Ripples: Utilizing Per-Contact Visualizations to Improve User Interaction with Touch Displays

Daniel Wigdor[1], Sarah Williams[2], Michael Cronin[2], Robert Levy[1], Katie White[2], Maxim Mazeev[1], Hrvoje Benko[3]

Microsoft Surface[1] | Microsoft Corp.[2] | Microsoft Research[3]
One Microsoft Way
Redmond, WA, 98052
{ dwigdor | sarahwil | micron | rlevy | katwhite | maximm | benko } @microsoft.com

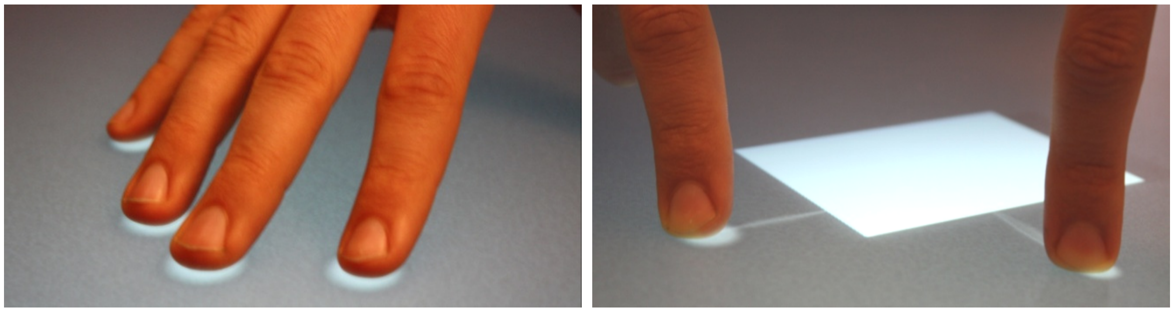
Figure 1. In Ripples each contact with the display is given a response and persistent visualization. Left: photograph of the Ripples system. Right: tethers indicate the fingers have slipped off the item because it reached maximum size.

ABSTRACT

We present *Ripples*, a system which enables visualizations around each contact point on a touch display and, through these visualizations, provides feedback to the user about successes and errors of their touch interactions. Our visualization system is engineered to be overlaid on top of existing applications without requiring the applications to be modified in any way, and functions independently of the application's responses to user input. Ripples reduces the fundamental problem of ambiguity of feedback when an action results in an unexpected behaviour. This ambiguity can be caused by a wide variety of sources. We describe the ambiguity problem, and identify those sources. We then define a set of visual states and transitions needed to resolve this ambiguity, of use to anyone designing touch applications or systems. We then present the Ripples implementation of visualizations for those states, and the results of a user study demonstrating user preference for the system, and demonstrating its utility in reducing errors.

Author Keywords
Touch, multi-touch, fat fingers, precision input, tabletop.

ACM Classification Keywords
H5.2f. Graphical User Interfaces

INTRODUCTION

In recent years, direct-touch displays have become a focus of research and commercial activity. A large amount of research has been dedicated to solving inherent limitations with touch technologies, such as precision and finger occlusion, generally known as the *fat finger problem* (e.g: [3, 6, 15, 23, 24, 25]). Equally important, but largely unaddressed, is the problem of *feedback ambiguity*. This problem is caused by the elimination of both the telepointer and the physical feedback provided by the mouse device itself.

In our own observations of users interacting with touch devices, we have found that this manifests 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. We have observed that this can result in a reduction in confidence in the device, as well 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 variability to system response.

In this paper, we describe *Ripples*, our contact visualization framework (Figure 1), and its ability to improve direct-touch interaction. First, we provide a detailed analysis of the feedback ambiguity problem, which is the core motivation for our work. Then we review related work and describe the design of our system, including interaction design, as well as the equally important visual and system considerations. Next, we report results from a user study which demonstrates the utility of contact visualization in reducing errors in making selections, as well users' preference for the system. Finally, we make a series of design recommendations for those looking to implement their own contact visualization system.

TOUCH FEEDBACK AMBIGUITY PROBLEM

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 [17,20,28].

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 pointer makes it apparent where the input was delivered. In touch-based systems, this is typically not the case [19, 22], and so it is left to the application to provide feedback for all of these potential causes. Table 1 describes various possible causes of unexpected behavior, 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. 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.

Table 1. Causes of an unexpected behaviour to input in a mouse-based system and the feedback given by the hardware or OS in typical mouse and touch systems to each, or left to applications (app).

Cause of Unexpected Behaviour	***Feedback Refuting Cause***	
	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	(app)
Input does not map to expected function	(app)	(app)

Sources of Error

Before discussing how the Ripples framework addresses the feedback ambiguity problem, it is important to discuss the sources of error that typically cause the unexpected behaviours in touch-based systems: it is these events which we aim to communicate to the user, resolving ambiguity.

