# Looking through the Eye of the Mouse: A Simple Method for Measuring End-to-end Latency using an Optical Mouse

**Géry Casiez[1], Stéphane Conversy[2], Matthieu Falce[3], Stéphane Huot[3] & Nicolas Roussel[3]**
[1]Université de Lille, [2]Université de Toulouse - ENAC, [3]Inria Lille
gery.casiez@univ-lille1.fr, stephane.conversy@enac.fr
{matthieu.falce, stephane.huot, nicolas.roussel}@inria.fr

## ABSTRACT
We present a simple method for measuring end-to-end latency in graphical user interfaces. It consists in positioning an unmodified optical mouse on the screen while displaying and translating a particular texture to fake mouse displacements resulting in controlled mouse events. The method works with most optical mice and allows accurate and real time latency measures up to 5 times per second. In addition, the technique allows easy insertion of probes at different places in the system – *i.e.* mouse events listeners – to investigate the sources of latency. After presenting the measurement method and our methodology to find the particular texture to display, we detail the measures we performed on different systems, toolkits and applications. Results show that latency is affected by the operating system and system load. Substantial differences are found between C++/GLUT and C++/Qt or Java/Swing implementations, as well as between web browsers.

### Author Keywords
Latency; lag; latency measure; latency jitter; computer mouse.

### ACM Classification Keywords
H.5.2 User Interfaces: Input devices and strategies

## INTRODUCTION
The end-to-end latency (or *lag*) of a graphical user interface is commonly defined as the duration between a user action (*e.g.* movement of an input device or human limb) and the corresponding on-screen visual feedback (*e.g.* cursor movement, object displacement). All interactive systems have some latency introduced by input and output devices, network, events handling and processing. Probably due to the difficulty to measure it and to the multiple factors that can influence its variability, latency is only occasionally reported on input or output devices datasheets, interactive systems specifications or HCI experiments reports.

Latency and its variation over time, called *latency jitter*, are known for a long time to affect human performance and qualitative perception [11, 19, 15, 7]. Latency with computer mice has been studied in Fitts' law experiments where it has been shown to have an interaction with the task index of difficulty [11, 14, 18]. A latency of around 50 ms is known to affect performance in mouse-based pointing tasks and latency jitter above 20-40 ms is likely to be noticed [7]. The effect is even more pronounced in direct touch interaction, where it has been shown that latency as low as 2 ms can be perceived and that performance is affected from 20 ms [13, 8].

Considering its importance in touch/direct-interaction systems, researchers have developed methods to ease its measurement [4]. However, there is no such easy and affordable measurement method for traditional mouse-based interaction. The most common procedure to measure end-to-end latency is to use an external camera to record both the input device (or finger) and the associated response on screen, then to analyze the video to match the two related events in order to count the number of elapsed frames between them [10, 19, 13, 17, 14, 18]. Since this process is generally done by hand, getting one measure of latency is already a tedious and error prone process, whatever the interactive system. Getting repeated measures is thus cumbersome and time consuming, which can be one of the reasons why latency is rarely measured and mentioned in HCI experiment reports.

We believe that as HCI researchers, it is essential to measure and report the average latency and the latency jitter observed during our experiments, to enable more accurate replication, but also in case latency reveals to be a confounding factor (especially in Fitts' law and reaction time experiments). For

**Figure 1: Sample setup used, with a Logitech MX310 mouse positioned on a horizontally-oriented laptop display. A particular texture is displayed and moved under the mouse sensor to fake displacements. Other mice discussed in the paper are positioned around the display.**


*UIST 2015*, November 8–11, 2015, Charlotte, NC, USA.

http://dx.doi.org/10.1145/2807442.2807454

practitioners and users, it is also important to provide methods to determine and troubleshoot how hardware, toolkits, softwares and parameters tuning can affect the latency of interactive systems. As suggested in [6], manufacturers and application programmers would be able to make sure that a relative gain in latency might be perceived by users to enhance the overall user experience. For that, we need a simple and affordable measurement method but that accounts for all the "real-world" parameters that are prone to influence latency.

The measurement method we introduce, consists in positioning an unmodified optical computer mouse on a monitor, displaying and moving a particular texture to fake a mouse displacement, and sampling the corresponding mouse events. Latency is then measured as the time elapsed between the texture update and the movement of the on-screen pointer. This method contributes to:

1 - a lightweight method to measure end-to-end latency in real time;

2 - the possibility to perform multiple measures up to 5 times per second in real time;

3 - the ability to insert probes at different levels of the system – *e.g.* drivers, system pointer, toolkits or APIs – thanks to the use of a common clock (which is hardly possible with an external measurement system such as a video camera). This allows determining where the latency comes from;

4 - the measurement of latency and jitter on different operating systems, toolkits and in different conditions.

## RELATED WORK

We first detail previous work showing the impact of latency and latency jitter on performance before presenting existing latency measurement techniques.

