Giving a Hand to the Eyes: Leveraging Input Accuracy for Subpixel Interaction

Nicolas Roussel[1], **Géry Casiez**[1,2,3], **Jonathan Aceituno**[1] **and Daniel Vogel**[4]
[1]Inria Lille & [2]LIFL, [3]University of Lille, France
[4]Cheriton School of Computer Science, University of Waterloo, Canada
nicolas.roussel@inria.fr, gery.casiez@lifl.fr, jonathan.aceituno@inria.fr, dvogel@uwaterloo.ca

ABSTRACT
We argue that the current practice of using integer positions for pointing events artificially constrains human precision capabilities. The high sensitivity of current input devices can be harnessed to enable precise direct manipulation "in between" pixels, called subpixel interaction. We provide detailed analysis of subpixel theory and implementation, including the critical component of revised control-display gain transfer functions. A prototype implementation is described with several illustrative examples. Guidelines for subpixel domain applicability are provided and an overview of required changes to operating systems and graphical user interface frameworks are discussed.

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

General terms: Design, Human Factors

Keywords: Display density; indirect pointing; input device sensitivity; device's human resolution; subpixel interaction; direct manipulation

INTRODUCTION
Over the last thirty years, there have been tremendous increases in computer processing power, storage capacity, and network bandwidth. Graphical user interfaces (GUIs) have played a crucial part in making these resources available, enabling the direct manipulation of data, which has also increased substantially in diversity and size. However, while processing, storage, and communication capabilities have experienced a hundred, thousand, or million-fold increase to handle these increased data manipulation requirements, the situation is quite different for computer displays. Most modern displays typically have a pixel density lower than 150 PPI (pixels per inch) and cannot display more than 2.5 megapixels[1]. Display capabilities have increased by only a factor of 21 compared to the original Macintosh.

[1]see `http://libpointing.org/resolution/` for information on the sensitivity and pixel density of commercial mice and displays

UIST'12, October 7–10, 2012, Cambridge, MA, USA.

When data density exceeds display density, direct manipulation becomes a problem. For example, selecting an item from an overview of a large discrete set, or fine adjustment of a continuous variable. Solutions usually involve a scale adjustment in visual space or layering transfer functions to reach the desired control precision. But these approaches take for granted the curious way in which operating systems map input device movements to data.

When you move a pointing device, motion deltas (in device-specific units) are sent to the I/O subsystem and transformed by a transfer function [5] into on-screen pointer motions (in pixels). The system then moves the pointer accordingly and generates *movement events* which are routed via the window subsystem and GUI framework to a widget to manipulate data (Figure 1a). What is curious is that although these are commonly called "mouse events," they are actually "pointer events" because the information they carry describe on-screen *pointer movements*, not *mouse movements*. So, a pointing device is not really how we interact with data: it is a device through which we interact with an on-screen pointer, through which we interact with data. Thirty years ago, mouse sensitivity and display density were comparable, so this kind of input mapping seemed reasonable. The original Macintosh mouse was 90 CPI (counts per inch), quite close to its ≈72 PPI display density. The problem is that this mapping still serves as the basis for all graphical interactions,

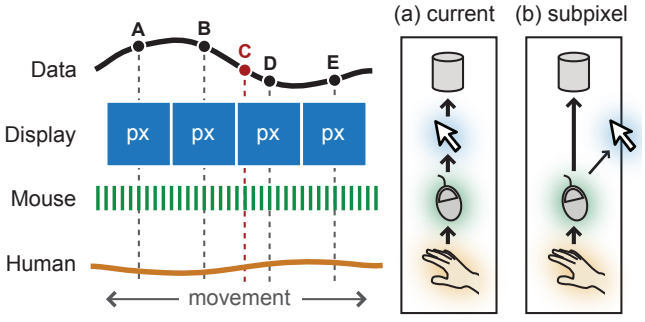


Figure 1: Input mappings: (a) currently, human movements are discretized by mouse sensitivity, then again by display density: data points "in between" pixels like 'C' are unreachable; (b) a subpixel mapping discretizes human movements by mouse sensitivity only, for precise data manipulation.

and since movements are measured in pixels, they are only as accurate as the display density.

Fifteen years ago, in his classic paper "*The Eyes Have It (...)*" [12], Shneiderman pointed out how our remarkable perceptual abilities were often underutilized. From what we just described, it is fair to say that our remarkable motor abilities are currently underutilized. To put it simply, the eyes could surely use a hand. The sensitivity of modern mice is now typically over 800 CPI and can reach more than 10000 CPI, and some touchpads are 1000 CPI — levels of precision more suitable for capturing high human sensitivity [4]. It is time the current input mapping is revised.

We are proposing *subpixel interaction*, a subtle, but critical alteration to current system architectures where device movements are mapped directly to data, enabling interaction "in between pixels" (Figure 1b). Our solution requires a small software-level modification, yet increases the accuracy of standard interaction techniques and remains compatible with complimentary approaches like Focus+Context lenses. After describing motivating situations where pixel-precision is inadequate and reviewing related work, we describe the details of our solution. Specifically, guidelines for useful subpixel resolution and, since our focus is on indirect pointing devices like mice and touchpads, we describe revised transfer function characteristics which leverage subpixel capability. Our work is a rallying cry with implementation details to leverage human input capability for precise direct manipulation.

