From Direct manipulation to Gestures

Caroline Appert

UNIVERSITÉ PARIS-SUD

HABILITATION À DIRIGER DES RECHERCHES

présentée par

Caroline Appert

Spécialité : Informatique – Interaction Homme-Machine

From Direct manipulation to Gestures:
Moving the Expressive Power from the Displays to the Fingers

le 26 Juin 2017

Michel Beaudouin-Lafon	Université Paris-Sud	Examinateur
Stephen Brewster	University of Glasgow	Rapporteur
Géry Casiez	Université Lille 1	Examinateur
Andy Cockburn	University of Canterbury	Rapporteur
Jean-Claude Martin	Université Paris-Sud	Examinateur
Laurence Nigay	Université Grenoble Alpes	Rapporteur
Shumin Zhai	Google	Examinateur

Habilitation à Diriger des Recherches préparée au sein
du Laboratoire de Recherche en Informatique de l'Université Paris-Sud
et d'Inria Saclay – Île-de-France

Acknowledgments

First of all, I would like to warmly thank all my jury members. I am very honored to have had the opportunity to present my work in front of so prominent researchers. I thank my *rapporteurs* (Stephen Brewster, Andy Cockburn and Laurence Nigay) who took some of their time to write thoughtful reports about my manuscript; and I thank all my jury members for having traveled to attend the defense (or stayed awake late for Andy), and for the insightful comments and discussions that followed the presentation. Special thanks to Shumin for having traveled from so far.

Thanks to all ILDA members. To Olivier who has been my *co-bureau* and friend for many years. To my PhD students (Hugo Romat, María Jésus Lobo and Rafael Morales) who all made me learn different aspects of what supervising means. To the Master students that I have supervised.

Thanks to many former InSitu members, and especially those who I worked with (Fanis Tsandilas, Stéphane Huot, Olivier Bau, Daniel Spelmezan, Halla Olafsdottir, Wendy Mackay, Jean-Daniel Fekete and, of course, Michel Beaudouin-Lafon).

I also think about all HCI people who I had fun with at different occasions (Olivier Bau, Nathalie Henry-Riche, Yann Riche, Fanny Chevalier, Stéphane Conversy, Nicolas Roussel, Renaud Blanch, Romain Primet, Mathieu Nancel, Julie Wagner, and many others). It is always great to see you at any possible occasion.

Friends for the administrative staff, Stéphanie Druetta and Alexandra Merlin, thank you!

Finally, many thanks to the ones I love. Kids (Antoine, Baptiste, Maxime and Thomas), I cannot express how lucky I am to have you in my life. Emmanuel... yes, yet another "..."... I love you.

Table of Contents

Abstract

Optimizing the bandwidth of the communication channel between users and the system is fundamental for designing efficient interactive systems. Apart from the case of speech-based interfaces that rely on users' natural language, this entails designing an efficient language that users can adopt and that the system can understand. My research has been focusing on studying and optimizing the two following types of languages: interfaces that allow users to trigger actions through the direct manipulation of on-screen objects, and interactive systems that allow users to invoke commands by performing specific movements. Direct manipulation requires encoding most information in the graphical representation, mostly relying on users' ability to recognize visual elements; whereas gesture-based interaction interprets the shape and dynamics of users' movements, mostly relying on users' ability to recall specific movements. This manuscript presents my main research projects about these two types of language, and discusses how we can increase the efficiency of interactive systems that make use of them. When using direct manipulation, achieving a high expressive power and a good level of usability depends on the interface's ability to accommodate large graphical scenes while enabling the easy selection and manipulation of objects in the scene. When using gestures, it depends on the number of different gestures in the system's vocabulary, as well as on the simplicity of those gestures, that should remain easy to learn and execute. I conclude with directions for future work around interaction with tangible objects.

Introduction

One of the grand challenges of research in Human-Computer Interaction (HCI) consists in optimizing the bandwidth of the communication channel between users and the system. Apart from the case of speech-based interfaces that rely on users' natural language, this entails designing an efficient language that users can adopt and that the system can understand. What "efficient" means depends on the type of users and the context of use. For example, operators whose goal is to maximize their productivity will need a vocabulary that allows them to invoke a limited set of commands very quickly. On the opposite, artists may not consider execution speed as highly critical, and will rather need a large vocabulary that allows them to explore a large design space. On average, good interaction design should make the language easy to learn and manipulate, allowing users to comfortably express their intent to the system with a reasonable speed.