### Impact of latency on performance

MacKenzie and Ware evaluated how latencies of 8.33, 25, 75 and 225 ms affect performance using a mouse in a pointing task of index of difficulties (IDs) up to 6.02 bits [11]. They showed a significant interaction between lag and the ID with an increasing degradation of the performance as the tasks get more difficult, the degradation being particularly important for latencies of 75 and 225 ms. In their experiment, they introduced additional latency to the system but they did not measure and report the actual one. In a Fitts' law experiment using a computer mouse, Pavlovych and Stuerzlinger also observed a more pronounced impact of latency with smaller targets, but did not notice any drop of performance up to 58 ms of latency when acquiring targets as small as 14 pixels wide [14]. In a similar experiment with targets of 12 pixels, Teather *et al.* measured a significant 15% decrease of performance between 35 and 75 ms end-to-end latency [18].

Concerning latency jitter, Ellis *et al.* have shown that users are able to detect a change in latency of less than 33 ms and probably less than 16.7 ms, independently of the base latency [7]. Pavlovych *et al.* measured the effect of both latency and latency jitter in a target following task [15]. Using a 35 pixels target, they showed that errors increase very quickly for latencies above 110 ms and latency jitter above 40 ms.

On touch systems, it has been shown that users are able to perceive a latency of 2 ms (6 ms in average) during a dragging task [13]. For tapping tasks, performance starts to decrease from 20 ms latency [8].

In summary, latency around 50 ms impacts performance in mouse-based pointing tasks, with a more pronounced effect as target size decreases (minimum target size evaluated: 12 pixels). Variations of latency (latency jitter) above 20-40 ms are also likely to be noticed and detrimental to performance.

### Latency measurement techniques

The classical approach to measure latency is to record a video of both the input device and graphical output and then to analyze the images in order to measure the end-to-end latency [10, 19, 13, 17, 14, 18].

Liang *et al.* measured the latency of a Polhemus Isotrak sensor by mounting it on a pendulum [10]. They used a video camera to record the periodic motion of the pendulum together with the timestamps of the device events displayed on the screen. They measured latency by playing back the video frame by frame, allowing them to determine the on-screen timestamp corresponding to the pendulum neutral position. They then processed their events log file to determine the timestamp of the associated neutral position. The difference between the two timestamps provided the overall latency.

Ware and Balakrishnan used a similar setup to measure the latency of a Polhemus Isotrak and a Logitech ultrasonic sensor [19]. Their replaced the pendulum by a stepper motor driven pulley assembly mounted on top of the computer monitor. They moved the attached sensor back and forth, at a constant speed. The latency was measured by playing back the video frame by frame, measuring the distance between the sensor and its corresponding on-screen position divided by the controlled moving speed.

Swindells *et al.* measured latency in a VR setup using a phonograph turntable equipped with a tracked marker [17]. The virtual position of the marker was video-projected on the turntable and the whole setup was video recorded. Latency was determined by the ratio of the angular distance between the two markers and the turntable angular speed. The authors also proposed a second method consisting in connecting the RGB wires of the VGA monitor to A/D converters. Latency was measured as the time from when the software renders a color scene until the software detects a voltage change. In a similar way, Mine used an oscilloscope to detect the breaking of a beam of light between a photo diode/LED by the input device (a tracker) mounted on a pendulum [12]. A photo diode connected to the oscilloscope was mounted on the display to measure the time when the display turned from black to white upon detecting the input device crossed the photo diode/LED.

Pavlovych and Stuerzlinger measured mouse to display latency by moving a mouse back and forth along the top bezel of a monitor at a rate of about 1 Hz [14]. Movements of both the mouse and the cursor were recorded with a digital camera and analyzed by measuring the time-frame differences of their corresponding phases of motions. They reported laten-

cies between 33.2 $\pm$ 2.8 ms and 102.9 $\pm$ 3.3 ms, depending on the type of mouse (wired or wireless) and screen (CRT, LCD or DLP projection screen). They found a latency equal to 43.2 $\pm$ 2.7 ms using a wired mouse and an LCD monitor.

In a similar work, Teather *et al.* positioned a mouse in front of the display behind a Styrofoam pendulum [18]. The mouse measured the displacement of the pendulum oscillating at approximately 0.8 Hz and updated a line displayed on the screen, following the pendulum's arm. The manual analysis of the video provided the latency. Using a wired mouse and a 120 Hz CRT monitor, they found an average 35 $\pm$ 2 ms latency. Steed also proposed a method to automatically extract the position of the pendulums, fit sine curves and determine their phase shift to measure the latency [16].

In the context of touch screens, Ng *et al.* measured the latency by video-recording a finger and the associated moving object at 240 Hz. The finger was moved along a ruler at a speed as constant as possible. By analyzing the video, the latency was computed by measuring the distance between the finger and the displayed object, and by estimating the finger speed using the distance traveled between successive frames. They found the latency to be typically between 50 and 200 ms. Using a high speed video camera, microphone and accelerometer Kaaresoja and Brewster measured the visual, audio and tactile latencies of different smart phones [9]. They found latencies from 60 and up to 200 ms, depending on the smartphone and the application. Bérard and Blanch proposed to measure latency on touch surfaces by following a wheel spinning at a constant speed with a finger [4]. Using a Wacom Cintiq display with a 3.4 GHz processor, they found an average latency of 43 ms. Despite its simplicity this method is obviously not applicable to mouse based interaction.

