

Designing for Low-Latency Direct-Touch Input

Albert Ng¹, Julian Lepinski², Daniel Wigdor², Steven Sanders³, Paul Dietz¹

¹Microsoft Applied
Sciences Group

Redmond, WA, USA

{v-albeng | pdietz}@microsoft.com,

²Dept. of Computer Science
University of Toronto

Toronto, Canada

{lepinski | dwigdor}@dgp.toronto.edu,

³Sanders Capital
New York, NY, USA

steven.sanders@sanderscapital.com

ABSTRACT

Software designed for direct-touch interfaces often utilize a metaphor of direct physical manipulation of pseudo “real-world” objects. However, current touch systems typically take 50-200ms to update the display in response to a physical touch action. Utilizing a high performance touch demonstrator, subjects were able to experience touch latencies ranging from current levels down to about 1ms. Our tests show that users greatly prefer lower latencies, and noticeable improvement continued well below 10ms. This level of performance is difficult to achieve in commercial computing systems using current technologies. As an alternative, we propose a hybrid system that provides low-fidelity visual feedback immediately, followed by high-fidelity visuals at standard levels of latency.

ACM Classification: H.5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Human Factors

Keywords: touch input, multi-touch, user input, latency

INTRODUCTION

Although long popular in automated teller machines and other kiosk applications, the last decade has seen the mass adoption of direct-touch interfaces. Earlier systems typically utilized soft buttons which the user pressed to interact with the system – hence the popularity of research dedicated to improving the precision of selection using a direct-touch device, beginning with [22]. It wasn’t until the emergence of multitouch that swiping, dragging, and pinching became standard interface elements. These operations are typically used in commercial offerings to mimic the manipulation of actual physical objects, leveraging the principles of a “reality-based interface” to render their use more simple for novice users [12].

Unfortunately, the illusion of direct physical manipulation is limited by a number of factors. Among these is the inherent delay between finger movement and the updating of the display. Direct-touch systems are particularly sensitive to this latency, as the unification of input and output makes latency more noticeable than with indirect touch.

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

UIST ’12, October 7–10, 2012, Cambridge, Massachusetts, USA.

Copyright 2012 ACM 978-1-4503-1580-7/12/10...\$15.00.

Using high-speed cameras, we informally tested a wide variety of iOS, Android, Windows, and Windows Phone devices doing basic OS navigation tasks, and found that this latency typically ranges between 50-200ms in current commercial devices. Past work has found that end users often describe devices with lower latency as feeling more “responsive” [2]. Utilizing a 1ms-latency touch input device, we expanded on this work by conducting a study designed to measure human perception of drag latency in touch. We found that participants could reliably perceive touchscreen latency as low as 5-10ms. Thus, an order of magnitude improvement is required over current commercial offerings before users may begin to perceive response to touch as instantaneous.

Latency in a direct-touch device has many sources, but usually has three main components: 1) the physical sensor that captures touch events; 2) the software that processes touch events and generates output for the display; 3) the display itself. Reducing system latency in a direct-touch system requires addressing latency issues in all three of these components, and is thus a significant challenge. We have constructed a direct-touch device that achieves 1ms of latency by combining a low-latency touch sensor and projector with a dedicated processing system. In order to achieve extremely low latency, our system replaces traditional operating system software and computing hardware with a dedicated, custom-programmed.



Figure 1: An “Accelerated Touch” visualization: the user receives both near-instantaneous visual feedback at low-fidelity (in red), as well as high-fidelity feedback with current-generation latency.

As we will discuss, low latency input and output devices are feasible in the relatively short term. However, overcoming latency due to software execution, while maintaining the convenience of a multi-threaded OS, user interface toolkit, and plethora of other utilities developers rely on, is a significantly greater challenge.

In order to address this, we propose a system we have termed Accelerated Touch, shown in Figure 1. In an Accelerated Touch system, we separate some of the basic visual responses to touch from the broader application logic. The result is a system with a nimble touch processor, capable of providing near-instantaneous visual feedback on user interaction, with a higher-fidelity visual layer rendered at more traditional levels of latency. This is shown in Figure 1, with the Accelerated Touch visualizations shown in red.

The contributions of this paper are fivefold. First, we describe the architectural problems in current systems with respect to latency, and describe possible avenues to reduce them. Next, we introduce our prototype device, which is capable of crude visual feedback at extremely low latency. Third, we present the results of a user study, which demonstrate that the lower latencies made possible by our prototype are perceptible to users, and therefore valuable. Fourth, we present a set of designs for a set of UI controls with bifurcated visual responses between low and high latency, in which users may receive immediate feedback appropriate to context, followed by a more complete response guided by application logic. Finally, we describe options in software architecture to achieve the bifurcated feedback we have presented. First, we present a review of relevant related work.

RELATED WORK

Three areas of related work are illuminating. First, we discuss the impact of latency on user interfaces, with a focus on the oft-quoted “100ms” number. Next, we describe methods by which *perceived* latency (as at least somewhat distinct from actual latency) have been addressed. Finally, we describe various technologies used for touch input.

Impact of Latency on UI

The effects of latency on direct-manipulation based interaction have not been extensively studied. On latency and delay in general, early work by Card, Mackinlay and Robertson [4] is frequently cited, in which the authors claim that the perception of animation requires a refresh rate no lower than 10Hz. This claim is based on Card, Moran and Newell's [5] work in estimating the cycle time of what they term to be the Perceptual Processor and is itself based on earlier work on visual perception [8, 11]. These figures appear to be frequently used to develop a 100ms rule, by which delays greater than 100ms are considered to be unacceptable [16].

An early paper by Miller [18] estimates that 100ms delay is acceptable when using a light-pen to make slow, deliberate strokes, however this figure is presented without further reference and without explanation of how this value was determined. Miller does not present estimates of acceptable latency for faster light-pen strokes, nor any definition of ‘acceptable’.

More recently, Anderson, Doherty, and Ganapathy [2] conducted a study to determine qualitative responses to touchscreen latency during various tasks, such as web browsing, with the aim of determining latencies users find to be “acceptable”. The authors found that above a value of 580ms, users found the latency unacceptable, but noted that their experimental tasks were relatively brief (specifically: zooming, panning, and page-turning). We believe that these results are subjective, and are highly influenced by user expectations based on experience with current devices. As technology increases performance, user expectations will rise, along with clear improvement. Thus, we wished to move beyond what users currently find ‘acceptable’, and instead find the level of latency at which further improvement is no longer perceptible.