WHY PIXELS ARE NOT ENOUGH

To illustrate the current problem more concretely, consider navigating a video player where frame selection accuracy is ultimately limited by the pixel width of the display. For example, the Apple OS X QuickTime player has a fixed timeline slider width of 315 pixels (Figure 2a). Moving the slider 1 pixel skips a 5 minute video by 1 second, but a 1.5 hour movie is skipped by 17 seconds (Table 1). This may be fine for coarse positioning, but tasks like skipping *only* commercials becomes tedious or impossible as the video length increases. Even more extreme is accurately selecting specific frames with Apple Keynote's video inspector which uses a 198 pixel, fixed width slider.

There are a myriad of other examples where there are not enough pixels to provide the required precision: picking a particular item in a large list using a slider; image navigation and editing (e.g. accurately cropping a megapixel image, picking a precise location on a map); resizing a calendar event to minute precision (e.g. when booking flight departures or lawyers tracking billing time); high precision vector drawing tasks (e.g. drawing a structural wall to centimeter precision or aligning objects in vertex-dense areas); or manipulating objects in 3D applications (e.g. setting orientation of CAD objects to within 0.1°).

	Duration	Keynote	QT Player
CHI video	05:00	00:01.515	00:00.952
TV Show	52:00	00:15.758	00:09.905
Movie	1:30:00	00:27.273	00:17.143
Gone with the Wind	3:58:00	01:12.121	00:45.333
Bergensbanen	7:14:13	02:11.581	01:22.708

Table 1: Time between pixels in the Keynote inspector and QuickTime Player sliders for different video durations.

A solution to these pixel precision problems could be to simply add more pixels. Returning to the video player example, consider enabling a more reasonable 1 second selection accuracy with a 1.5 hour video by increasing the timeline slider width. Unfortunately, the slider would need to be 5400 pixels wide, the width of three 1080p HDTVs. This is clearly unworkable, so a common way to overcome the limitation is to introduce alternate navigation controls such as arrow keys, but these can be slow and error prone, partly because they no longer follow principles of direct manipulation. Next, we discuss how researchers have approached the pixel precision problem while trying to maintain the benefits of direct manipulation.

RELATED WORK

To make direct manipulation more accurate, previous work introduces explicit *precise pointing modes* such as discrete or continuous zooming and Focus + Context Lenses. In addition to problems inherent with modes, these techniques are impeded by the assumption that whole-pixel input resolution is a hard constraint – they ignore the high resolution capabilities of modern pointing devices and human limb accuracy.

Increasing Pointing Accuracy with Modes

A straightforward way to increase input accuracy is to *zoom* and magnify the desired target area to make pointing easier. For example, the OS X QuickTime Player 10.1 has what is essentially a dwell-while-dragging zoom mode to select video frames more precisely (Figure 2a). Other exam-

Figure 2: Manipulating visual space to increase pointing accuracy: (a) dwelling while dragging the OS X QuickTime Player 10.1 timeline slider *zooms* the timeline in to provide 1 second precision in a 10 second window; (b) some YouTube videos have a Focus+Context timeline with a secondary slider providing 1 second precision.

ples include Popup Vernier [3] which supports explicit incremental cursor space magnification during drag operations, and Ramos and Balakrishnan's Zliding technique [10] which continuously maps pen tip pressure to view scale. These examples are part of the more general class of zoomable user interfaces (ZUIs) [8] where the zoom level is adjusted using an explicit control like dwell, pressure, keyboard keys, mouse scroll wheel, or GUI buttons, sliders, or textfields. However, zooming requires an explicit cognitive decision, locating the optimal zoom level to facilitate precise pointing can be time consuming and difficult, and the overall context of the information space is lost while zoomed in.

To maintain context, Focus+Context Lenses embed a magnified portion of the information space in the overview view [9]. Although this addresses some shortcomings of ZUIs, Appert et al. [2] argue that when the input device is used to simultaneously move the lens *and* point at targets in the lens, there is a *quantization problem* limiting the usable range of lens magnification levels. Their solution is also to introduce an explicit mode switch, this time to switch between positioning the lens and pointing at targets inside the lens. Techniques like Alphaslider [1] and YouTube's secondary magnified timeline slider (Figure 2b) are one-dimensional examples of the Focus+Context strategy. Since they are one-dimensional, the switch to the accurate mode is achieved spatially, by acquiring the magnified slider.

Another solution is to switch to a mode with a custom precise pointing transfer function. This can be achieved with a technique called *pointer lock* where "*input methods of applications [are] based on the movement of the mouse, not just the absolute position of a cursor*" [11]. This is often used for first-person navigation in games, but can also be used for modal precise control of parameters like object rotation in 3D applications. Since direct manipulation no longer follows the system pointer, the pointer is usually hidden or "locked" in place to reduce user confusion.