To sum up, methods based on video recording are the most commonly used to measure latency. But they introduce problems and biases related to the frame rate (each measure cannot be lower than the capture period), spatial resolution (measured distances cannot be below the camera pixel accuracy, and high frame rate camera typically have low resolutions) and blur (fast moving objects get blurred, and since high frame rate cameras require good light exposure for the image to be sharp, this can also wash out the image from the monitor). Good measures require perfect alignment between the camera, physical device and screen and good lighting conditions. In addition videos generally require to be processed by hand, which is a time consuming and error prone process. Finally, these methods help at measuring end-to-end latency but do not allow to insert probes at different levels of the system in order to finely determine where the latency comes from.

## MEASURING LATENCY USING AN OPTICAL MOUSE

A modern computer mouse consists of a small camera comprising a pixel array between typically 18x18[1] and 30x30[2] pixels, a lens and an LED or laser to illuminate the surface. The microcontroller processes the images at a high framerate, typically between 1,500 Hz and 12,000 Hz, to detect visible features in the image and measure their translation between consecutive images. The measured displacements are reported to the computer as counts. The sensor has a relative responsivity above 80% typically for wavelengths between 500 nm (green) and 950 nm (infra-red)[1,2]. The camera adjusts its shutter to keep the brightest pixel at a given value. The lens determines the dimensions of the area under the mouse recorded by the camera. Laser mice typically record a smaller area to see finer details.

Our method for measuring end-to-end latency consists in positioning a standard computer mouse at a fixed location on an horizontally oriented monitor. Alternatively, one can use adhesive (*e.g.* Blu-Tack) to stick the mouse on a vertical monitor but should take care not to increase the distance between the sensor and the screen too much in order to avoid blurring the captured image. Depending on the shininess of the screen surface, the LED of the mouse has to be obscured with *e.g.* black tape, if the mouse cursor hardly or does not move when moving the mouse on the monitor displaying a white background. Once the mouse is correctly set up on the display, a given texture is displayed on the screen and moved of a controlled distance (*e.g.* 1px), while a timestamp is recorded. The texture displacement is intended to create a well-controlled closed-loop by producing a fake mouse displacement at a given and measurable time, which will thus be detected and reported by the mouse sensor to the system as a normal event. Upon reception of this event and subsequent ones (drivers, system pointer, toolkit, etc.), successive timestamps are recorded and latency can be measured at several levels.

### Finding the right texture to display

The main assumption of our method is that the texture we display and its translation will produce a displacement that (i) can be detected by the mouse sensor; and (ii) can subsequently produce a system cursor displacement.

Mice resolution is reported in CPI (counts per inch) which corresponds to the smallest displacement they can report. It starts at 400 CPI and can go up to 12,000 CPI[3]. Screens resolution, in PPI (pixels per inch), is far lower, around 100 PPI for current average-end LCD monitors, while high-resolution monitors can go up to 300-400 PPI. This means that a translation of a single pixel on the highest-end monitor is larger than what can measure the lowest-end mouse. As a result, moving a texture of one pixel ensures the displacement can be detected by the mouse. It is also the best case scenario to make sure that a part of the texture remains visible by the mouse sensor after it is translated: the mouse needs to see common features between two images to compute a displacement.

Moving the texture of one pixel is likely to produce mouse reports as low as 1 count. Thus, we have to make sure that 1 count produces a displacement of the system pointer of at least 1 pixel after the operating system transfer function is applied. Casiez and Roussel have shown that each operating

[1] http://www.alldatasheet.com/datasheet-pdf/pdf/520966/AVAGO/ADNS-2610.html

[2] http://www.pixart.com.tw/upload/ADNS-9800%20DS_S_V1.0_20130514144352.pdf

[3] http://gaming.logitech.com/en-us/product/g502-proteus-core-tunable-gaming-mouse

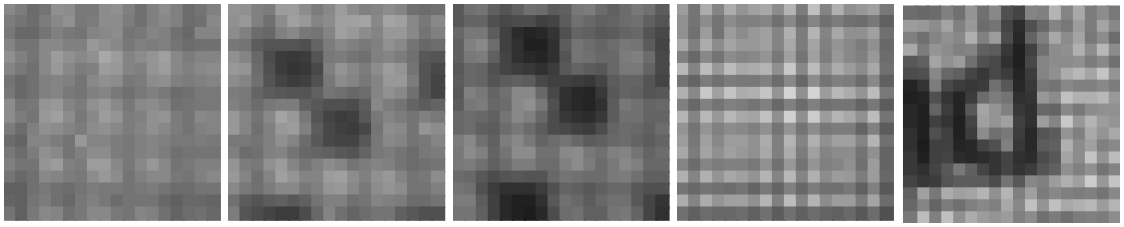

**Figure 2: Raw images captured using an Avago ADNS-2610 sensor. From left to right: white background on a 86 PPI Dell 1905FP; black dots on a white background with no LED occlusion (same display); same pattern with LED occluded using black tape (same display); white background on a 148 PPI Apple Retina display; black letter 'd' on a white background captured on the same display. Note that pixels from the sensor range between 0 and 63. The above images are displayed using re-scaled values between 0 and 255 for better visibility.**

system uses custom transfer functions, but their reverse engineering helps determining the appropriate mouse configuration panel settings [5]. On Windows XP, 7, 8 and 10, setting the slider to one of the 2 last positions when the "enhanced pointer precision" box is checked results in at least a 1 cursor pixel displacement for every 1 count. When "enhanced pointer precision" is unchecked, any position higher than the middle one works. On Linux, any configuration of the settings in the mouse configuration panel works. On OS X, "acceleration" needs to be disabled[4] because all settings for non-linear transfer functions prevent to systematically get at least one pixel displacement for one count.