A series of works by Pavlovych and colleagues has examined the role of latency on other types of input devices including the mouse [19,20,21]. We build on this work by examining its role in direct-touch input.

Reducing perceived latency

Significant efforts are expended in software and hardware engineering to reduce the latency of a given input device or system. A complete survey of best practices in this field is beyond the scope of this paper, though we will describe some of these in section 3. Also relevant is work in user interface design intended to reduce *perceived* latency – that is, delays perceived by the user, rather than actual delays. Seow [27] argues that the goal of an engineer (be it software or hardware) should be to reduce the perceived latency of a system, which may only be partially a function of the actual latency of that system. Indeed, researchers have found that adding differentiated visual responses to touch input makes a system appear to have improved performance, even when the actual performance is unchanged (e.g.: [31]). While we value this approach, the focus of our efforts has been in understanding the value of reducing *actual* latency.

Touch Input Technologies

Direct touch input can be achieved using a variety of technologies, including resistive [22, 23], direct illumination [30], frustrated total-internal reflection [9], diffuse illumination [24], projective capacitive [3], and capacitive coupling [7]. One property these technologies share is that they contribute to the latency between user action and the system's response. The scale of this contribution varies across technology and implementation. Notable, however, is that in none of the papers cited above is there a measurement of the latency of the device being described.

LATENCY IN TOUCH

We define *touch-to-display latency* as the time delay between a touch action (such as the user moving a finger) and the resulting display change [26]. After testing numerous commercial devices using the method described in the Appendix, we found that, on modern touch systems, this is typically between 50 and 200ms. For example, we used a high-speed camera to examine this delay on an iPad. For this device, there is typically about 75ms of lag between the motion of the finger and the updating of the display. This number varies depending on various factors, including when in the scan and video cycles the user made the movement, the particular application, etc. High latency is particularly problematic on larger displays, such as the iPad, where users are more likely to perform gestures more quickly than with smaller, mobile-phone sized displays.

Direct-touch interfaces are particularly challenging because they make it easy for users to perceive latency. With a mouse or trackpad, a user compares his or her motion to visual perception. Since motor reflexes have 10s to 100s of milliseconds of delay themselves, it is difficult to make the comparison accurately. In contrast, direct manipulation interfaces make latency apparent within a single sensing modality – sight. If a dragged graphical object lags behind a moving finger, the user sees the finger and the misaligned object at the same time, independent of any delay in visual processing. This can even be seen in still photographs, such as in Figure 4. One might think that because the finger is moving, this misalignment will be difficult to notice. However, this is typically not the case. The human eye can easily track a finger moving at moderate speed, and thus the image of the finger and the misaligned graphic can be fairly stable on the retina.

The amount of misalignment between the finger and the tracking graphic can be considerable. For a finger moving at the moderately fast rate of 1 meter/second, a 100ms latency will cause the graphic to lag behind the finger by 100mm, or about 4 inches. At 10ms latency, this reduces to 10mm – about the width of a small finger. This is much better, but still quite noticeable. In designing our test system, we presumed that we would need to get down to about 1/10th of a finger width for the misalignment to be negligible. Thus, our goal was to produce a test system with a touch-to-display latency of 1ms.

There are three major components that contribute to latency in a touch system: 1) the touch sensor itself, 2) processing by the system software, UI toolkits, and ultimately the application, and 3) the display. Reducing the latency of direct touch systems requires understanding, and addressing, the latency introduced by each of these components.

Touch Sensor

Most modern multitouch systems use mutual capacitance sensors that measure the capacitive coupling from each row to each column on a 5-6mm grid. The change in this coupling due to the presence of a finger is very small, typically less than one picofarad. This is a very challenging measurement to make, particularly in the presence of extraneous electric fields such as those produced by an LCD display, backlight, etc. Allowing enough time for each measurement to average away the noise is critical.

Most current touch sensors run at a 60Hz scan rate. This means that each grid point is scanned every ~17ms. Typically, this raw data is then processed into touch points by an embedded touch controller, before being sent out over the communications link. These steps add further delay. Additionally, some touch controllers run at a slower scan rate until they detect a touch event in order to save power. This behavior adds additional delay at first contact, but does not impact tracking performance.

It would be fairly straightforward to decrease the latency of current touch technologies – particularly vision-based techniques, given that high speed cameras are nearly commoditized. However, it would involve running more hardware in parallel, increasing cost and power dramatically. To address this, low latency touch sensors will need to use region of interest, tracking, and related techniques to limit the number of points scanned.

It is worth noting that latency specifications can be extremely misleading. It is common in the industry to refer to frame rate rather than latency. They are not the same thing. If multiple frames are buffered, the latency can be quite high, even with a fast frame rate. Further, it is not uncommon for commercial devices to be advertised with “typical” delays, with no mention of worst case. Our research on this issue has been frustrated by the fact that the industry has not adopted a full disclosure policy that would require specifying the range of latencies and their distribution, hence the need to perform our own measurements.

Processing

Between the sensor and the display there is an arduous path that involves communications, the operating system, UI toolkits, the application layer, and ultimately, the graphics controller. Each of these adds latency.

The latency introduced by the operating system is highly variable. Windows, iOS, OSX, Android, etc. are not real time operating systems. Because of this, there is no guarantee a response will happen within a certain time period. If the processor is heavily loaded, it is possible for latency to increase dramatically. Some operations are handled at a very low level in the software stack, and have high priority. For example, the mouse pointer is typically highly optimized, so that even under heavy load, the perceived latency is relatively low. In contrast, an operation such as resizing a photo with two fingers requires constant rescaling of the image at the application level.

While application developers clearly have an incentive to make their programs seem responsive, there are in fact few constraints on the amount of latency they may introduce. UI threads typically dispatch input events one at a time, requiring that the application finish dealing with one input event before another can be attended to. While the input queue ensures that all input is addressed, this one-event-at-a-time model provides ample opportunity for application developers to introduce significant latency. Unfortunately, developers quickly see diminishing returns on their own efforts to increase application performance given the limits dictated by other latencies in the system.