This manuscript focuses on the two types of languages that are respectively used in *point-based* and *gesture-based* interfaces. The term *point-based* interfaces refers to graphical interfaces featuring objects that users can designate and manipulate with a pointing device to invoke commands. In those interfaces, most information is encoded in the graphical representation, and very few in users' movements: when users perform a pointing action, the system only considers the graphical object on which this action ends, ignoring its trajectory or speed. On the opposite, *gesture-based* interfaces associate movements with controls, allowing users to express a message to the system by executing a specific movement. This latter type of interface carries either a part or all of the information in users' movements, that feature varying shapes and dynamics.

Designing interfaces that are metaphors of the physical world is common in HCI. If we think about using the types of languages described above for communicating in the real world, we quickly understand what their potential strengths and limits are. Using a *point-based* paradigm for communicating would require people to convey concepts and designate things by reaching different objects one after the other. Giving people a rich expressive power thus means that the environment should feature a large number of objects. Also, to make people able to express themselves efficiently, the objects should be easy to reach. However, in an environment that contains many objects, those objects can be potentially very far and/or very small. A *gesture-based* paradigm does not rely on objects, but rather corresponds to adopting a sign language. The expressive power depends on the number of different signs and how they can express varying things and concepts. Of course, the signs should be easy to memorize and perform to offer an efficient means of expression. However, offering a large set of signs that has enough variability usually requires considering complex signs, implying that learning and manipulating them will require a lot of cognitive and motor resources.

What emerges is a tension between two types of limited resources: physical space, and human cognitive and motor abilities. Point-based interfaces heavily rely on *recognition*. Their expressive power is a function of the number of graphical primitives that the system can display, and their usability depends on the difficulty to point at each graphical primitive. Gesture-based interfaces rather resort to a *recall* paradigm. Their expressive power is a function of the different movements in the vocabulary of the system, and their usability depends on the difficulty to learn and perform each movement. Going from one type of interface to the other can be seen as transferring the expressive power from the *display* to users' *hands*.

This *display-hands* opposition between point- and gesture-based interaction is, of course, a simplistic characterization of interactions that rely on these paradigms. For example, a swiping gesture for deleting an object on a tactile screen relies on both display and hand. However, this display-hand opposition remains interesting to identify fundamental research questions about the two interaction channels independently. My research for the past ten years has been driven by such questions: *how can direct manipulation scale to large graphical scenes that are too large to fit on the display*? *how can users manipulate very small graphical objects*? *are users able to learn and perform large sets of gestures*? *how can we offer a high power of expression without resorting to complex gestures*? I believe that addressing these fundamental questions is crucial for designing efficient interactions in today's virtual environments that can feature very small to very large displays, and that can capture a large variety of user movements through multiple sensors.

1.1 Point-based Interfaces

As mentioned above, point-based interfaces may have to show a very large amount of graphical objects on the display. Zoomable interfaces (also called Multi-scale interfaces) have this capability as they can present a graphical scene that can be far larger than the display viewport (such as in, *e.g.*, Google Maps). To visualize a given area, users usually resort to navigation techniques such as traditional Pan&Zoom. However, when zoomed-in on a given region, users may miss important information from the surrounding context [JF98]. Focus+Context techniques such as fisheye lenses [CM01] offer an alternative by providing in-place magnification of a region without requiring users to zoom the whole representation. However, adoption of Focus+Context techniques may be hindered by both perceptual and motor issues when transitioning between focus and context. These usability problems have driven several of the research projects I have worked on.