We first tried to display and move different textures designed on the rule of the thumb (grids, images, Perlin noise, random noise ...) but only some of those sometimes produced a mouse event. Obviously we needed a less arbitrary and more systematic approach.

*What the mouse sees*

To help finding the right texture to display, we first needed to get the raw image from a mouse sensor in order to better understand "what the mouse is seeing". We opened a 400 CPI Logitech M-BT58 mouse and wired its Avago ADNS-2610[1] sensor to an Arduino board. We chose this mouse as we could easily find its datasheet, it is widely available and it corresponds to the lowest mouse resolution available (our worse case scenario). We set the sensor in pixel grab mode, put the mouse on a LCD screen displaying white background and black text (without anti-aliasing), and sent the $18\times18$ captured pixels to a custom made application. This first helped us to figure out the physical dimensions of the area recorded by the sensor, which is around $1\times1$mm. On a 100 PPI screen, this corresponds to about $3\times3$ visible pixels, giving us the size of our unit pattern for the texture. Figure 2 shows images captured using the sensor.

We noticed on the captured images that the screen's pixel grid and the corresponding small physical separation between its pixels are visible (Fig. 2), which is known as the "*screen door effect*" [2]. This explains why moving a mouse with the LED obscured on a white on-screen image is still producing pointer movements: the sensor detects the fixed pixel grid which is helpful to compute the mouse displacements (this is how the very first optical mice were working, with a grid printed on a dedicated pad). As we will see below, the screen door effect proved to be an important factor for reliable texture displacement detection.

*What we decided to show*

To systematically investigate the textures that could fake a mouse displacement, we built $512\times512$ pixels textures based on the repetition of all the possible $3\times3$ patterns composed of black and white pixels (we discarded gray levels as they reduce the contrast of the images and reduce the chance of finding relevant patterns). The mouse was roughly aligned with the screen edges, so the pixels of the sensor are approximately aligned with the screen ones. We tested the resulting $2^9$ patterns using 9 offset positions and 15 repetitions. The offset position corresponds to a translation of the starting position of a texture, between 0 and 2 pixels in x and y. This ensured invariance with respect to the position of the mouse sensor on the screen, avoiding to get a texture working just by chance because the sensor would be well aligned with it.

We developed a testbed application in C++ using the Qt 5 framework. Textures were pre-computed and stored as PNG images. A trial consisted in displaying a texture with no anti-aliasing – making sure that one pixel of the texture was corresponding to one pixel on screen – and moving it one pixel along the x screen axis. Trials with no mouse event received within 200 ms were marked as failed.

We ran this experiment with several mice and monitors. After that we visually compared the patterns with the higher success rate for each pair of mouse and monitor to select the most robust ones over all the conditions. The patterns that work best are those that cancel the screen door effect: if the physical gaps between pixels are visible, the mouse sensor is likely to detect them as visual features that do not move or disappear between two captured frames, while other features in the texture move. In such situations, the mouse is not able to compute a displacement. As a result, two adjacent pixels in a texture cannot be white, and working textures are those made of oblique and non adjacent lines of pixels (*e.g.* a repetition of the pattern displayed in Figure 3).

**Evaluating the texture in different conditions**

We evaluated the texture presented in Figure 3 with 10 different mice and with 2 monitors using the previously described application on a laptop MacBook Pro Retina 15-inch (Mid 2014) 2.5 GHz, 16GB Ram, SSD hard drive, $1920\times1200$

[4]Using the command `defaults write .GlobalPreferences com.apple.mouse.scaling -1` and logging out and back in.

| | Dell 1905fp (86 PPI) | | | Apple Retina (148 PPI) | | |
|---|---|---|---|---|---|---|
| | SR | mean | std | SR | mean | std |
| Apple A1152 (Agilent ADNS-2051) | 83.2 % | 62.0 ms | 14.1 ms | 98.8 % | 64.1 ms | 6.0 ms |
| Dell M-UVDEL1 (Agilent S2599) | 18.0 % | 62.8 ms | 20.1 ms | 0.0 % | - | - |
| Dell MS111-L (unknown) | 0.0 % | - | - | 0.0 % | - | - |
| IBM MO09BO (Agilent H2000) | 17.3 % | 71.8 ms | 26.2 ms | 64.4 % | 75.9 ms | 5.5 ms |
| Logitech M-BT58 (Avago ADNS-2610) | 75.7 % | 62.9 ms | 16.5 ms | 99.9 % | 68.4 ms | 5.8 ms |
| Logitech M-U0017 (unknown) | 71.3 % | 71.6 ms | 9.8 ms | 41.1 % | 76.9 ms | 8.2 ms |
| Logitech M-UV96 (Agilent S2599) | 16.7 % | 63.5 ms | 16.9 ms | 98.7 % | 61.8 ms | 5.7 ms |
| Logitech M100 (unknown) | 0.0 % | - | - | 16.0 % | 71.3 ms | 9.0 ms |
| Logitech MX310 (Agilent S2020) | 99.6 % | 55.1 ms | 7.0 ms | 99.9 % | 65.8 ms | 5.1 ms |
| Kensington Ci65 Wireless (unknown) | 94.1 % | 70.0 ms | 7.5 ms | 87.7 % | 81.6 ms | 6.6 ms |