Display

The display system (including the graphics system as well as the display itself) is another major contributor to latency. Again, an obsession with frame rates obscures the actual latency through the system. For example, a 60Hz monitor may include one or more frames of buffer in order to allow for sophisticated image processing effects. Some display devices, such as projectors, include double-buffering in the electronics, effectively doubling the display latency.

The desire for 3D televisions and reduced motion artifacts is driving the development of faster LCD's. 240Hz displays are now commercially available. Unfortunately, the physics of the liquid crystals themselves appear to make performance beyond 480Hz unlikely. In contrast, Plasma and OLED technologies can run at higher speeds, but face significant commercialization challenges.

In our work, we have developed a high-performance touch demonstration system. Through various engineering efforts, we have been able to reduce the latency in each of the input device, processing of touches, and the display, in order to enable our simple applications to run with a latency of ~1ms. We now describe those efforts.

A HIGH-PERFORMANCE TOUCH TEST SYSTEM

Our ultimate goal is to understand how user perception of a touch system is impacted by touch-to-display latency, and to begin the work of engineering systems which are able to produce appropriately fast responses. To do this, we needed a system capable of generating a wide range of latencies - particularly ones much shorter than those provided by current systems. Unfortunately, no such device is commercially available. Thus, we needed to build our own test system.

The requirements for our test system were very demanding. Presuming a finger moving at 1m/s, we wanted the maximum slippage of dragged-UI elements to be 1mm. This implies a latency of 1ms. To reach this level of performance, our system would require very high speed sensing and display – far faster than typical commercial systems. In addition, any processing would need to be extremely fast. The following sections describe the components of our implementation of such a high-performance touch system. Figure 2 shows a system-level block diagram of the system.

Touch Sensor

We utilized a proprietary multi-touch resistive touch sensor with an active area of 24 cm x 16 cm, with custom electronics that allows very high-speed operation. The delay through this sensor is slightly less than 1ms, with no theoretical upper bound on detected movement speed. Touch data is transmitted serially over an optical link.

Projector

For the display, we chose the DLP Discovery 4100 kit based on Texas Instruments' Digital Light Processing technology [28]. The system utilizes front-projection onto the touch sensor, eliminating parallax error that might disturb a user's perception of finger and image alignment. A DLP projector uses a Digital Micromirror Device (DMD), a matrix of mirrors which turn pixels on or off at very high speed. Typically, the high speed of the mirrors is used to change the percentage time on vs. off to create continuous colored images. If one gives up this capability and only allows simple binary images, these can be produced at an extraordinary rate. Our projector development system displays 32,000 binary frames/second at 1024x768 resolution with latency under 40 μ s. To achieve this speed, the video data is streamed to the DMD at 25.6 Gbps.

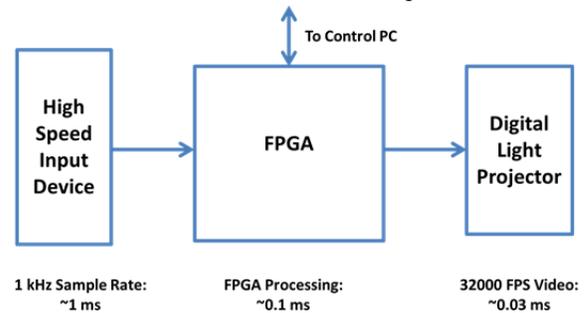


Figure 2: System block diagram of the high-performance touch test system with estimated latency contributions of the individual components.

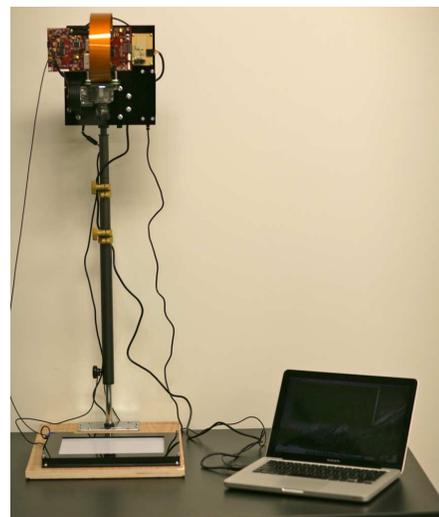


Figure 3: Photograph of device, with the control computer at right. The control computer is used on-ly to set behaviour parameters for the device.

FPGA Processing

To achieve minimal latency, all touch processing is performed on a dedicated FPGA – there is no PC or OS between the touch input and the display output. We used the DLP kit’s onboard XC5VLX50 application FPGA for processing the touch data and rendering the video output. To simplify user testing, a USB serial connection to the FPGA allows parameters to be changed dynamically. Latency can be adjusted from 1ms to several hundred ms with 1ms resolution, and different testing modes can be activated. This port also allows touch data to be collected for analysis. Figure 2 shows a block diagram of the FPGA design.

To receive touch data from the sensor, the system communicates through a custom high-speed UART. To minimize the latency, we use a baud rate of 2 Mbps, the highest baud rate we can use without losing signal integrity due to high frequency noise across the communication channel.

The individual bytes of compressed touch data are then processed by a touch detection finite state machine. This FSM simultaneously decodes the data and performs a center-of-mass blob-detection algorithm to identify the coordinates of the touches. The system is pipelined such that each iteration of the FSM operates on the last received byte such that no buffering of the touch data occurs.

The touch coordinates are then sent to a 10-stage variable delay block. Each delay stage is a simple FSM with a counter and takes a control signal that indicates the number of clock cycles to delay the touch coordinate, allowing us to test various levels of latency. The delay block latches the touch sample at the start of the iteration and waits for the appropriate number of cycles before sending the sample and latching the next. The delay block therefore lowers the sample rate by a factor of the delay count. To keep the sample rate at a reasonable level, 10 delay stages are used, so that, for example, to achieve 100ms of latency, the block waits 10ms between samples for a sample rate of 100Hz.

To run basic applications, we chose to incorporate a MicroBlaze soft processor to render the display. Our first prototype originally used a hardcoded control FSM in place of the MicroBlaze for absolute maximum performance, but due to the inflexibility of a hardcoded FSM, we decided to use a soft processor instead. The MicroBlaze is a 32-bit Harvard architecture RISC processor optimized to be synthesized in Xilinx FPGAs [33].