Activation Event

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 [15]. 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.

Fat Fingers

There are two elements of the fat finger problem: occlusion of the screen by the finger, and the reduction of the contact area to a single point can cause users to 'miss' targets they are physically touching [25]. When the fat finger problem causes a missed target, the correct feedback must distinguish that this failure was due to a miss and, ideally, demonstrate how to avoid missing in the future.

Selection

In systems where land-on selection [15] is employed, visualizing whether a user has successfully touched an on-screen target is essential. This is related to, but distinct from, the fat finger problem.

Non-Responsive Content

Invariably, applications will include elements which are not intended to respond to touch: deactivated controls, background images, etc. 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.

Accidental Activation

As Ryall et al. point out, with a multi-touch system, "every touch counts". Especially with horizontal touch systems, accidental activations are common [17]. 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.

Multiple Capture States

In a WIMP system, UI controls have two capture states: captured (typically entered when the mouse is clicked on a control), and uncaptured. When working with controls on a multi-touch system, more than one contact can capture controls simultaneously. 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. We term this state *over-captured*. To help the user understand over-capture, the contact visualization system must include a visual distinction between not only uncaptured and captured contacts, but over-captured ones as well.

Physical Manipulation Constraints

The direct-touch paradigm is also broken when movement constraints are reached. This can occur, for example, when attempting to move an object past the bounds of its container, or to resize an object past its size limit.

Interaction at a Distance

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

Stolen Capture

In a traditional GUI, controls are captured by selecting them with the mouse pointer. 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' [19]. In the Microsoft Surface SDK, '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' by a user. When twinned with the issue of *interaction at a distance*, it is possible that a button can be 'held down' by a contact not actually touching that button. When a subsequent 'tap' fails, the source of failure should be visualized.

Tabletop Debris

Users of tabletop systems have been observed to place objects on the surface of the screen [26]. The table used in that study did not sense the presence of objects on its surface [5]. This is not true, however, of all sensing technologies used in multi-touch systems. 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 SDK uses the average distance traveled 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 in [26], 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.

A Need for a Framework

Currently there is no generalized visual language for conveying these 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.

RELATED WORK

There are several areas of research which are highly relevant, and we will discuss each in turn.

Several multi-touch displays have been described in the literature. *TouchLight* used two cameras located behind a display to detect the position of users' hands [27], and displayed the raw video image as feedback. This is similar in concept to the *Holowall* [13]. Other technologies include Frustrated Total Internal Reflection [9], capacitance [16, 5], bezel cameras (Smart Board, smarttech.com), and rear-vision (Microsoft Surface - microsoft.com/surface). There are also mobile devices which have enabled direct touch input with a finger or stylus. Early examples of such devices include the Apple Newton and the PARC TAB [18], each of which was designed for input with a stylus, treated in a direct-touch manner. Apple's *iPhone* supports two-point, direct touch input (apple.com/iphone/). It is our intention that our contact visualization system will be of benefit to direct-touch systems of all kinds: single or multi-touch, single or multi-user, small or large display, and utilizing any input technology. Each of these devices is similarly afflicted by feedback ambiguity.

Several researchers have attempted to overcome these problems, though each has had limitations [1, 3, 6, 8, 15, 23, 24, 25]. One possible solution to the problem of imprecision is to move the input surface to the back of the device, such as BehindTouch [11], HybridTouch [21], Under the Table [24], and LucidTouch [25]. Only the LucidTouch provides a solution to the problem of enabling precise land-on selection [15], via 'fingertip cursors'. This comes at the expense of requiring input to the back of the device. Such a solution is not practical in all multi-touch form factors.

Ultimately, the goal of the our visualization system is to increase user confidence in touch-screens by eliminating error ambiguity, reducing errors when touching small targets, and by providing an information channel to convey state information about each contacts. There have been other design and research efforts intended to address each of these. 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 [10]. Other researchers have demonstrated the use of haptic feedback in touch displays, though their methods preclude scaling to multi-touch [14]. Finally, perhaps the most highly related work is in legacy touch-screen systems. In many such systems, the touch 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. Although similar in concept, the size of the pointer is usually such that it cannot be seen until *after* the user has lifted her finger. As such, it cannot convey the full breadth of information included in our contact visualization system.

Also related are any touch applications which include a visual response to every touch, including those that are unsuccessful. Such systems, however, may fail to fully visualize all of the touch states and transitions between those states which we have identified and designed for.

DESIGN

We designed Ripples, a contact visualization system, 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.