**Table 1: Success rate (SR), mean latency and standard deviation for 10 mice with 2 monitors on a MacBook Pro using OS X 10.10. When known the sensor reference is provided in brackets. Latency is measured as the time elapsed between the one pixel horizontal texture translation to the reception of the first mouse move event in the Qt application.**

pixels resolution (148 PPI[5]), using OS X 10.10. The monitors were the laptop screen and an external Dell 1905fp connected through DVI. There were few applications running but we could not control the background processes from running. Each pair of mouse/display was evaluated 1000 times. A trial consisted in translating the texture one pixel along the x abscissa. Trials alternated translation from left to right and right to left. We measured the success rate (SR) to produce mouse events, the mean and standard deviation (std) for latency, measured from the texture translation to the first mouse move event in Qt. We noticed that the mouse does not always send a single report for one pixel texture translation. We hypothesis this can be due to the high frame rate of the mouse sensor measuring some transition when pixels change color. Trials with no mouse event produced within 200 ms were marked as errors. Mice were always positioned at the center of the screen.

Results are presented in table 1. On average the texture works better on the higher resolution monitor, probably because the mouse sensor can see finer details. Except two mice that do not work at all on each monitor, the other mice show a success rate above 16%, which means it is at least possible to perform one measure of latency per second using these mice, using a 200 ms timeout. Two mice for the lower end monitor and four mice for the higher resolution monitor have a success rate above 95%, which means it is possible to have up to 5 measures of latency per second with them. On average the latency is close to 70 ms with a higher standard deviation for the external monitor.

#### Measuring latency at different positions on a display

We measured end-to-end latency at different positions on an Apple Retina display using a Logitech MX310. We found a mean of 58.9 ms (std=8.4 ms) for top right position, 65.6 ms (std=9.1 ms) for top left, 68.2 ms (std=7.8 ms) for center, 77.7 ms (std=8.0 ms) for bottom right and 78.8 ms (std=7.6 ms). Note that all positions were not measured simultaneously.

[5]We set the monitor to the maximum available resolution in the configuration panel (1920×1200 pixels) and the physical dimensions of the display are 330.63 × 207.347 mm.

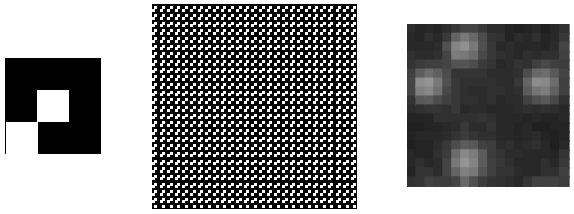

**Figure 3: The displayed texture (middle) is composed of a repetition of the 3×3 pixels pattern composed of black and white pixels (left). Image of the texture captured by the ADNS-2610 mouse sensor (right) showing the screen door effect is canceled.**

Considering that LCD monitors usually update the screen from top to bottom, it makes sense to get lower values for the top of the screen. If we subtract the mean latency at the bottom of the screen and at the top of the screen, we get 16 ms which is close to the time screen time period (1000/60 = 16.67 ms). For all the following measures, the computer mouse was positioned at the center of the screen.

## INSERTING PROBES IN THE SYSTEM

Conversely to other lag measurement methods that rely on external devices, our probe is a mouse connected to the computer on which we want to measure lag. Our method thus "closes the loop" and enables to establish time measurements based on a single (internal) clock. We use this property to insert probes at different places in the system to investigate the sources of latency: since all time measurements are based on the same clock, comparing these times is reliable, while it would require substantial supplemental work with external systems such as a video camera.

Our probes consist of callback functions registered at different places. Each callback logs a timestamp and information about the event. We start measuring time before the texture is moved (*Window repaint* in Fig. 4). The texture is then updated on-screen after a given amount of time (*on-screen texture moved*). Upon detecting a displacement, the mouse sends an HID report (*HID report sent*) that is handled by the low-level layers of the operating system before getting out of the HID stack (*HID report received*). The operating system then

processes the event and applies the transfer function to move the system pointer (*System pointer moved*). Upon notification of this pointer movement, the toolkit creates an event (*MouseMove event created*) and dispatches it to the appropriate widget (*MouseMove event dispatched*). The end-to-end latency we measure is the time elapsed between *Window repaint* and *MouseMove event dispatched*. Three probes inserted at steps 1, 2 & 3 (Fig. 4) allow us to further characterize this latency.

