J (2009) SLAP widgets: Bridging the gap between virtual and physical controls on tabletops. In: Proceedings of the 27th international conference on human factors in computing systems (Boston, MA, April 04–09, 2009). CHI '09, ACM Press, New York, pp. 481–490, doi: 10.1145/1518701.1518779
13. Harrison C, Hudson SE (2009) Providing dynamically changeable physical buttons on a visual display. In: Proceedings of the 27th international conference on human factors in computing systems (Boston, MA, April 04–09, 2009). CHI '09, ACM Press, New York, pp 299–308, doi: 10.1145/1518701.1518749

14. Buxton W (1990) A three-state model of graphical input. In: Diaper D, Gilmore DJ, Cockton G, Shackel B (eds) Proceedings of the IFIP Tc13 third interational conference on human-computer interaction (August 27–31, 1990). North-Holland Publishing Co., Amsterdam, pp 449–456
15. Vogel D, Baudisch P (2007) Shift: A technique for operating pen-based interfaces using touch. In: Proceedings of the SIGCHI conference on human factors in computing systems (San Jose, CA, April 28–May 03, 2007). CHI '07, ACM Press, New York, pp 657–666, doi: 10.1145/1240624.1240727
16. Forlines C, Shen C, Buxton B (2005) Glimpse: A novel input model for multi-level devices. In: CHI '05 extended abstracts on human factors in computing systems (Portland, OR, April 02–07, 2005). CHI '05, ACM Press, New York, pp 1375–1378, doi: 10.1145/1056808.1056920
17. Han JY (2005) Low-cost multi-touch sensing through frustrated total internal reflection. In: Proceedings of the 18th annual ACM symposium on user interface software and technology (Seattle, WA, October 23–26, 2005). UIST '05. ACM Press, New York, pp. 115–118, doi: 10.1145/1095034.1095054
18. Siek KA, Rogers Y, Connelly KH (2005) Fat finger worries: How older and younger users physically interact with PDAs. In: Proceedings of INTERACT '05, pp 267–280
19. Schilit, B, Adams, N, Want, R (1994) Context-aware computing applications. In: Proceedings of the 1994 first workshop on mobile computing systems and applications – Volume 00 (December 08–09, 1994). WMCSA, IEEE Computer Society, Washington, DC, pp 85–90, doi: 10.1109/WMCSA.1994.16
20. Albinsson P, Zhai S (2003) High precision touch screen interaction. In: Proceedings of the SIGCHI conference on human factors in computing systems (Ft. Lauderdale, FL, April 05–10, 2003). CHI '03, ACM Press, New York, pp 105–112, doi: 10.1145/642611.642631
21. Wigdor D, Forlines C, Baudisch P, Barnwell J, Shen C (2007) Lucid touch: A see-through mobile device. In: Proceedings of the 20th annual ACM symposium on user interface software and technology (Newport, Rhode Island, October 07–10, 2007). UIST '07, ACM Press, New York, pp 269–278, doi: 10.1145/1294211.1294259
22. Hiraoka S, Miyamoto I, Tomimatsu K (2003) Behind touch, a text input method for mobile phones by the back and tactile sense interface. Information Processing Society of Japan, Interaction 2003, pp 131–138
23. Wigdor D, Leigh D, Forlines C, Shipman S, Barnwell J, Balakrishnan R, Shen C (2006) Under the table interaction. In: Proceedings of the 19th annual ACM symposium on user interface software and technology (Montreux, Switzerland, October 15–18, 2006). UIST '06, ACM Press, New York, pp 259–268, doi: 10.1145/1166253.1166294
24. Baudisch P, Chu G (2009) Back-of-device interaction allows creating very small touch devices. In: Proceedings of the 27th international conference on human factors in computing systems (Boston, MA, April 04–09, 2009). CHI '09, ACM Press, New York, pp 1923–1932, doi: 10.1145/1518701.1518995
25. Esenther A, Ryall K (2006) Fluid DTMouse: Better mouse support for touch-based interactions. In: Proceedings of the working conference on advanced visual interfaces (Venezia, Italy, May 23–26, 2006). AVI '06, ACM Press, New York, pp 112–115, doi: 10.1145/1133265.1133289
26. Moscovich T, Hughes JF (2006) Multi-finger cursor techniques. In: Proceedings of graphics interface 2006 (Quebec, June 07–09, 2006). ACM International Conference Proceeding Series, vol 137, Canadian Information Processing Society, Toronto, ON, pp 1–7
27. Vogel D, Cudmore M, Casiez G, Balakrishnan R, Keliher L (2009) Hand occlusion with tablet-sized direct pen input. In: Proceedings of the 27th international conference on human factors in computing systems (Boston, MA, April 04–09, 2009). CHI '09, ACM Press, New York, pp 557–566, doi: 10.1145/1518701.1518787
28. Cao X, Wilson AD, Balakrishnan R, Hinckley K, Hudson SE (2008) ShapeTouch: Leveraging contact shape on interactive surfaces. In: Proceedings of the IEEE international workshop on horizontal interactive human-computer systems (Tabletop '08), pp 129–136

29. Wigdor D, Perm G, Ryall K, Esenther A, Shen C (2007) Living with a tabletop: Analysis and observations of long term office use of a multi-touch table. In: Proceedings of the IEEE international workshop on horizontal interactive human-computer systems (Tabletop '07), pp 60–67
30. Ullmer B, Ishii, H (1997) The metaDESK: Models and prototypes for tangible user interfaces. In: Proceedings of the 10th annual ACM symposium on user interface software and technology (Banff, AB, October 14–17, 1997). UIST '97, ACM Press, New York, pp 223–232, doi: 10.1145/263407.263551
31. Nacenta MA, Baudisch P, Benko H, Wilson A (2009) Separability of spatial manipulations in multi-touch interfaces. In: Proceedings of graphics interface 2009 (Kelowna, BC, May 25–27, 2009). ACM International Conference Proceeding Series, vol 324, Canadian Information Processing Society, Toronto, ON, pp 175–182
32. Forlines C, Vogel D, Balakrishnan R (2006) HybridPointing: Fluid switching between absolute and relative pointing with a direct input device. In: Proceedings of the 19th annual ACM symposium on user interface software and technology (Montreux, Switzerland, October 15–18, 2006). UIST '06, ACM Press, New York, pp 211–220, doi: 10.1145/1166253.1166286
33. Feeney R (2002) Specific anthropometric and strength data for people with dexterity disability, consumer and competition policy directorate, Department of Trade and Industry, London
34. Benko H (2007) User interaction in hybrid multi-display environments. Ph.D. Dissertation. Department of Computer Science, Columbia University, New York, May 2007
35. Baudisch P, Tan D, Collomb M, Robbins D, Hinckley K, Agrawala M, Zhao S, Ramos G (2006) Phosphor: Explaining transitions in the user interface using afterglow effects. In: Proceedings of the 19th annual ACM symposium on user interface software and technology (Montreux, Switzerland, October 15–18, 2006). UIST '06, ACM Press, New York, pp 169–178, doi: 10.1145/1166253.1166280