Early Consideration: Raw Sensor Output

An early design consideration was to show the raw sensor output. Since our work is based on Microsoft Surface, this is an amalgamated image of the infrared light reflected off of the hands, similar to TouchLight and Holowall [13, 27]. Our preliminary design displayed a number of previous input frames on the screen, so that, when moving, the output would not be occluded (Figure 2). This design was ultimately rejected for two reasons. First, the solution would have been particular to vision-based systems. Second, raw sensor data did not provide sufficient information to overcome all elements of feedback ambiguity.

System Lag

Because of the necessity of processing of sensor input, all interactive systems suffer somewhat from the problem of lag: a requisite hysteresis between the physical input and the visual output in response. This is especially apparent with direct-touch systems, when dragged content will trail behind the finger. What we found in evaluating our early designs of contact visualizations was that lag was especially noticeable, and that it reduced users' perception of responsiveness of the system as a whole. Any implementation of a contact-visualization system must address this problem, in addition to that of feedback ambiguity.

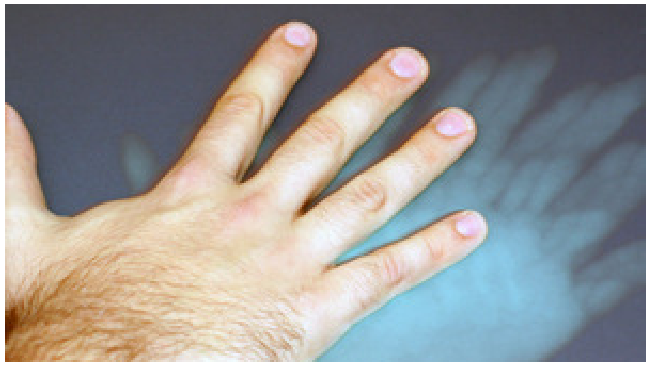
Figure 2. An early attempt to visualize input by displaying raw sensor data. This approach does not generalize across device types.

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 (Figure 3).

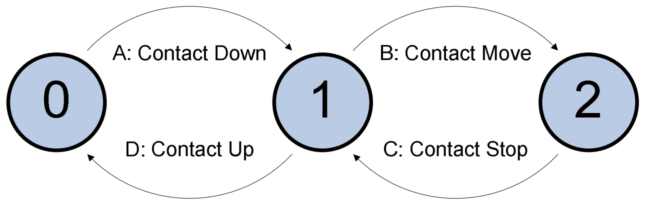

Figure 3. Ripples states & transitions. 0: not yet touching 1: stationary contact 2: moving contact.

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 (Figure 4).

To address *fat fingers*, we also included an animation for transition D (Figure 5). This animation emphasizes the hit-testing point, similar to [15,23,25]. 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 continue to be visible for a moment after the user lifts their finger. Further, as the contact visualization disappears, it contracts to the hit-test point, so that this point is the last thing seen by the user (Figure 5). 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.

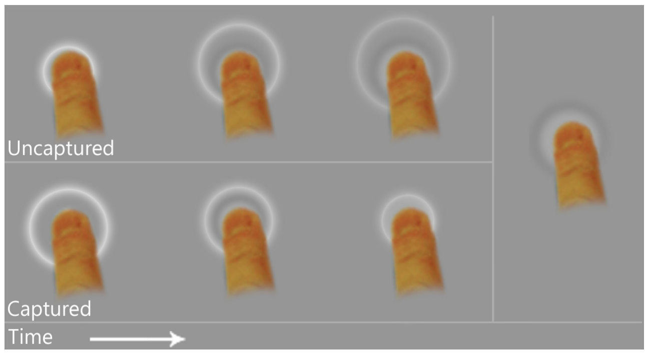

Figure 4. Left: 1 of 2 animations is shown for transition A. If an object is captured, a circle shrinks around the contact. If not, it 'splashes' outward. Right: State 1 is identical for both captured & uncaptured.

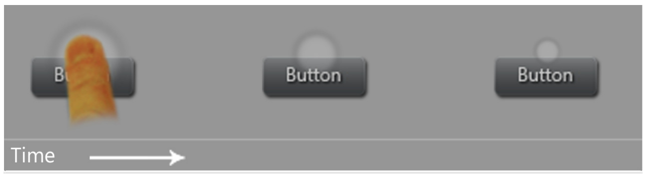

Figure 5. Transition D (see Figure 3): when contact is lifted, the visualization shrinks to the hit testing point.

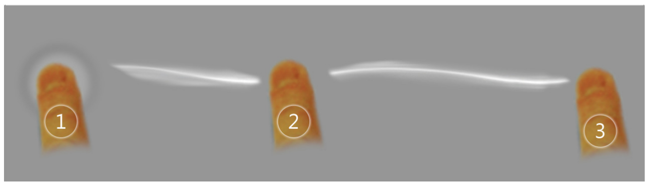

Figure 6. state 2 is shown as a trail, which reduces the perception of lag. 1: contact is static (state 1), 2: begins to move (transition B), 3: moving (state 2).