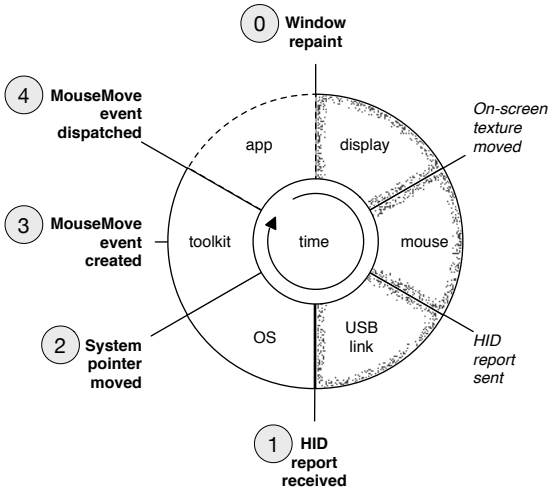


**Figure 4: Conceptual pipeline between a (simulated) physical movement of the mouse and the notification of this movement to the appropriate on-screen widget. Our system is able to measure end-to-end latency by inserting probes at different software locations, numbered from 1 to 4.**

### Comparison of different toolkits

We used libpointing[6] [5] to get notifications of HID reports reception and platform-specific code (Quartz event taps) for notifications of system pointer updates on OS X. We implemented these probes along with toolkit-specific code for *Mouse Move event* creation and dispatching using C++ / GLUT, C++ / Qt and Java / Swing on the Apple MacBook Pro laptop previously described. All probes were implemented as asynchronous and non-blocking callbacks. All toolkits used double buffering for the display. Timestamps were measured with sub-millisecond precision with `GetSystemTimeAsFileTime` on Windows 7 and `gettimeofday` on Linux and OS X.

We compared the measurements obtained on 1000 trials with our three implementations using a Logitech MX310 mouse. Table 2 shows the success rate (percentage of texture displacements that resulted in a valid *MouseMove* event, i.e. one that matches the sequence of observations illustrated by Figure 4), mean end-to-end latency (time between *repaint* and the first dispatched *MouseMove* event) and the corresponding standard deviation. Results show substantial differences between GLUT and Swing or Qt in terms of mean latency but comparable standard deviations. Note that in the three cases, the latency introduced by the movement of the system pointer and the toolkit is below 2 ms. Figure 5 shows the distribution of lag measurements between *repaint* and the first *MouseMove*. This plot shows a clear difference between Java / Swing and C++ / Qt despite comparable mean values, for which we are unable to provide a definite explanation. The differences illustrated by Table 2 and Figure 5 clearly require further investigation.

[6]http://libpointing.org

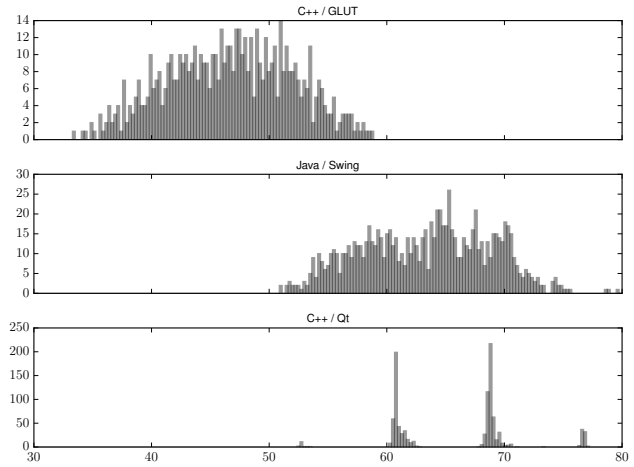


**Figure 5: Distributions of the time (in ms) elapsed between *Window repaint* and the first *MouseMove event dispatched* using the Logitech MX310 and different toolkits on a MacBook Pro.**

### Comparison with other systems

We measured end-to-end latency on Linux Ubuntu 14.04, Windows 7 Pro and web browsers, all on the same machine, an Asus N53JQ (i7Q740@1.73GHz, 4Gio DDR3@1333MHz, Nvidia Geforce GT425M 1Gio, 250 Gio SSD). We used the previously described Dell 1905fp monitor and the Logitech MX310 mouse. End-to-end latency was measured using our Qt 5 application. For web browsers we used an HTML5 / Javascript version performing the same measures. Latency was measured from texture translation to the mouse move event received by the toolkit.

| | Application | mean | std |
|---|---|---|---|
| Ubuntu 14.04 | Qt 5 app | 50.9 ms | 7.6 ms |
| | Chrome 41 | 71.3 ms | 5.7 ms |
| | Firefox 35 | 65.7 ms | 6.6 ms |
| Windows 7 | Qt 5 app | 74.9 ms | 9.2 ms |
| | Chrome 41 | 62.2 ms | 8.5 ms |
| | Firefox 37 | 83.0 ms | 9.7 ms |

**Table 3: End-to-end latency on different operating systems and web browsers measured on the same computer (Asus N53JQ with a Logitech MX310 mouse and a Dell 1905fp monitor).**

On Linux Ubuntu 14.04, latency is comparable with OS X 10.10, although measured on a different machine. The web browsers add between 15-20 ms latency compared to Qt 5 while jitter remains similar (Table 3). Results on Windows 7 are more surprising: latency on Chrome 41 is lower than the one found on Qt 5. We repeated the measures several times and we kept finding the same results.