The MicroBlaze soft processor instantiation allows us to select only the cores, peripherals, and memory structures required for our application. For our system, in addition to the base MicroBlaze configuration, we include an interrupt controller, GPIOs for the touch data, a GPIO to set the variable latency, a BRAM memory controller for the image buffer, and a UART unit to communicate with a PC. The MicroBlaze is clocked at 100 MHz.

The MicroBlaze uses an interrupt system to detect valid touch coordinates. A touch ready interrupt event is generated when valid touch data arrives on the GPIOs from the delay block, and the corresponding image is written to the image buffer. Because of the non-uniform nature of an interrupt-based system, the exact latency cannot be computed, but it is insignificant in comparison to the 1ms latency due to the input device.

The image buffer is synthesized in the on-chip BRAM blocks. These blocks provide a dual-port high-speed configurable memory buffer with enough bandwidth to support our high frame-rate display. The image buffer is clocked at 200 MHz with a bus width of 128 bits for a total bandwidth of 25.6 Gbps, as needed by the DLP.

Finally, the DMD controller continuously reads out frames from the image buffer and generates the signals with appropriate timing to control the DMD. This controller can be changed to vary the frame rate in future testing.

Tools

The FPGA design was done using the Xilinx ISE Design Suite 13.2. The MicroBlaze was instantiated and configured using the Xilinx Platform Studio (XPS), and the MicroBlaze was programmed in the C language through the Xilinx Embedded Development Kit (EDK). The image buffer BRAM was instantiated and configured using the Xilinx Core Generator. All VHDL simulations were done using the Xilinx ISim simulator. Synthesis was done using the Xilinx Synthesis Tool (XST).

Having developed a device capable of extremely low latency, we wished to understand precisely what levels of latency would achieve perceptibly improved responses. Thus, we conducted a study to measure the latencies which users are able to perceive while performing drag input.



Figure 4: Our high-performance touch device, capable of sensing and responding to input at 1 kHz with only 1 frame of latency. Demonstrating dragging an on-screen object with the same velocity, with latency (from left to right) of 100ms, 50ms, 10ms, and 1ms.

PERCEPTION OF LATENCY IN DRAGGING

While earlier work has demonstrated that users are able to perceive latency down to 100ms, our own use of our prototype device indicated that latency was apparent at far lower levels – as shown in Figure 4. In order to guide our efforts in reducing latency, we wished to understand at precisely what latency value users would no longer notice a difference in performance when performing drags. Thus, we conducted a formal experiment in which we modulated the latency of the system, and measured participants' ability to perceive the differences in performance.

Just Noticeable Difference

We wished to determine the precise level of performance that users are able to perceive when performing common tasks on a touch screen interface. To that end, we conducted an experiment to determine the just-noticeable difference (JND) of various performance levels. The JND is defined as the threshold level at which a participant is able to discriminate between two unequal stimuli – one consistently presented at the same level, termed the reference, and one whose value is changed dynamically throughout the experiment, termed the probe [15]. A commonly accepted value for the JND at some arbitrary reference value is a probe at which a participant can correctly identify the reference 75% of the time [15]. A probe value that cannot be distinguished from the reference with this level of accuracy is considered 'not noticeably different' from the reference.

We conducted a within-subjects experiment to determine the JND level of the probe latency when compared to our maximum performance of 1ms of latency, which served as the reference. While such a determination does not provide an absolute value for the maximum perceptible performance, it does serve as our 'best case' floor condition against which we are able to measure other levels of latency. Our hypothesis was:

H1: Participants will be able to discern latency values that are significantly lower (<20ms) than current-generation hardware provides (~50-200ms).

Apparatus

Our experimental setup, shown in Figure 3, incorporated the previously described system, controlled by a laptop via a USB serial connection. The laptop runs custom software written for the purpose of generating the latency value staircases in real time; each user response is entered and logged on the laptop, and used to generate the following trial's latency level. The experimental task is shown in Figure 4.

Participants

Ten right-handed participants (3 female) were recruited from the local community. Ages ranged between 24 and 40 (mean 27.80, standard deviation 4.73). All participants had prior experience with touchscreen devices, and all participants owned one or more touch devices (such as an iOS- or Android-based phone or tablet). Participants were paid \$20 for their participation.

Procedure

Participants were repeatedly presented with pairs of latency conditions: the reference value (1ms) and the probe (between 1 and 65ms of latency). Participants dragged their finger from left to right, then right to left on the touchscreen display. While any dragging task would have been suitable, left/right movements reduce occlusion in high-latency cases. Participants were asked to move in both directions to ensure they did not 'race through' the study. Beneath the user's contact point, the system rendered a solid white 2cm × 2cm square. The speed of movement was left to be decided by the participants.

The order of the conditions was randomized for each pair. The study was designed as a two-alternative forced-choice experiment; participants were instructed to choose, within each trial, which case was the reference (1ms) value and were not permitted to make a "don't know" or "unsure" selection [25]. After each pair, participants informed the experimenter which of the two was "faster".

Design

In order for each trial to converge at our desired JND level of 75%, the amount of added latency was controlled according to an adaptive staircase algorithm, which is a more recent and now broadly adopted approach than previous staircase and modified staircase algorithms [15]. Each correct identification of the reference value caused a decrease in the amount of latency in the probe, while each incorrect response caused the probe's latency to increase. In order to reach the 75% confidence level, increases and decreases followed the simple weighted up-down method described by Kaernbach, wherein increases had a three-fold multiplier applied to the base step size, and decreases were the base step size (initially 8ms) [13]. When a participant responded incorrectly after a correct response, or correctly after an incorrect response, this was termed a reversal as it caused the direction of the staircase (increasing or decreasing) to reverse. The step size, initially 8ms, was halved at each reversal, to a minimum step size of 1ms. This continued until a total of 10 reversals occurred, resulting in a convergence at 75% correctness.

Each participant completed eight staircase 'runs'. Four of these started at the minimum probe latency (1ms) and four at the maximum (65ms). The higher starting value of the staircase was chosen because it roughly coincides with commercial offerings, and because pilot testing made it clear that this value would be differentiated from the 1ms reference with near 100% accuracy, avoiding ceiling effects.

Staircases were run two at a time in interleaved pairs to prevent response biases that would otherwise be caused by the participants' ability to track their progress between successive stimuli [15]. Staircase conditions for each of these pairs were selected at random without replacement from 8 possibilities (2 starting levels × 4 repetitions). The entire experiment, including breaks between staircases, was completed by each participant within a single 1-hour session.