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 Figure 6.

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 Figure 7.

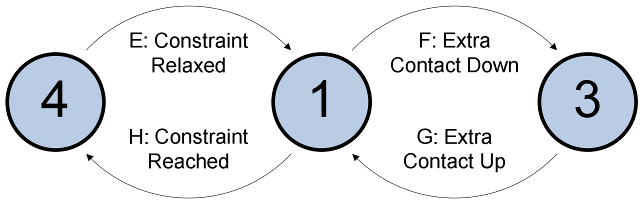

Figure 7. Additional Ripples states & transitions for multi-touch. 1: (see Figure 3). 3: object is over-captured. 4: contact operating beyond constraints.

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.

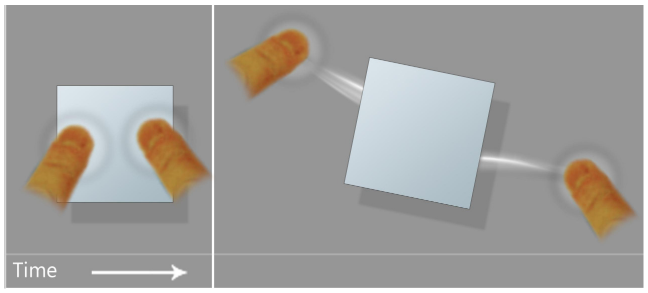

Figure 8. Ripple tethers indicate that a size constraint has been reached on an item being scaled.

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 Figure 6). 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 (Figure 8).

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).

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 (Figure 9).

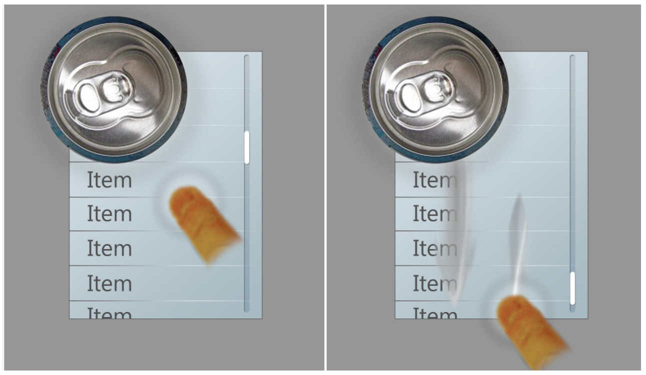

Figure 9. Tethers indicate that slow scrolling of the list is due to the presence of the stationary contact.

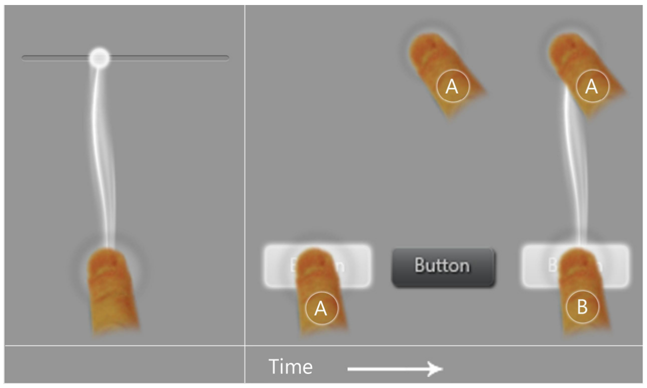

Figure 10. Left: contact controlling the slider is visually tethered to it at all times. Right: for stationary controls, such as buttons, the tether is shown only when another contact attempts to actuate the control.

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 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 (Figure 10).

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.

STUDY: ACCURACY & PERCEIVED RESPONSIVENESS

Having defined the various states and transitions requiring visualization, and having designed visualizations for those states, we set out to test the efficacy of Ripples in aiding the user to interact with a touch display.

We wished to examine two issues with respect to Ripples. First, we wished to test our belief that this feedback would reduce errors when interacting with touch screens. Second, we wished to assess user preference for the presence or absence of the Ripples as an application-independent visualization, and to collect feedback in order to refine them.

Goals and Hypotheses

Part 1: User Preference

Participants were asked to state a preference for Ripples to be enabled or disabled, following a session in which they interacted with two Microsoft Surface applications under both conditions. We wished to ascertain user preference, as well as determine the reason for their preference. We hypothesized that the majority of participants would state a preference for Ripples. We further hypothesized that the majority of participants would react well to the presence of feedback, and that most negative reaction would be to the particular visualization of the contact states implemented.

Part 2: Selection Accuracy

Given the targeting help provided by Ripples (Figure 5), we hypothesized that errors would be reduced with Ripples, and that this effect would be inversely proportional to the size of the target being selected.