Unfortunately, using pointer lock to enable a modeless subpixel solution is not practical. For the second transfer function to increase accuracy, it must scale down the relatively large integral movement deltas received from the system event (Figure 3). This will increase motor space which lowers comfort and performance due to device clutching. In addition, since precise pointing is necessarily mapped to low speeds which have a lower-bound constrained by integral deltas, then larger 'normal' pointing movements will be harder to control since the speed-to-position mapping is condensed. Finally, since the precise transfer function is layered, it is has to compensate for different hardware, software, and user settings which determine the underlying system transfer function [5]. Like the other techniques above, using pointer lock does not harness the potential precision of modern input devices.

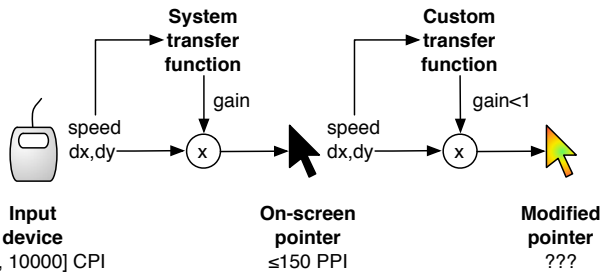


Figure 3: Using pointer lock for subpixel interaction amounts to layering a second precise transfer function in which low speeds are constrained by integral movement deltas.

Lost Capabilities of Input Devices and Human limbs

Traditional GUI toolkits like Java Swing report pointer movement events as integers. Even toolkits like Qt or WPF which are designed for display resolution independence[2] report motion events as integers, regardless whether they are specified as floating point (with WPF, this is due to integral `WM_MOUSEMOVE` messages). On OS X, position events are floating point, but these only hold non-integral values with absolute pointing devices such as tablets. For relative devices, Casiez and Roussel [5] show that pointing transfer functions used by Microsoft Windows, OS X, and X.Org apply a speed-dependent float factor on motion deltas but move the pointer according to the sole integral part of the result. Although the fractional part is preserved internally for later accumulation, it is inaccessible from high-level software.

Raw motion deltas can usually be obtained (e.g. through `WM_INPUT` messages) and are commonly used by games for custom high-sensitivity pointer control. But until recently [5], the lack of appropriate knowledge and tools made it quite difficult for this custom pointer to behave like the system one for "normal" pointing tasks. The idea of using floating coordinates to improve the precision of the system pointer with relative devices was once suggested within the X.Org community[3], but received very little attention.

The sensitivity of modern pointing devices ranges from 400 up to 10000 CPI or more for high-end mice, enabling minimal measurable displacements of 0.0635 mm and 0.00254 mm respectively. Whether people can leverage this level of precision depends on their ability to control fine movements of input devices in motor space. In a multi-scale pointing experiment, Guiard et al. [7] found that users can comfortably acquire 0.06 mm targets in motor space (423 CPI), but this is not an upper bound since it was the smallest width they evaluated. More recently, Bérard et al. [4] defined a *Device's Human Resolution* (DHR) as "*the smallest target size that users can acquire with an ordinary amount of effort using a particular device*". They found that the DHR depends on the input device and the user's sensorimotor capabilities. For computer mice they found DHR values from 700 CPI to 1400 CPI. This means that humans can easily exploit input resolutions up to 7 to 14 times higher than a typical 100 PPI display. The challenge is how to translate these human capabilities into effective control of direct manipulation interfaces.

[2] i.e. where graphical objects are positioned using floating point coordinates in a space independent of the screen

[3] `http://johan.kiviniemi.name/blag/making-x-report-the-mouse-position-with-subpixel-precision/`

LEVERAGING INPUT ACCURACY

Direct manipulation interfaces translate objects of interest into physically manipulatable, graphical representations. Each object is composed of a collection of underlying data, often referred to as a model. Models can represent discrete elements (e.g. integers, video frames) and continuous elements (e.g. floating values). The range of elements may be bounded (e.g. choosing a video frame from a clip) or unbounded (e.g. specifying a CAD object's length). In practice, unbounded models can be considered bounded given reasonable minimum and maximum values for the task, and continuous models may be considered discrete given a reasonable level of precision for the task. By treating all models as discrete, we can define the model's cardinality (N) as the number of elements which users expect to select from.

When display density prevents the object from being represented at the desired granularity (i.e. with the right level of detail), auxiliary representations are needed such as text labels, zoomed views, or even sounds or haptic effects. This solves the output granularity problem, but the problem remains for input. This is what motivates us: we want to overcome the problem of input granularity being limited by display density; we want to support physical actions on objects with a higher precision than the display. As discussed earlier, there is a hidden potential in input devices that is compatible with the degree of control that people can exert. We would like to take advantage of this potential in a way that smoothly integrates with current practices.