Results

Our study was designed to find the just-noticeable difference (JND) level for latency values greater than 1ms. This JND level is commonly agreed to be the level where the participant is able to correctly identify the reference 75% of the time. Participant JND levels ranged from 2.38ms to 11.36ms, with a mean JND across all participants of 6.04ms (standard deviation 4.33ms). JND levels did not vary significantly across the 8 runs of the staircase for each participant. Results for each participant appear in Figure 5.

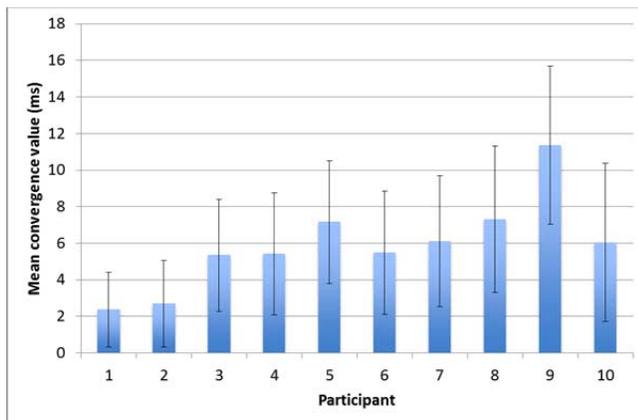


Figure 5: Mean JND by participant. Bars show std-dev.

Discussion

Our results have confirmed our hypothesis. Participants were able to discern differences in latency far below the threshold of current consumer devices [H1] (50-200ms). It should be noted that a baseline of 1ms was a limitation of our experimental hardware, and it is possible that our results are therefore exhibiting the effects of this technical floor; further experimentation is necessary to determine if participants' JND values would be even smaller than those shown here, given a faster reference value.

We should also note that our participants were likely often determining latency by estimating the distance between the onscreen object and their finger as it was moved around the touchscreen; this is an artifact of an input primitives used in modern UIs (specifically, dragging). We have no doubt that testing a different input primitive (tapping, for example) would exhibit different perceptions of latency.

Our results confirm that an order-of magnitude improvement in latency would be noticed and appreciated by users of touch devices. Thus, we set-out to design a software interface which would enable application developers to continue to use toolkit-based application design processes, but enable those toolkits to provide feedback at extremely low latencies, given the presence of a low-latency system.

A TOUCH ACCELERATION TOOLKIT

As we have discussed, input and display devices can conceivably be made in the relatively near term which could enable extremely low-latency interaction. The apparent remaining bottleneck, therefore, is the creation of software that can be executed with sufficiently low latency.

An obvious first step would be to hardware-accelerate application software. While this may be technically feasible, it would require application software be written in a fundamentally new way, foregoing long-standing toolkits and development practices.

We propose a middle-ground, which builds on the model-view-controller model of UI development, upon which many modern UI toolkits are based [14]. The MVC holds that application logic can be separated from the visual representation of the application. In our modified version of MVC, a second, overlaid view is provided for the application. In particular, touch input receives an immediate response from the UI controls, which is based in part on the state of the application at the time the touch is made. Our goal is not to reduce overall system latency, but rather to provide immediate responses that are nonetheless contextually linked to the underlying application.

We aim to extend previous work on application-independent visual responses to touch [31]. While a straightforward port of Ripples might be feasible, a limitation of the visualizations shown in that work is that they are completely separate from even the visual elements of the UI, adding visual complexity [32]. Instead, we aimed to more fully integrate a set of visual responses into the UI elements themselves, so as to reduce visual complexity. Thus, unlike Ripples, where the particular visuals shown provide a *de facto* "mouse pointer" for touch, our goal is to integrate high performance responses into the controls themselves, providing a more unified visualization. We thus designed Accelerated Touch interactions, wherein the traditional direct-touch software runs as it would normally, in a high-latency visual layer, while an additional visual layer contains immediate visual feedback, customized for the UI element at a very low latency.

In our prototype, these two layers are combined by superimposing two projected images – one from the low-latency touch device, and a second from a traditional projector connected to a desktop computer running custom touch software, receiving input from the low-latency subsystem. A commercial implementation would doubtless be architected such that the low-latency layer is generated in hardware, operating directly on touch sensor data, with a high bandwidth link to the display hardware. Our two-display solution is meant only to serve as a demonstrator.

To facilitate future implementations, we now describe our Accelerated Touch user interface designs. We begin by first describing the design constraints which guided our design process, and then present our final designs. We then describe the architecture of our implementation.

User Interface

Design Constraints

In designing a user interface for the low-latency subsystem, we considered a number of constraints:

Information: The low-latency layer has limited information about the current state of the application software; in particular, it is limited to knowledge of the state of the application prior to the beginning of a touch event. This is because any closed loop between the high and low-latency subsystems would necessarily add lag.

Performance: The response of the low-latency layer must be generated very quickly. Even with hardware acceleration, the design of visual responses must be carefully performance-driven.

Fidelity: The low-latency layer is restricted to generating simple visualizations, to the degree that they can be customized at the time of development of the low-latency visual responses. In the case of our prototype, we had the additional constraint that the visuals could be only monochromatic and limited visual primitives. Constraints of this type can be introduced by the acceleration hardware, or by the display hardware.

Non-Interference: Given that some of the application’s response will be generated in the low-latency, hardware accelerated layer, and some in the high-latency software layer, a key consideration is how the two will be blended to provide a ‘seamless’ response to the user’s input. Low-latency responses must not interfere with any possible application response, which will necessarily occur later.

Design

We conducted a design process to create a set of UI controls with differentiated low and high latency visual responses to touch. To begin, we sought to find a metaphor which would enable a seamless transition between the two layers of response. These visualizations included such information as object position and state. Our designs were culled based on feasibility under the above constraints.

Our final design was based on a heads-up display (HUD) metaphor, similar to the visualizations used in military aircraft. The HUD was suitable, since traditional HUDs are geometrically simple, and thus was easy to implement such at an authentic fidelity. The HUD, notably, is another example of two visual layers being combined, though in the case of many HUDs, a computerized display is superimposed on video or the real world itself [10]. Accordingly, they are designed to be non-interfering.