Participants

Fourteen paid participants (7 male and 7 female) between the ages of 30 and 62 were recruited from the local community. 13 were right handed, and 1 was left-handed. Education levels varied from some undergraduate to postgraduate degrees. None had used a multi-touch tabletop before, and none had experience with touch devices (excluding automated tellers and self-checkouts). Problems with the apparatus prevented us from analyzing one participant's results, and so it has been excluded.

Apparatus

Two versions of the test suite were loaded onto two different Microsoft Surface units: one version with Ripples enabled, the other with it disabled. Testing was done to ensure that calibration differences would not impact our results.

The device is equipped with a display running at a resolution of 1024x768 (62x46.5cm). The apparatus included three software packages. The first two were the Photos and Music applications from within the Microsoft Surface Application Suite. The Photos application allows users to view and manipulate digital photographs. The Music application allows users to view, manipulate, and listen to digital music and music albums. The third application was a test of selection accuracy, an implementation of the test described in ISO standard 9241-9 [12].

The software ran entirely on the Microsoft Surface unit, with statistics recorded for later analysis. Participant ratings and comments were elicited through questioning and recorded by the lab technician.

Task and Procedure

The experiment was completed in two parts. Participants first completed various simple tasks within the applications. They would then switch to the other table, and complete the same tasks a second time (order was randomly assigned between participants). Next, they were asked to assess any differences between the two tables. Qualitative comments were gathered on the users' opinion of Ripples with respect to interaction with the applications. In the second part of the experiment, each participant completed the selection accuracy measure to assess accuracy. Again, table order was randomly assigned between participants.

Preference data was intentionally collected *before* conducting the accuracy tests, to ensure that the responses would not be coloured by that portion of the experiment.

Part 1: Preference

Participants were shown each of the Photos and Music applications from the Microsoft Surface application suite. Each participant interacted with the application for five minutes. During this time, they were prompted to perform several interactions that would elicit the various Ripples visuals (Ripples case) or demonstrate the situations where they would disambiguate errors (no-Ripples case). This was then repeated with the other Microsoft Surface unit. Participants were then asked which tabletop they preferred and why. Qualitative comments were collected regarding user sentiments towards the visual responses.

Part 2: Selection Accuracy

Participants were shown a selection task application that displays a circular arrangement of 16 equally–spaced, equally-sized, white-coloured targets aligned to the horizontal and vertical centre of the background. The diameter of the arrangement determines the movement amplitude (A) for each trial. This was fixed at 512 pixels (measured from centre of one circle to the centre of an opposing circle), 30.5 cm given the screen resolution of the unit (Figure 5). Three different diameters of target circles were used: 2cm, 2.6cm, and 3.2cm, representing task difficulties of 4.04, 3.69, and 3.41 bits, respectively.

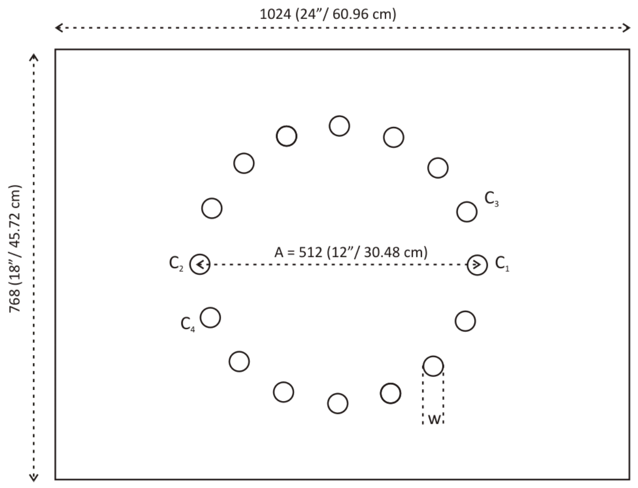

Figure 11. The experimental display in the selection task. Test as described in [12].

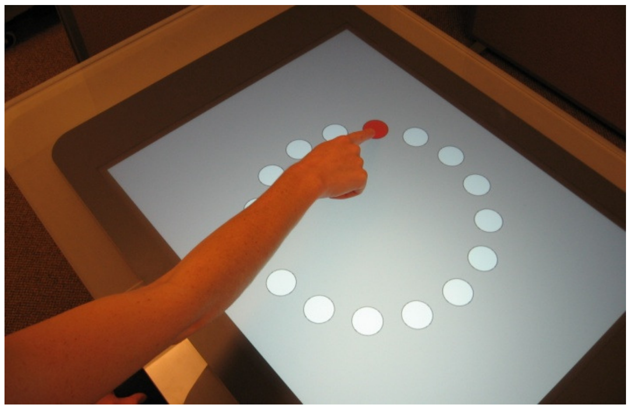

Figure 12. Participant completing selection task using the Microsoft Surface table.