In this section we develop a way to provide subpixel input while maintaining current graphical object representations and without introducing any explicit "high precision" mode. We preserve the on-screen pointer since it provides essential visual feedback for direct manipulation. We alter the mapping between movements in motor space and visual space, but we do this without changing the current "feeling" of pointing: we avoid situations where the cursor feels stuck and then suddenly moves quickly, and we do not extend the device operating space. People continue to interact as they do now, but with higher, subpixel precision when needed.

Taking the input device into account

To leverage input accuracy, the system first needs to know the device sensitivity, expressed in device-independent units (such as CPI). This is important so that the system understands what it measures: a higher sensitivity should result in a more precise movement, not a change of magnitude. Unfortunately devices and hardware drivers typically do not provide this information (and probably for that reason, the Windows and X.Org pointing transfer functions completely ignore it [5]).

As already explained, pointing transfer functions apply a dynamic gain factor on device displacements, move the pointer according to the integral part of the result, and store the fractional part internally. We want that information all the way up: the fractional part should be accessible in an easy way to the code actually responsible for the interaction. We propose to generalize what some systems do for absolute devices: expose the fractional part by using floating values for the pointer coordinates. To summarize, we propose first that transfer functions should take into account device sensitivity; and second, that float values resulting from these computations should be forwarded *as is* to higher-level code.

This solution takes the sensitivity of the input device into account so that movements in motor space are no longer magnified by input sensitivity or constrained by display density. In theory, granularity of control can be increased 100 times for a high end 10000 CPI mouse (assuming a display density of 100 PPI). However, standard transfer functions must be adapted to unlock this potential, and this adaptation must be done carefully to maintain normal cursor behavior.

Taking the user into account

Having a device with a high sensitivity does not necessarily mean one can fully take advantage of it: users' capabilities and limitations should also be taken into account. As noted above, Bérard et al. found the *Device's Human Resolution* ($\text{RES}_{\text{human}}$) for mice to be in the range of 700 - 1400 CPI [4]. Devices with a sensitivity ($\text{RES}_{\text{input}}$) below $\text{RES}_{\text{human}}$ are not accurate enough to capture fine movements in motor space. But at the same time, users are not likely to benefit from higher sensitivities. We can thus define the *useful resolution* ($\text{RES}_{\text{useful}}$, in CPI) of a device and *epsilon* (ε, in inches), the smallest measurable displacement one can produce with it:

$$\text{RES}_{\text{useful}} = \min\left(\text{RES}_{\text{input}}, \text{RES}_{\text{human}}\right) \quad (1)$$

$$\varepsilon = \frac{1}{\text{RES}_{\text{useful}}} \quad (2)$$

Considering the current unitless gain applied by the pointing transfer function (G), we can calculate the number of practicable subpixels (S) as:

$$S = \frac{1}{\text{RES}_{\text{screen}}} \times \frac{\text{RES}_{\text{useful}}}{G} \quad (3)$$

where $\text{RES}_{\text{screen}}$ denotes the pixel density of the display (in PPI). S is actually the number of *epsilons* required to move from one pixel to another. Note that S necessarily varies over time since G is dynamic (speed-dependent, based on user movements). Note also that the highest S value $S_{\max}$ corresponds to the lowest gain level $G_{\min}$.

Adapting the transfer function

Modern pointing transfer functions are discrete, speed-dependent, and were generally designed for a 400 CPI mouse sampled at 125 Hz and a 96 PPI display refreshed at 60 Hz [5]. They typically produce gain values between 0.8 and 4.1 for motor speeds under 2.5 $cm.s^{-1}$ and then smoothly transition from these values to high ones at high speeds. Figure 4 shows a sample sigmoid function producing gain values in the same range for a particular hardware configuration. Assuming $\text{RES}_{\text{human}}$ = 1000 CPI and applying Equation 3 to $G_{\min} = 1.0$, this function would provide access to $S_{\max} \approx 11$ subpixels. But what if this was not enough?

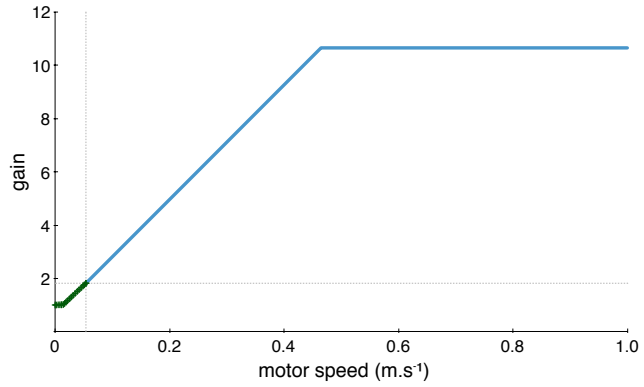


Figure 4: Sample sigmoid transfer function. The plot was made considering a 4000 CPI mouse sampled at 500 Hz and a 90 PPI display refreshed at 60 Hz. Green crosses indicate the 22 gain levels that generate subpixel motion.