| | C++ / GLUT | | | Java / Swing | | | C++ / Qt | | |
|---|---|---|---|---|---|---|---|---|---|
| | SR | mean | std | SR | mean | std | SR | mean | std |
| HID | 100.0 % | 46.2 ms | 5.3 ms | 97.9 % | 62.0 ms | 5.5 ms | 99.9 % | 65.4 ms | 5.1 ms |
| sysPointer | 100.0 % | 46.5 ms | 5.3 ms | 97.9 % | - | - | 99.9 % | 65.7 ms | 5.1 ms |
| tkEvent | 100.0 % | - | - | 97.9 % | 63.2 ms | 5.5 ms | 99.9 % | 65.9 ms | 5.1 ms |
| tkMouseMove | 100.0 % | 46.7 ms | 5.3 ms | 97.9 % | 63.3 ms | 5.5 ms | 99.9 % | 66.0 ms | 5.1 ms |

**Table 2: Comparison between GLUT, Swing and Qt implementations with four different probes on a Macbook Pro with a Logitech MX310 mouse.**

| | C++ / Qt | | Java / Swing | |
|---|---|---|---|---|
| | mean | std | mean | std |
| $< 5$ % | 54.2 ms | 6.7 ms | 51.3 ms | 6.7 ms |
| 25 % | 71.9 ms | 14.6 ms | 59.7 ms | 10.1 ms |
| 50 % | 70.1 ms | 15.1 ms | 71.6 ms | 23.5 ms |
| 75 % | 77.3 ms | 21.5 ms | 71.0 ms | 19.1 ms |
| 100 % | 83.6 ms | 25.8 ms | 75.6 ms | 25.6 ms |

**Table 4: Influence of system load on end-to-end latency on Linux Ubuntu, measured on the same computer (Asus N53JQ with a Logitech MX310 mouse and a Dell 1905fp monitor). Comparison between C++/Qt and Java/Swing.**

### Influence of system load

We investigated how the system load can affect the latency. We simulated different loads on Ubuntu 14.04 using the *stress* utility [3] to control the load, with the command `stress --cpu CPU --io IO --vm VM --timeout 300s --verbose`. The load was simulated with an increasing number of CPU, IO and memory allocation processes. For 25% load, we used CPU=2, IO=5, VM=1; CPU=4, IO=10, VM=2 for 50% load; CPU=6, IO=15, VM=4 for 75% load and CPU=8, IO=20, VM=8 for 100% load. We used the Asus N53JQ computer, Logitech MX310 mouse and Dell 1905fp monitor. We performed 1000 measured using the C++/Qt and Java/Swing toolkits. Results are summarized in Table 4. We can observe that both latency and latency jitter increase with system load. With both toolkits, latency increased by 25-30 ms and jitter by 20 ms between a low system load and a 100% system load.

## DISCUSSION

The methodology we followed helped us determine a pattern and a texture that work with most optical mice (8 of 10 mice we tested). Understanding exactly why it works with some mice and not others is difficult: mouse sensors datasheets are often unavailable and the algorithms for computing mice displacements are never disclosed. Given that optical mice are widely available, it is however easy to use another mouse if one is not working. We could not make any laser mouse work with our method. A first reason can be that laser mice see primarily in the infra-red but they could also see in the visible spectrum according to some datasheets we found[2]. A second reason can be that the surface seen by the sensor is smaller for laser mice than optical ones: laser mice are intended to see finer details on surfaces not visible by optical ones. However, we hypothesize that optical and laser mice would have the same latency as their sensors are similar. The main difference is that the red LED is replaced by an infra-red laser.

The results we found are consistent: measures obtained in similar conditions give similar results. For example we obtain a 50.9 ms mean latency for Linux Ubuntu 14.04 / Qt 5 using the Logitech MX310 and Dell 1905fp monitor on an Asus computer (Table 3) and 54.2 ms for the same system with less than 5% system load (Table 4). With the same mouse and monitor we obtain 55.2 on a Macbook Pro running OS X (Table 1). These results are similar to the only comparable result available in the literature that reported a 43.2 ms end-to-end latency for a wired mouse with an LCD monitor [14]. In addition, when measuring latency at different positions on a screen, we found lower latencies for the top of the display and a difference between its bottom and top of around 16 ms, which is consistent with the way LCD monitors are refreshed.

The different measures we performed indicate that latency and latency jitter are similar on C++/Qt and Java/Swing, even with different system loads. We expected that the Java virtual machine would add some latency, but it is not the case. Our GLUT implementation not only shows a lower latency but also a different distribution of latency values from our Qt implementation, which will require further investigation. Linux and OS X show similar latencies around 50-55 ms with the external Dell monitor and MX310 mouse. Windows 7 in contrast appears to show higher latency (75 ms). Web browsers increase latency by 15-20 ms on average. System load increases latency as well of about 25-30 ms with 100% load. In addition we have shown that latency jitter increases with system load while it remains otherwise below 10 ms. Note that we performed these measures using the mouse having one of the lowest latencies (Table 1). We measured latency from the texture displacement to the mouse move event received by the toolkit. Real usages would have to include the latency introduced by the application-specific code running upon reception of mouse events.

As discussed above, our method allows the insertion of probes at different places in the system to provide reliable time comparison. We observed that the time from when the event exits the HID stack until it enters the mouse move callback is very low, below 2 ms. We expected it would be higher considering the different application layers, computations to determine the widget beneath the mouse pointer and the queues to go through. It means that most of the latency we measured is caused by the display, mouse and low level layers of the operating system. A 60 Hz display introduces a latency of at most 16.67 ms (excluding the display's response time, which probably adds about 10 ms). A computer mouse reports up to 125 Hz which adds another 8 ms on average. This accounts for about 30 ms. Further investigations are re-

quired to clarify where the other 15-30 ms come from. The literature has shown that performance degrades above 50 ms latency and we have shown we are most of the time above this value. This means it is important to report latency in HCI experiments and that further work is needed to reduce latency or to compensate it using appropriate techniques.