As trials proceed, one circle changes from white to red. The participants' task is to touch the red circle in a clockwise pattern. The first circle to turn red is always the one labelled C_1 in Figure 14, followed by C_2, then C_3, C_4, and so forth.

Trials are defined as a successful activation of a target. If the participant does not touch and lift inside the target, the trial counts as an error and participant is instructed to continue to try to touch the circle until that target is successfully selected (Figure 12). Each participant completed a total of eight blocks on each table. The first set of trials was dropped for all participants, as it served as a practice set to familiarize the participant with the task. In summary, the design was as follows:

16 selections of a target
x 3 target sizes
x 7 blocks
x 13 participants
= 4368 total trials

Results

Part 1: Preference

After completing the application usage tasks on both tables, participants were asked to describe their experiences and differences between the two tables, and to state their preferences for or against Ripples.

Across all participants, eight (62%) preferred Ripples to be enabled, three participants (23%) preferred that it be disabled, and two participants (15%) stated they had no preference. Two who preferred the absence of Ripples made comments to the effect that they felt that the "fingerprints were distracting". One of these participants stated no-Ripples was preferred because it felt "more like reality".

The majority of participants liked the added visual cues provided by Ripples and were able to use these cues to their benefit when interacting with the applications. The participants who preferred Ripples made comments such as "I know where I am touching technically and I'm reassured"; "I can tell where I'm touching … I can see the point of contact [and] know which picture I am touching"; and felt that the "sensors caught my actions better", and "it lets you know it was getting the signal".

In some cases, participants actually believed that the sensing was enhanced in the table with Ripples enabled, and made comments such as "I would prefer… if both tables had equal sensitivity". It was also clear that users appreciated the disambiguation of error cases. Several noted this in comments such as "I like to see what is happening, the visuals help me figure out what's going on."

Part 2: Selection Accuracy

In the selection task, we measured both the number of errors committed by each participant (*errors*) and the completion time for trials without errors (*time*).

Overall, participants made significantly fewer *errors* per trial when using Ripples (Ripples: mean = 19.4%, SD = 17.9%; no-Ripples: mean = 46.9%, SD = 63.8%; t_{272}= -7.34, p < .001). Condition also had a small but significant effect on *time* (Ripples: 0.799 sec; std dev=0.238, no-Ripples: mean=0.833 sec; std dev=0.268; t_{272}=2.349 p=0.020).

Block had a significant effect on the differences in *time* between conditions. There was no significant differences between Ripples and no-Ripples in the 3 blocks of the experiment (no-Ripples: mean=0.838 sec; std dev=0.273; Ripples: mean=0.823 sec; std dev=0.218; t_{116}=0.628, p=0.531). In the last 3 block of the experiment, a significant difference in *time* was observed (no-Ripples: mean=0.832 sec; std dev=0.263; Ripples: mean=0.767 sec; std dev=0.247; t_{116}=3.124, p=0.002). This result suggests an asymmetric learning effect, with greater performance improvements with Ripples than without.

To assess whether the overall accuracy differences between Ripples and no-Ripples varied by size of the object being selected, paired t-tests were also conducted by button diameter. It was hypothesized that the differences would be greater at the smaller sized buttons, since the smaller size is thought to have a higher chance of error. As seen in Table 2, the effect of Ripples on error rate was significant across all target sizes. As hypothesized, this effect was greater for smaller targets than for large ones.

Discussion

The results of the study reveal several things about the efficacy of the current design of Ripples in achieving our goals. There are two general issues examined in this study: qualitative feedback / preference, and assistance with accuracy of selection.

Table 2. Error rate and (SD) per trial by target size and condition across all 7 blocks.

	Target Size		
Condition	20mm	26mm	32mm
Ripples	32.8% (19.6)	15.7% (14.3)	9.7% (9.4)
No-Ripples	79.1% (79.6)	40.2% (60.0)	21.6% (24.4)
Significance	t_{90} = -5.38, p < .001	t_{90} = -3.82, p < .001	t_{90} = -4.82, p < .001

Preference

It is not surprising that the majority of users prefer a system utilizing Ripples. Interestingly, these results were found while interacting with applications which have been specifically designed for touch input – arguably this effect would be even greater for applications not specifically designed for this input mechanism.

Despite this success, 3 participants specifically stated that they would prefer Ripples to be disabled, complaining that they were distracting. While the benefit of the system to the other users was clear, and was generally preferred, it is also clear that there may be a need to enable users to disable the system if they so choose.