We can determine the optimal gain G_{opt} which provides enough subpixels to select among all model elements with cardinality N given P pixels of screen space:

$$G_{opt} = \frac{P}{\text{RES}_{screen}} \times \frac{\text{RES}_{useful}}{N} \tag{4}$$

From this equation, it can be seen that when $P < N$ (i.e. $G_{opt} < \text{RES}_{useful} / \text{RES}_{screen}$) subpixels are beneficial. For example, using the video durations in Table 1 with RES_{useful} = 1000 CPI and RES_{screen} = 90 PPI, a one second accuracy in the QuickTime Player corresponds to a G_{opt} of 1.121 for the TV show, 0.648 for the generic movie, 0.245 for *Gone with the Wind*, and 0.134 for *Bergensbanen*. If we compare these numbers to the gain values in Figure 4, we see that the function should work for the TV show as-is (the corresponding G_{opt} is higher than G_{min}), but not the other videos. Achieving frame accuracy is even more challenging. Assuming a frame rate of 30 fps, G_{opt} values are much lower (0.037, 0.022, 0.008, and 0.004). In order to approach these optimum gain levels, we need a strategy for function adaptation.

To maintain normal pointer behavior, we alter an existing transfer function so that it produces G_{opt} at low speeds. Specifically, we calculate V_{min}, the minimum speed in meters-per-second on which the transfer function operates, and V_{use}, the speed associated with RES_{useful}:

$$V_{min} = \frac{0.0254}{\text{RES}_{input}} \times \text{FREQ}_{input} \tag{5}$$

$$V_{use} = \frac{0.0254}{\text{RES}_{useful}} \times \text{FREQ}_{input} \tag{6}$$

where FREQ_{input} is the frequency of the pointing device. Since there is no guarantee that people can actually move the device at such a slow speed like V_{min}, we maintain the G_{opt} value for all speeds under at least V_{use}. To maintain normal pointer behavior, we smoothly transition between this point (V_{use}, G_{opt}) and the point where the function starts producing motions of 1 pixel or more (V_{pix}, G_{pix}). Note that like G_{min}

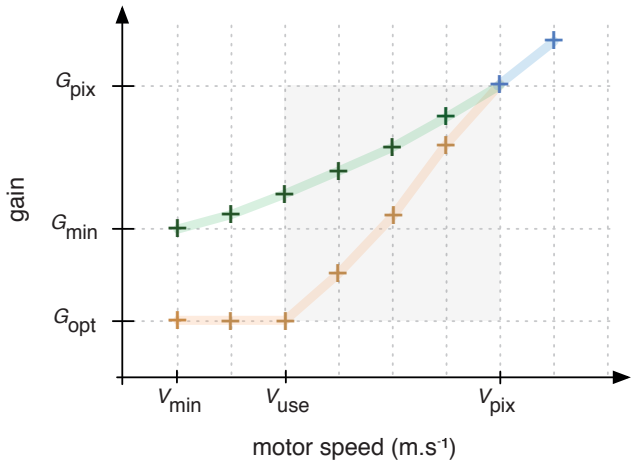


Figure 5: Closeup of the low-speed domain of a transfer function adapted to a particular model. Green and blue crosses show the original gains (subpixel & pixel). Yellow crosses show a possible adaptation.

and unlike V_{min} and V_{use}, the values of V_{pix} and G_{pix} depend on the function definition.

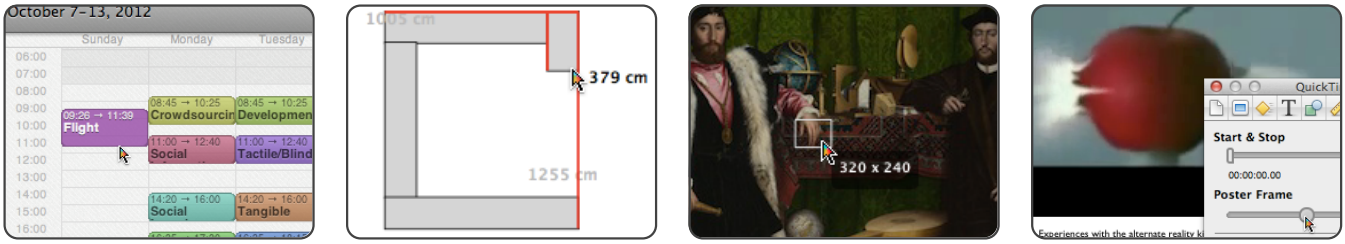
Figure 5 shows a closeup of the low-speed domain for a hypothetical transfer function that one might want to adapt. The blue crosses show the points of the original function that produce pixel motions, the green ones subpixel motions. The yellow crosses illustrate a possible interpolation between (V_{use}, G_{opt}) and (V_{pix}, G_{pix}). The distance between V_{use} and V_{pix}, the one between G_{opt} and G_{pix}, and FREQ_{input} constrain the interpolation. One needs enough speed steps (n_{steps}, Equation 7) to keep a reasonable distance between interpolated gain levels to reduce the risk of overshooting. At the same time, one does not want to alter the gain values corresponding to pixel motions since this would result in a possibly perceivable modification of the pointer behavior. Note that in a worse-case scenario, there may be no subpixel speed/gain combination (i.e. green cross) in the original function.

$$n_{steps} = \frac{V_{pix} - V_{use}}{V_{min}} \tag{7}$$

In the next section, we provide concrete examples of how this method can be applied to adapt a transfer function to a particular model, taking into account device and human capabilities.