Our measures can be affected by both the function used to query time and the resolution of the timers used by the application, the toolkit or the system. Windows, OS X and Linux functions to query time have sub-millisecond precision. On Windows the default timer resolution is of 15.6 ms while it is 1 ms on Linux [1] and OS X. In addition modern versions of Windows, Linux and OS X all use timer coalescing techniques to reduce CPU power consumption. This certainly contributes to some variability in the measured end-to-end latency, but it can not be easily avoided. Overall, this is just another source of (small) variability found in "real life" conditions. An advantage of our method over camera-based ones is that it makes it possible to easily and quickly perform many measures to estimate this variability.

## CONCLUSION

We introduced a method that makes latency measure of desktop interfaces simpler. This method relies on cheap hardware already in the hands of every users and a particular texture to display, which can thus be easily replicated. We have shown that our method works with most optical mice and provides real time measures of latency. The measures we performed can guide interface designers and HCI researcher to choose appropriate toolkits, operating systems and hardware. More generally we believe that our simple method for measuring latency has the potential to change the way HCI researchers run experiments and the way practitioners tune their interactive systems. An on-line interactive demo and additional material are available at http://ns.inria.fr/mjolnir/lagmeter/.

## ACKNOWLEDGMENTS

This work was supported by ANR (TurboTouch, ANR-14-CE24-0009). We thank Daniel Vogel for the video voice over.

## REFERENCES

1. Linux high resolution timers. Retrieved July 7, 2015 from http://elinux.org/High_Resolution_Timers.
2. Screen-door effect description. Retrieved July 7, 2015 from http://en.wikipedia.org/wiki/Screen-door_effect.
3. Stress workload generator manual page. Retrieved July 7, 2015 from http://people.seas.harvard.edu/~apw/stress/.
4. Bérard, F., and Blanch, R. Two touch system latency estimators: High accuracy and low overhead. In *Proceedings of ITS '13*, ACM (2013), 241–250.
5. Casiez, G., and Roussel, N. No more bricolage! methods and tools to characterize, replicate and compare pointing transfer functions. In *Proceedings of UIST '11*, ACM (2011), 603–614.
6. Deber, J., Jota, R., Forlines, C., and Wigdor, D. How much faster is fast enough? user perception of latency & latency improvements in direct and indirect touch. In *Proceedings of CHI '15*, ACM (2015), 1827–1836.
7. Ellis, S. R., Young, M. J., Adelstein, B. D., and Ehrlich, S. M. Discrimination of changes of latency during voluntary hand movement of virtual objects. In *Proceedings of HFE '99* (1999), 1182–1186.
8. Jota, R., Ng, A., Dietz, P., and Wigdor, D. How fast is fast enough? a study of the effects of latency in direct-touch pointing tasks. In *Proceedings of CHI '13*, ACM (2013), 2291–2300.
9. Kaaresoja, T., and Brewster, S. Feedback is... late: measuring multimodal delays in mobile device touchscreen interaction. In *Proceedings of ICMI-MLMI '10*, ACM (2010), 2:1–2:8.
10. Liang, J., Shaw, C., and Green, M. On temporal-spatial realism in the virtual reality environment. In *Proceedings of UIST '91*, ACM (1991), 19–25.
11. MacKenzie, I. S., and Ware, C. Lag as a determinant of human performance in interactive systems. In *Proceedings of CHI '93*, ACM (1993), 488–493.
12. Mine, M. Characterization of end-to-end delays in head-mounted display systems. Tech. rep., University of North Carolina at Chapel Hill, 1993.
13. Ng, A., Lepinski, J., Wigdor, D., Sanders, S., and Dietz, P. Designing for low-latency direct-touch input. In *Proceedings of UIST '12*, ACM (2012), 453–464.
14. Pavlovych, A., and Stuerzlinger, W. The tradeoff between spatial jitter and latency in pointing tasks. In *Proceedings of EICS '09*, ACM (2009), 187–196.
15. Pavlovych, A., and Stuerzlinger, W. Target following performance in the presence of latency, jitter, and signal dropouts. In *Proceedings of GI '11*, Canadian Human-Computer Communications Society (2011), 33–40.
16. Steed, A. A simple method for estimating the latency of interactive, real-time graphics simulations. In *Proceedings of VRST '08*, ACM (2008), 123–129.
17. Swindells, C., Dill, J. C., and Booth, K. S. System lag tests for augmented and virtual environments. In *Proceedings of UIST '00*, ACM (2000), 161–170.
18. Teather, R. J., Pavlovych, A., Stuerzlinger, W., and MacKenzie, I. S. Effects of tracking technology, latency, and spatial jitter on object movement. In *Proceedings of 3DUI '09*, IEEE (2009), 43–50.
19. Ware, C., and Balakrishnan, R. Reaching for objects in VR displays: Lag and frame rate. *ACM ToCHI 1*, 4 (Dec. 1994), 331–356.