Selection Accuracy

It is clear that Ripples is effective in reducing errors in the selection task. This is encouraging: previous more elaborate measures at improving accuracy for touch-selections have had similar results – but only Ripples does not require any rethinking of the direct-touch input paradigm. What is also of interest is that Ripples does not actually provide any functionality to improve accuracy of touches. Rather, it teaches users about the precise location of their hit-testing point, *after* each successful and unsuccessful selection. To our knowledge, this is the first demonstration that users can learn to be more precise in overcoming the fat-finger problem – and Ripples is the first technique which has been demonstrated to provide the information needed to learn.

CONCLUSIONS AND FUTURE WORK

Ripples has proven to be successful in improving user experience with touch displays. Selections are more accurate, and qualitative feedback clearly indicates that they aid with interaction with applications.

We believe our contributions to be fourfold. First, we described the various sources of ambiguous system feedback in touch systems. Second, we defined a set of visual states and transitions which span these sources, and thus describe elements necessary to provide ambiguity-free feedback in a touch system. Third, we defined a particular set of visualizations for those states and transitions. Last, we presented the results of a controlled experiment which demonstrated that providing visualizations is helpful to interaction for both preference and accuracy, and which provided feedback about the particular visualizations we implemented.

In future, we intend to continue to refine the visual design of Ripples, and to investigate the details of its use with non vision-based hardware. While Ripples was designed to be agnostic to the sensing technology, particularities of sensors may lead to the need to refine the system. Also clear is the need to refine and extend Ripples for systems which are capable of detecting not just fingertips, but also the shape of input, such as full hands or other postures and gestures.

Also worth examining is how the visual design of Ripples affected user preference results shown in the study, and how different designs might meet the needs with less of an effect on preference.

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.

Visual Overhead

Adding visualizations to every contact point creates some visual overhead. It was our goal to provide a visualization that acts as an information channel, while not distracting the user from their primary task or occluding content. Several iterations on the design, guided through consultation, prototyping, and reactions from our target users, saw a constant reduction in their size and complexity. Many different visualization and rendering techniques were attempted. Ultimately, it became clear that a minimalist design was required, rendering only enough to visually convey the parameters necessary to eliminate feedback ambiguity.

Touch Visuals Final Rendering

As we described, the design underwent several iterations. The issue of visual overhead in particular required several changes. A particularly surprising outcome of our process was the transition away from simple, minimal circles around the fingers ('halos') to the partially transparent amorphous shape (Figure 13). It was this iteration that largely eliminated complaints of visual overhead and distraction.

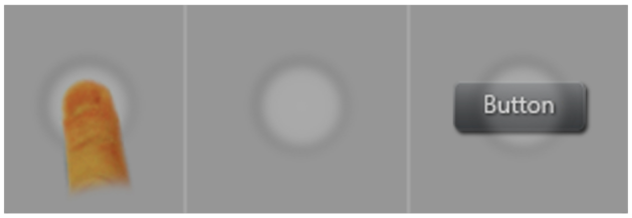

Figure 13. Rendering of Ripples contact visualization. Left: what the users see. Centre, right: what is rendered beneath the finger.

Alternative Design Approaches

Several designs were ultimately rejected as part of our iterative design process. In order to help those looking to build on our work, we review two of the approaches in particular, and describe why they were rejected.

The first is the version shown in Figure 14. This approach provided a large canvas for visualizing parameters, but was ultimately rejected.

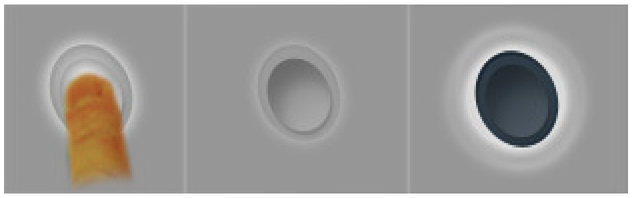

Figure 14. Rejected design for various states. The visual seemed to 'get between' users and content.

A subsequent iteration placed a solid 'halo' around each contact. While minimalist, this design too was rejected through our iterative process. Feedback indicated that it was too 'large', even when the halo was made to be as small as it could be and still be visible beneath the finger.

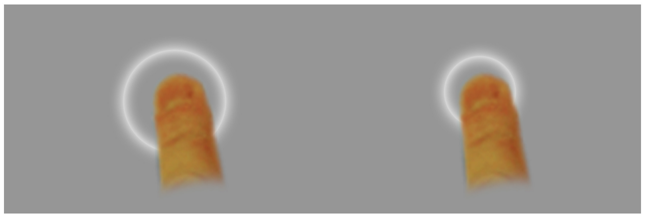

Figure 15. A follow-up design: a 'halo' placed around each contact was described as 'too large' in informal evaluations, no matter its size.