ILLUSTRATIVE EXAMPLES

To create illustrative examples of subpixel interaction (Figure 6), we developed a cross platform software application written in C++ that runs subpixel-enabled applications through the WebKit browser engine. We used libpointing [5] to get raw information from devices, apply transfer functions taking into account hardware characteristics (input and output resolutions and frequencies) as well as DHR, and access remainders to produce floating pointer coordinates to control a custom pointer.



(a) (b) (c) (d)

Figure 6: Illustrative subpixel interaction applications: (a) manipulating calendar events with minute precision; (b) precise CAD object dimensioning; (c) image pixel accurate cropping; (d) selecting one frame of a video.

We used a sigmoid transfer function, similar to Figure 4 which is representative of transfer functions used by modern operating systems. Equation 8 describes this function which we refer to as F_G. It produces unitless gain values between 1.0 and 10.0, where dX represents a relative distance measured in motor space (in meters), dt is the elapsed time since the last input event (in seconds), $v = dX/dt$, and v_1 and v_2 are equal to 0.15 $m.s^{-1}$ and 0.5 $m.s^{-1}$ respectively. Thus, $F(dX,dt) = dX \times F_G(dX,dt)$ is the transfer function used by default to control the pointer.

$$F_G(dX,dt) = \begin{cases} 1 & : v \leq v_1 \\ 1 + 9 \times \frac{v - v_1}{v_2 - v_1} & : v \in]v_1; v_2[\\ 10 & : v \geq v_2 \end{cases} \quad (8)$$

When interacting with an object, we first compute G_{opt} as defined by Equation 4. If G_{opt} is below the minimal gain value defined by Equation 8 ($F_G(dX,dt) = 1$), then the transfer function is blended with G_{opt} to yield the function $H(dX,dt) = dX \times H_G(dX,dt)$, where H_G is described by Equation 9:

$$H_G(dX,dt) = \begin{cases} G_{opt} & : v \leq V_{use} \\ F_G(dX,dt) & : v > V_{pix} \\ (1-q)G_{opt} + q\,F_G(dX,dt) & : \text{elsewhere} \end{cases} \quad (9)$$

$$q = \frac{v - V_{use}}{V_{pix} - V_{use}}$$

We used a 22" Dell 2208WFP display set to its native resolution of 1680×1050 at 60 Hz, providing a pixel density of about 90 PPI. Our input device was a Microsoft Sidewinder X8 mouse with a 500 Hz polling rate. The mouse sensitivity was set to 4000 CPI, corresponding to its maximum configurable value. We fixed $RES_{useful} = 1000$ CPI. Combined with our particular transfer function, this configuration leads to the following values:

$$S_{max} \approx 11 \text{ for } G_{min} = 1.0$$
$$V_{use} = 0.0127 m.s^{-1}$$
$$G_{pix} = 1.0$$
$$V_{pix} = \frac{0.0254}{RES_{screen}} \times \frac{1.0}{G_{pix}} \times FREQ_{input} \approx 0.141 m.s^{-1}$$

$V_{pix}/V_{min} \approx 44$ discrete speed values below V_{pix} are available for subpixel interaction and transfer function blending.

Calendar

It is often convenient to enter and modify calendar events using direct manipulation. Depending on the number of hours in the current view, it may not be possible to set event times like flight or train departures requiring one minute precision: a height of 720 pixels is needed for a 12 hour view and 1080 pixels for 18 hours. These heights may exceed typical calendar window space. For example, an 18 hour view represented in a 400 pixel window requires a modest 2.7 subpixels for one minute precision. With subpixel input, it is even possible to represent a month view while still enabling individual event manipulation to the minute. The calendar application we created represents 17 hours in 272 pixels, yielding 16 pixels per hour (Figure 6a). With our particular setup, this view requires 3.75 subpixels, which our transfer function F_G provides. For subpixel feedback, start and end times are shown on each calendar event.

Computer-aided design

Computer-aided design (CAD) applications require precise specification of object dimensions and placement. Direct manipulation is preferable given the graphical nature of the task, but to achieve the required precision, current applications resort to text entry or zooming. Precise feedback is often shown already as a numerical measurement near the manipulated object, but with subpixel input, the manipulation can be made equally precise. We developed a small subset of an architectural CAD application where it is possible to dimension and position walls with one centimeter precision while maintaining a view of the entire structure. For example, consider a room plan where three walls are already drawn, and a fourth vertical wall of length 2205 cm must be created and aligned exactly. The whole drawing is viewed at a scale which maps 10 cm to 1 pixel (i.e. 1 : 354). Here 10 subpixels are required

to perform the task with required precision (1 cm), which our transfer function F_G again provides.