Based on the HUD metaphor, we developed a set of touch-event and UI element-specific low-latency layer visualizations for a set of UI elements found in many direct-touch systems. These 3 elements are both common and representative; their interactions (taps, drags, two-finger pinching) cover the majority of the interaction space used in current direct-touch devices. The low-latency responses we developed are described in Table 1, and they are shown in Figure 6 and in our video figure.

ELEMENT	TOUCH DOWN	TOUCH MOVE	TOUCH UP
Button	Outline	(none)	If within bounds, 2 nd outline, else none.
Draggable / Resizable	Bounds outlined	Outline changes / moves	Outline fades when high-latency layer catches up
Scrollable View	List item outlined	List edges highlight to scroll distance.	Edge highlights fade as high-latency layer catches up

Table 1: Accelerated visuals for each element and touch event, which compliment standard high-latency responses to touch input.

These 3 elements represent broad coverage of standard UI toolkits for touch input. Most higher-order UI elements are composed of these simpler elements (eg: radio buttons and checkboxes are both ‘buttons’, a scrollbar is a “draggable/resizable” with constrained translation and rotation).

Our Accelerated Touch system depends on the marriage of visuals operating at two notably different latency levels; this latency difference has been incorporated into the design of our low-latency visualizations. Users are informed of the state of both systems, with a coherent synchronization as the visual layers come into alignment.

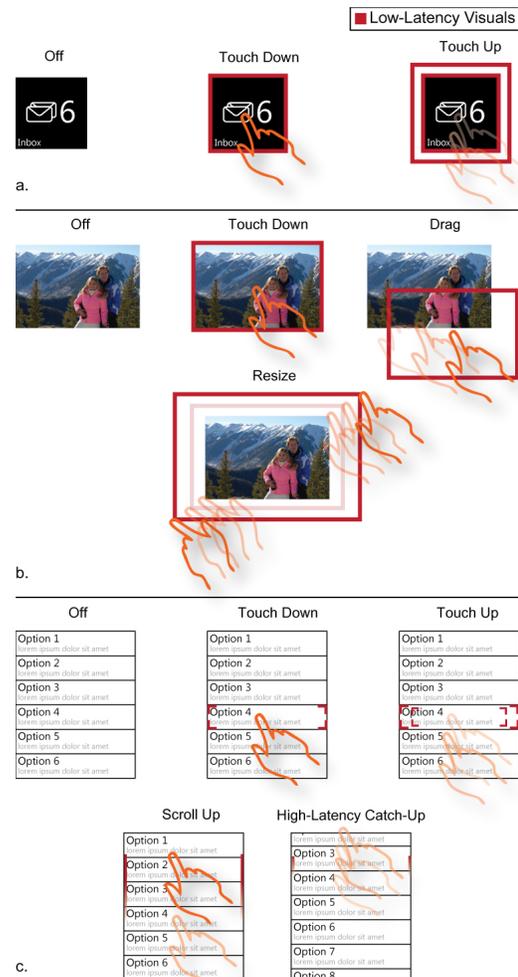


Figure 6: Our element types: a. buttons, b. draggable / resizable elements, c. scrolling viewports.

Architecture

We envision that the application developer utilizing a toolkit such as the one we have described will build their application through the normal process of assembling GUI controls. Upon execution, the UI elements would bifurcate their visualizations, with high- and low-latency visualizations rendered and overlaid on a single display. Information flow through our system is as shown in Figure 7.

As we have described, this bifurcation creates a fundamental communication problem where any parameterization of visualizations provided by the low-latency subsystem required by application logic must be defined before the user begins to give input. Any visualization which requires processing by the application will introduce a dependency of the low-latency system upon the high-latency system, and will therefore introduce lag back into the system.

Therefore it was critical for us to build UI element logic into the low-latency subsystem. Between user touches, the application, which is executing in the high-latency subsystem, has the opportunity to provide parameters for the low-latency subsystem's model of the UI elements. As we have described, our approach was to extend the MVC model of UI software design, by providing a separate controller responsible for low-latency feedback. In our software design, the following can be specified for each control:

- Element type (*button, draggable object, scrollable list*)
- Bounding rectangle dimensions (*x, y, width and height*)
- Conditional: additional primitive information (for example: *size of list items*, in the case of a scrollable list)

Logic for a given element-type's response to touch input is stored in the low-latency subsystem. In future implementations further parameterization of the low-latency subsystem's responses to user input could be communicated in the same manner, allowing a greater degree of customization.

In our implementation, sensor data is processed on the FPGA to generate events, which are then separately distributed to the low-latency and high-latency subsystems, as shown in (Figure 7). The low- and high-latency subsystems' response to user input is therefore independent but coordinated. In this implementation, the high-latency subsystem acts as the 'master', setting state of the low-latency subsystem between user touches. Future implementations might change this relationship to increase synchronization between the two subsystems, or offload more processing to the low-latency subsystem.

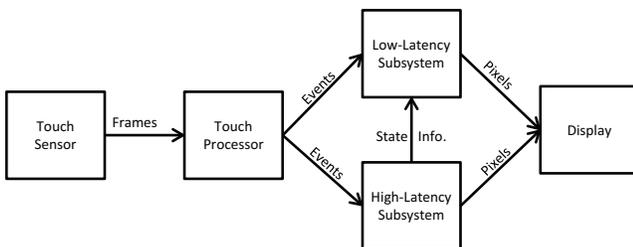


Figure 7: The Accelerated Touch architecture

Synchronization

Communication between the high-latency and low-latency layers is of critical importance. Three issues must be considered in determining how the high- and low-latency subsystems remain synchronized.

Latency differences: Our low-latency visualizations require information about the latency difference between the high- and low-latency visual layers in order to synchronize visualizations. In our system these latency values are static, and thus preprogrammed into the FPGA. In implementations where latency levels vary in either subsystem, it may be advantageous to fix the latency level at an always-achievable constant rather than having a dynamic value that may become unsynchronized.

Hit testing: Hit testing decisions are conditional on additional data regarding the visual hierarchy and properties of visible UI elements. Our implementation resolved this consideration by disallowing overlapping bounding rectangles, requiring a flat, 'hit test friendly' map of the UI. Other implementations may instead provide the necessary information (object state, z-order, listeners) to the low-latency subsystem and have both subsystems conduct hit testing in parallel, or conduct it solely in the low-latency subsystem, providing the results to the high-latency subsystem.