This led us to begin to explore less distinct, filled shapes, which led ultimately to our ultimate design (Figure 13). We found that this approach, when combined with transparency, largely eliminated complaints of visual overhead.

ACKNOWLEDGMENTS

The authors would like to thank various members of Microsoft Research and Microsoft Surface groups, our participants, and paper reviewers for valuable feedback. In particular, we thank Bryan Beatty, Merrie Morris, and Jarrod Lombardo for their assistance.

REFERENCES

1. Albinsson, P. and Zhai, S. (2003). High precision touch screen interaction. *CHI '03*. 105-112.

2. Baudisch, P., et al. (2006). Phosphor: Explaining Transitions in the User Interface Using Afterglow Effects. *UIST '06,* 169-174.

3. Benko, H., et al. (2006). Precise selection techniques for multi-touch screens. *CHI '06*. 1263-1272.

4. Buxton, W. (1990). A Three-State Model of Graphical Input. *Human-Computer Interaction - INTERACT '90*. 449-456.

5. Dietz, P. and Leigh, D. (2001). DiamondTouch: a multi-user touch technology. *UIST 2001*. 219-226.

6. Esenther, A., Ryall, K., (2006). Fluid DTMouse: Better Mouse Support for Touch-Based Interactions. *AVI 2006*. 112- 115.

7. Forlines, C., Shen, C., (2005). DTLens: Multi-User Tabletop Spatial Data Exploration. *UIST '05*. 119-122.

8. Forlines, C. et al. (2006). HybridPointing: Fluid switching between abs and relative pointing with a direct input device. *UIST '06*. 211-220.

9. Han, J. (2005). Low-cost multi-touch sensing through frustrated total internal reflection. *UIST '05*. 115-118.

10. Hancock, M.S., et al. (2005). Exploring non-speech auditory feedback at an interactive multi-user tabletop. *GI '05*. 41-50.

11. Hiraoka, S. et al. (2003) Behind Touch, a Text Input Method for Mobile Phones by The Back and Tactile Sense Interface. *Information Processing Society of Japan, Interaction '03*. 131-138.

12. ISO, 2002. Reference Number: ISO 9241-9:2000(E). Ergonomic requirements for office work with visual display terminals (VDTs)—Part 9—Requirements for non-keyboard input devices (ISO 9241-9) (Vol. February 15, 2002). ISO.

13. Matsushita, N., Rekimoto, J. (1997). HoloWall: designing a finger, hand, body, and object sensing wall. *UIST '97*. 209-210.

14. Poupyrev, I., et al. (2002). Ambient Touch: Designing tactile interfaces for handheld devices. *UIST '02*. 51-60.

15. Potter, R., Weldon, L., and Shneiderman, B. (1988). Improving the accuracy of touch screens: an experimental evaluation of three strategies. *Proceedings of CHI '88*. p. 27-32.

16. Rekimoto, J. (2002). SmartSkin: an infrastructure for freehand manipulation on interactive surfaces. *CHI '02*. 113-120.

17. Ryall, K. et al. (2006). Experiences with and Observations of Direct-Touch Tabletops. T*abletop '06*. 89-96.

18. Schilit, B. et al. (1994). Context-Aware Computing Applications. *Workshop on Mobile Computing Systems and Applications*. 85-90.

19. Shen, C. et al. (2004). DiamondSpin: an extensible toolkit for around-the-table interaction. *CHI '04*. 167-174.

20. Shen, C. et al. (2006) Informing the Design of Direct-Touch Tabletops, *Special Issue of IEEE Computer Graphics and Applications*, September/October, 2006.

21. Siek, K.A. et al. (2005). Fat Finger Worries: How Older and Younger Users Physically Interact with PDAs. *INTERACT '05*. 267-280.

22. *Touchlib, A Multi-Touch Development Kit* (http://nuigroup.com/touchlib/).

23. Vogel, D. and Baudisch, P. (2007). Shift: A Technique for Operating Pen-Based Interfaces Using Touch. To appear in *CHI '07*. 657-666.

24. Wigdor, D. et al. (2006). Under the table interaction. *Proceedings of UIST '06*. 259-268.

25. Wigdor, D. et al. (2007). LucidTouch: A See-Through Mobile Device. *UIST '07*. 269-278.

26. Wigdor, D. et al. (2007). Living with a Tabletop: Analysis and Observations of Long Term Office Use of a Multi-Touch Table. *Tabletop '07*. 60-67.

27. Wilson, A. D. (2004). TouchLight: an imaging touch screen and display for gesture-based interaction. *ICMI '04*. 69-76.

28. Wu, M. et al. Gesture Registration, Relaxation, and Reuse for Multi-Point Direct-Touch Surfaces. *TableTop '06*. 183-190.