Cropping a high resolution image

Many digital cameras capture images exceeding 10 megapixels, yet most computer displays cannot show more than 2.5 megapixels. Cropping such an image to specific dimensions, say 320 by 240 pixels, usually requires zooming. The crop area can be so large that multiple zoom and pan operations are required making the task difficult. For example, consider a 17.8 megapixel image measuring 4215 by 4215 pixels. To show this entire image in a 300 by 300 pixel window, it must be scaled to 7% where one display pixel corresponds to 14 image pixels. To achieve image pixel precision, 14 subpixels are required. Since this is more than S_{max}, we need to switch from F_G to H_G (Equation 9) with $G_{opt} \approx 0.794$ (Equation 4). To facilitate subpixel output, a tooltip shows image-space coordinates for the subpixel pointer and image-space size of cropping rectangle (Figure 6c). When cropping according to visual details, lens-like feedback would be more appropriate.

Selecting a frame in a video

We replicated the Apple Keynote video interface where a 198 pixel wide slider is used to select a single "*Poster Frame*" (Figure 6d). As an example, consider the TV show in Table 1. At 30 frames-per-second, one pixel of the slider corresponds to $52 \times 60 \times 30/198 = 472.72$ frames (i.e. 15.75 s) in the video. To achieve frame-level precision, 473 subpixels are required: we again need to switch from F_G to H_G (Equation 9), this time with $G_{opt} \approx 0.023$ (Equation 4). We informally tested this scenario with several participants. In spite of the low value for G_{opt}, all managed to comfortably reach the intended frame and no one spontaneously noted any change in pointer behavior. Note that selecting one frame in this scenario corresponds to a 16.51 bit pointing task.

The strategies described in the above examples are specific to the transfer function F_G and the particular hardware configuration that we used. Had we used the same monitor with a low-end mouse (400 CPI, 125 Hz) and the default transfer function of Windows, OS X, or Xorg, S_{max} would have been less than 4. In all cases except the simplest Calendar example, it would have been necessary to switch from F_G to H_G.

DISCUSSION

In this section we provide guidelines for applying subpixel in generalized data manipulation situations and enumerate modifications which must be made to current operating systems and GUI toolkits.

Domain of applicability for subpixel interaction

There are three parameters which determine the applicability of a subpixel-enabled floating pointer and custom transfer functions: the number of available pixels (P); the number of practicable subpixels (S, from Equation 3); and the cardinality of the underlying model (N). Using these we can define two critical values for N: $N1 = S \times P$, the value above which custom transfer functions must be used to reach all values of the model; and $N2$, the value above which custom transfer functions can no longer operate due to usability issues. These determine four zones (illustrated in Figure 7):

- $N \leq P$: all values of the model can be addressed with a standard integer pointer, subpixel interaction is unnecessary but compatible;
- $P < N \leq N1$: all values of the model can only be addressed with subpixel interaction, but a standard transfer function is compatible without any change in pointer behavior;
- $N1 < N \leq N2$: to address all model values with subpixel interaction, a custom transfer function like those described earlier is required;
- $N > N2$: all model values cannot be addressed with subpixel interaction.

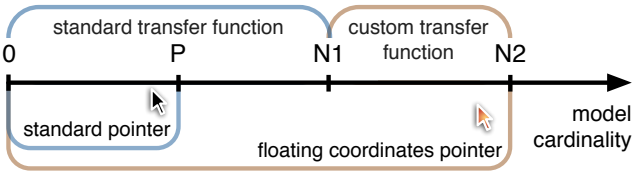


Figure 7: Four zones of applicability for subpixel and custom transfer functions (see text for description).

While N1 is unequivocally specified, N2 is harder to define. As model cardinality increases, G_{opt} decreases but the number of discrete speed steps available to interpolate between (V_{use}, G_{opt}) and (V_{pix}, G_{pix}) remains the same (n_{steps}, Equation 7). For the custom transfer function to remain usable, it must remain gradual. No matter the definition we might choose for this property, there will always be a point where n_{steps} is not enough to preserve it within $[V_{use}, V_{pix}]$. Extending the custom range over V_{pix} might also help, but it will undoubtedly result in a perceivable modification of the pointer behavior after some point. N2 captures this inevitable fact: for a given user and hardware configuration, there will always be an upper limit on the model cardinality that can be smoothly supported while preserving the standard pointer behavior. Note that using devices with higher sensitivity and frequency is the easiest way to push that limit since it reduces V_{min} which increases the number of discrete speed steps available for interpolation.

Impact on operating systems and GUI frameworks

As stated above, our subpixel prototypes were implemented using libpointing [5] to create a custom transfer function taking into account actual input device CPI and screen PPI. We also took into account human precision capabilities (DHR) and the cardinality of the model being manipulated to adjust the transfer function when required[4].

To support subpixel floating point coordinates universally, operating systems and GUI frameworks need to make relatively small, but fundamental changes. First, system transfer functions must take into account input device sensitivity so that higher device sensitivities result in higher precision. Currently, operating systems generally translate higher device sensitivity to faster pointer movements either because they do not take the sensitivity information into account or because it is not provided by the device [5]. Device configuration interfaces definitely need to be re-designed.