Conditional responses: Many interface visualizations are conditional not only on immediate user input, but on further decision-making logic defined in the application logic. The degree to and method by which application logic might be migrated to the low-latency subsystem is a topic requiring further research. Two illustrative examples of conditional response logic are as follows:

Consider a credit-card purchase submission button, which is programmed to be disabled (to prevent double-billing) when pressed, but only upon validation of the data entered into the form. In such a case, the behavior of the button is dependent not only on an immediate user interaction, but is further conditional on additional information.

Consider also linked visualizations, such as the one shown in Figure 8. In this case, feedback is provided to the user not only by the UI element they are manipulating, but also by a second UI element.

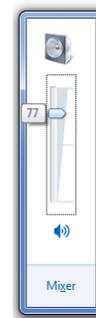


Figure 8: The Windows 7 volume control. When dragging the slider, a tooltip appears showing a numeric representation of the current setting.

These relatively straightforward examples could easily be programmed directly into the low-latency subsystem, even in our relatively primitive FPGA-based demonstrator system. However, how these might be described in general terms that do not require application-specific development within the low-latency subsystem is an open question, and one that is critical to answer to allow developers to continue to use traditional development methods when creating software in an Accelerated Touch-like toolkit.

CONCLUSION & FUTURE WORK

In this paper, we have described sources of latency, and demonstrated how several of these can be eliminated in building a demonstrator system capable of 1ms touch latency. Further, we have described the results of tests which showed that users were able to perceive order-of magnitude improvements in latency over current-generation hardware. Our results suggest that performance beyond 1ms may still yield improvement that is perceptible to users.

We have constructed a prototype Accelerated Touch system, wherein a traditional direct-touch layer is paired with a low-latency layer that displays nearly immediate visual feedback on user interaction, independent of application logic, but visually tied to the underlying UI widget. We have further described the design of this visual language to satisfy the various constraints of a dedicated low-latency touch processor, and we have described a potential architecture for a direct-touch system that pairs Accelerated Touch with more traditional touch interaction.

A common complaint that we heard from people who used our system extensively was that it “broke” them – that they now find the latency of current generation devices completely unacceptable. The implication is that improving latency might be an effective competitive strategy for device vendors. It is our hope that this paper will spark innovation in the design of hardware and software capable of lower latency of response to user input.

We see a wealth of future work in further investigating the limits of human perception of touchscreen computer systems, and better understanding the effect of performance parameters such as latency on the usage of touchscreens. For example, are there performance benefits for input under reduced latency? Further, we have conflated latency and frame rate, future devices may decouple these two parameters, and could optimize one or the other; further investigation is needed into the effects of such a change.

ACKNOWLEDGEMENTS

We thank our experimental participants, the UIST reviewers for their helpful comments, our UIST AC for meticulous feedback, and Ricardo Jota for assistance with the video. We also thank the members of the Dynamic Graphics Project at the University of Toronto, and the Applied Sciences group at Microsoft for valuable discussions.

APPENDIX: MEASURING LATENCY

We use a simple technique for measuring the touch-to-display latency. The device is placed in a mode where a graphic is meant to follow a touch point. (eg: inking, translating an icon). A ruler is placed on top of the device. A finger is then dragged across the screen along the ruler edge at as constant a rate as possible. A high speed camera of known frame rate captures the finger motion as well as the resulting graphical response. By examining individual frames in the video, we can calculate the speed of the finger (distance travelled between successive frames x frame rate). The amount of separation between the finger and the following graphic can be directly observed. The latency is the separation distance / the finger speed. We take two measurements: worst case latency, and best case latency.

We used an inexpensive consumer camera – a Casio EX-FH100 High Speed Exilim – to record 240 fps video at 448x336. This is significantly faster than most display systems, thus the graphic can be seen to jump in discrete increments, while the finger moves between every frame. In addition to this type of variation, latency can also vary with different locations on the screen, OS load, etc.

Figure 9 shows two images taken 12 frames apart from our testing. In the example shown in the figure, the fingers have moved 1.8cm over 12 frames (0.05 seconds, 36 cm/s). The lag distance is 4.2cm, thus the worst-case latency is computed as 116.7 ms.

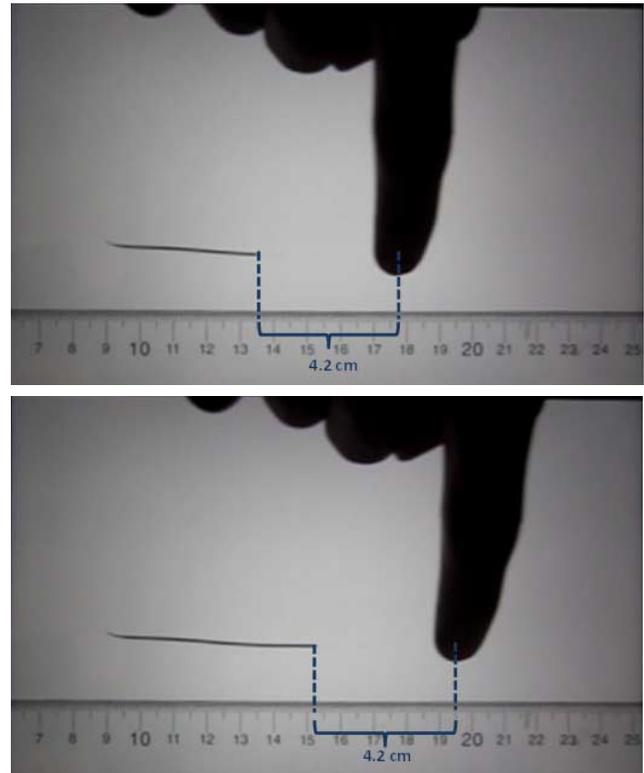


Figure 9. Images of Autodesk Sketchbook Pro running on an iPad 3.

Best-case latency is computed by finding the frame showing the shortest lag distance (among the 10 frames between the samples shown), and assuming the same speed of finger movement. In the case of our example, the best-case lag distance was 2.7cm (shown in Figure 10), thus yielding a best-case lag of 75ms.

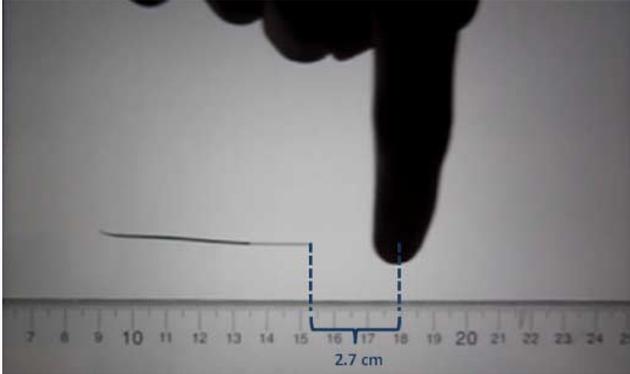


Figure 10. Shortest lag distance among the frames of video between the ones shown in Figure 9.

REFERENCES

1. Accot, J., and Zhai, S. 1997. Beyond Fitts law: models for trajectory based HCI tasks. *Proceedings of ACM CHI'97*, 295-302.
2. Anderson, G., Doherty, R., and Ganapathy, S. 2011. User Perception of Touch Screen Latency. *Design, User Experience, and Usability. Theory, Methods, Tools, and Practice*. 195-202.
3. Boie, R.A. 1984. Capacitive Impedance Readout Tactile Image Sensor. *Proceedings of the 1984 IEEE International Conference on Robotics and Automation*, 370-378.
4. Card, S. K., Mackinlay, J. D., and Robertson, G. G. 1991. The Information Visualizer: An Information Workspace. *Proceedings of (CHI '91)*, 181-188.
5. Card, S.K., Moran, T.P., and Newell, A. 1983. *The Psychology of Human-Computer Interaction*. Erlbaum, Hillsdale, NJ, USA.
6. Cornsweet, TN. 1962. The staircase-method in psychophysics, *American Journal of Psychology*, 75, 485-491.
7. Dietz, P., and Leigh, D. 2001. DiamondTouch: multi-user touch technology. *Proceedings of the UIST' 01*, 219-226.
8. Ganz, L. 1975. Temporal factors in visual perception. In E. C. Carterette and M. P. Friedman, eds. *Handbook of Perception, Vol. V, Seeing*. New York: Academic Press
9. Han, J. 2005. Low-cost multi-touch sensing through frustrated total internal reflection. *Proceedings of UIST '05*, 115-118.
10. Harrison, B.L. and Ishii, H. and Vicente, K.J. and Buxton, W.A.S. 1995. Transparent layered user interfaces: An evaluation of a display design to enhance focused and divided attention. *Proceedings of CHI '95*, 317-324.
11. Harter, M. R. 1967. Excitability and cortical scanning: A review of two hypotheses of central intermittency in perception. *Psychological Bulletin* 68, 47-58.
12. Jacob, R. J. K., Girouard, A., Hirshfield, L. M., Horn, M. S., Shaer, O., Solovey, E. T., and Zigelbaum, J. 2008. Reality-based interaction: a framework for post-WIMP interfaces. *Proceedings of CHI '08*, 201-210.
13. Kaernbach, C. 1991. Simple adaptive testing with the weighted up-down method, *Perception & Psychophysics* 49, 227-229.
14. Krasner, G., Pope, S. 1988. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *J. Object Oriented Program.* 1, 3, 26-49.
15. Levitt, H. 1971. Transformed up-down methods in psychoacoustics. *Journal of the Acoustical Society of America*, 49(2): 467-477.
16. MacKenzie, I.S., and Ware, C. (1993). Lag as a determinant of human performance in interactive systems. In *Proceedings of CHI '93*, 488-493.
17. Microsoft Corporation. 2011. Windows Developer Preview: Win 8 Guide. <http://windows.microsoft.com/en-US/windows-8/preview>, January 2012.
18. Miller, R. B. 1968. Response time in man-computer conversational transactions. *Fall Joint Comp. Conf.* 1968, 267-277.
19. Pavlovykh, A. C. Gutwin, (2012) Assessing Target Acquisition and Tracking Performance for Moving Targets in the Presence of Latency and Jitter, *GI 2012*, to appear.
20. Pavlovykh, A., W. Stuerzlinger, (2009) The Tradeoff between Spatial Jitter and Latency in Pointing Tasks, *ACM SEICS 2009*, 187-196.
21. Pavlovykh, A., W. Stuerzlinger, (2011) Target Following Performance in the Presence of Latency, Jitter, and Signal Dropouts, *GI 2011*, 33-40.
22. Potter, R. Weldon, L. J., and Shneiderman, B. 1988. Improving the accuracy of touch screens: an experimental evaluation of strategies. In *Proceedings of CHI '88*.
23. Rosenberg, I. and Perlin, K. 2009. The UnMousePad: an interpolating multi-touch force-sensing input pad. In *Proceedings of ACM SIGGRAPH 2009 (SIGGRAPH '09)*.
24. Rubine, D., McAvinney, P. 1988. The Videoharp. In *Proceedings of International Computer Music 1988*. Computer Music Association. September, 1988.
25. Savage, W. C. 1970. *The Measurement of Sensation: A Critique of Perceptual Psychophysics*. Berkeley: UC Press.
26. Sears, A., Jacko, J. 2012. *HCI Handbook, 3rd Edition*. Ch. 9.

27. Seow, S. 2008. *Designing and Engineering Time: The Psychology of Time Perception in Software*. Addison-Wesley, NY.
28. Texas Instruments. DLP Discovery 4100 Development Kit. <http://www.ti.com/tool/dlpd4x00kit>, January 2012.
29. Van Kessel, P. F., et al. 1998. A MEMS-Based Projection Display. In *Proceedings of the IEEE*, 86 (8), pp. 1687-1704.
30. Wellner, P. 1993. Interacting with paper on the Digital Desk. *Communications of the ACM* 36 (7), 87-96.
31. Wigdor, D., Williams, S., Cronin, M., Levy, R., White, K., Mazeev, M. and Benko, H. 2009. Ripples: utilizing per-contact visualizations to improve user interaction with touch displays. In *Proceedings of UIST '09*. ACM, New York, NY, USA, 3-12.
32. Wigdor, D., Wixon, D. (2011). *Brave NUI World: Designing Natural User Interfaces for Touch and Gesture* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
33. Xilinx. Microblaze Processor Reference Guide. http://www.xilinx.com/support/documentation/sw_manu als/xilinx13_3/mb_ref_guide.pdf, January 2012.