[4]Basic source code to create a subpixel libpointing application is available from `http://libpointing.org/`

Operating systems then need to use the remainders stored between two input events to create floating point coordinates. These subpixel coordinates would be dispatched by the windowing system, and ultimately received by the GUI framework. Framework event loops, methods, callback methods, etc. also need to be updated to handle floating point coordinates.

GUI frameworks also need to provide developers with a way to take into account human limb resolution directly or, even better, indirectly. Ideally, a developer would specify model cardinality and the framework would alter the transfer function used when a subpixel widget is manipulated. This also requires that operating system transfer functions can be changed dynamically, which is already achievable to some extent on some systems. Frameworks with dynamic layout capabilities can also adjust the size of widgets, making a subpixel enabled slider smaller without sacrificing precise control of the underlying data. For example, the preferred size of a slider would correspond to $N1$, and the minimal size of a widget would correspond to $N2$.

Since limb resolution varies between individuals [4], the operating system should provide a method to specify the current user's level of limb precision. This could be part of the input device settings with default values set conservatively based on the literature. Ideally, a simple calibration step would tune this value to specific individual, perhaps using a game-like procedure [6] rather than a dry experiment task like Bérard et al. [4]. Since human resolution can improve with practice, the calibration process should be updated intermittently, or automatically adapted over time based on patterns of use – like the strategy of adjusting the offset distance in the Shift pointing technique [13].

CONCLUSION AND FUTURE WORK

Subpixel interaction is a fundamental way to increase direct manipulation accuracy. For too long, positional input has been artificially constrained by design decisions made when input devices were about as accurate as the pixel density of a display. Now that input device sensitivity far surpasses display capabilities, the subpixel methods, transfer functions, and guidelines described above can enable interaction at a level of precision bounded only by human capability. Best of all, subpixel interaction does not change the way people interact, remains compatible with other precision techniques like ZUIs and Focus+Context, and would only require moderate changes to current operating systems and toolkits.

Our focus so far has been to enable subpixel interaction, but there is future work to extend its application context and test the potential benefit in actual settings. For example, formal testing of subpixel-enabled interfaces with realistic direct manipulation scenarios, experimental analysis examining the trade-off between pure subpixel interaction and subpixel interaction augmented by traditional precision techniques like ZUIs and Focus+Context, and investigating subpixel applications to very sensitive absolute touchscreens. There is also opportunity to extend the work of Bérard et al. [4] by measuring DHR with a wider range of devices and a broader range of participants. This will add even more substance to the overarching subpixel philosophy: pixels may be the limit of what we can see, but they should not limit what we can do.

REFERENCES

1. C. Ahlberg and B. Shneiderman. The alphaslider: a compact and rapid selector. *Proceedings of CHI'94*, 365–371. ACM, 1994.
2. C. Appert, O. Chapuis, and E. Pietriga. High-precision magnification lenses. *Proceedings of CHI'10*, 273–282. ACM, 2010.
3. Y. Ayatsuka, J. Rekimoto, and S. Matsuoka. Popup vernier: a tool for sub-pixel-pitch dragging with smooth mode transition. *Proceedings of UIST'98*, 39–48. ACM, 1998.
4. F. Bérard, G. Wang, and J. R. Cooperstock. On the limits of the human motor control precision: the search for a device's human resolution. *Proceedings of INTERACT'11*, 107–122. Springer-Verlag, 2011.
5. G. Casiez and N. Roussel. No more bricolage! Methods and tools to characterize, replicate and compare pointing transfer functions. *Proceedings of UIST'11*, 603–614. ACM, Oct. 2011.
6. D. R. Flatla, C. Gutwin, L. E. Nacke, S. Bateman, and R. L. Mandryk. Calibration games: making calibration tasks enjoyable by adding motivating game elements. *Proceedings of UIST'11*, 403–412. ACM, 2011.
7. Y. Guiard, M. Beaudouin-Lafon, and D. Mottet. Navigation as multiscale pointing: extending fitts' model to very high precision tasks. *Proceedings of CHI'99*, 450–457. ACM, 1999.
8. K. Perlin and D. Fox. Pad: an alternative approach to the computer interface. *Proceedings of SIGGRAPH'93*, 57–64. ACM, 1993.
9. E. Pietriga and C. Appert. Sigma lenses: focus-context transitions combining space, time and translucence. *Proceedings of CHI'08*, 1343–1352. ACM, 2008.
10. G. Ramos and R. Balakrishnan. Zliding: fluid zooming and sliding for high precision parameter manipulation. *Proceedings of UIST'05*, 143–152. ACM, 2005.
11. V. Scheib. Pointer lock. W3C Editor's draft, Web Applications Working Group, Apr. 2012.
12. B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. *Proceedings of VL'96*, 336–343. IEEE Computer Society, 1996.
13. D. Vogel and P. Baudisch. Shift: a technique for operating pen-based interfaces using touch. *Proceedings of CHI'07*, 657–666. ACM, 2007.