We have first investigated how to design transitions that are more efficient than those that are solely based on spatial deformation by rather relying on dynamic behavior and translucence. We have proposed a design space for such transitions that we use for creating new lenses, called *Sigma lenses* [19, 21]. Our empirical study

showed that some of these new lenses outperform traditional magnification lenses for focusing on a given area. While well-designed lenses offer a good solution for *focusing* on objects, they can still suffer from usability issues when *interacting* with those objects. The typical implementation of Focus+Context techniques makes two representations of the data exist simultaneously at two different scales, with the focus region's location associated with the pointer, meaning that a one-pixel displacement may make the pointer jump by several pixels in the magnified (focus) region. This quantization problem, *i.e.*, the mismatch between visual and motor precision in the magnified region, gets worse with increasing zoom factors, thus limiting the range of applications that could offer magnifying lenses. We have studied this quantization problem and introduced new interaction techniques for *selecting* with a high-precision while preserving fast navigation performance [9]. Quantization is also partly responsible for the difficulty to follow a route with a magnifying lens, the route having a tendency to "slip off" the side of the lens. We have also designed lenses to make these *steering* tasks easier [1]. Our *RouteLenses* automatically adjust their position based on the geometry of a route making users able to comfortably navigate along paths of interest.

Most point-based interfaces allow users not only to select but also move elements through drag-and-drop actions according to the principles of direct manipulation [Shn87]. For example, users pan the view in multi-scale interfaces, they move and edit the geometry of elements in graphics editors, they adjust parameters using controllers such as sliders, or they move and resize windows. While direct manipulation stipulates that actions should be easily reversible, reverting changes made via a drag-and-drop usually entails performing the reciprocal drag-and-drop action. This can be costly, as users have to remember the previous position of the object and put it back precisely where it was. We have worked on identifying the inconsistencies that exist between the different situations where users perform drag-and-drop actions, and on proposing a unifying model, DND^{-1}, that allows users to easily undo and redo drag-and-drop actions in any situation [11]. Our Dwell-and-Spring widget [10] allows users to interact with this model in order to restore any past location of an individual object or of a group of objects.

The overall goal of these research projects is to push direct manipulation to its maximum expressive power by making users able to reach any graphical object and manipulate it. The related publications result from collaborative work with Emmanuel Pietriga, Olivier Chapuis, Olivier Bau (who was PhD student at that time), and two master students Jessalyn Alvina and María Jesús Lobo (now PhD students). Chapter 2 details these different projects.

1.2 Gesture-based Interfaces

Gesture-based interaction consists in associating a given human gesture with a command in the application. The HCI literature proposes different types of gestures that involve users' hands at different granularities, ranging from micro-movements of finger tips [RLG09] to whole-arm movements [NWP+11]. Some systems also rely on mid-air gestures (*e.g.*, [BBL93]) while others propose gestures that take their meaning relative to a device (*e.g.*, [BIH08]) or to a surface (*e.g.*, [WMW09]). Other dimensions can be identified to structure the potentially infinite design space of gestures. Proposing taxonomies for the use of gestures for interaction has actually retained HCI researchers' attention (*e.g.*, [Ks05, WMW09]). I worked on such a taxonomy for the specific family of stroke gestures in collaboration with Shumin Zhai *et al.* [26]. In our integrative review, we discuss the use of stroke gestures along cognitive aspects such as discovery and memorization, and also point at the difficulties of developing robust gesture recognizers. My projects on gesture-based interaction address both system and user aspects: engineering solutions for integrating gestures that are robustly recognized by the system on the one hand, and defining novel vocabularies of gestures that remain simple for users to memorize and perform on the other hand.

I started to work on recognition engines when I was a post-doc in 2007-2008 at IBM Almaden. I designed and developed a toolkit to implement stroke shortcuts in software applications with only a few lines of code, with the motivation that stroke shortcuts should not be more costly to implement than keyboard shortcuts are [13]. This first project was more focused on integrating gestures within traditional graphical interfaces. In terms of recognition engine, it was relying on a template-based algorithm that runs once the gesture is complete. Since then, I have worked on *incremental* gesture recognizers (*i.e.*, recognizing gestures *during their execution* as opposed to *after their execution*). I strongly believe that we should move towards this type of recognizer for two main reasons. First, it enables the development of interaction techniques that can guide users during gesture execution, supporting users in the discovery and learning of gestures. Second, with well-designed gesture vocabularies, incremental recognition enables transitions between different gestures for smoothly chaining command invocations and parameter adjustments. I have worked on designing incremental recognition engines for both single point and multi-touch input. In [3], we present an algorithm for estimating the scale of any partial single point input in the context of a gesture recognition system. We show how it can be used as a support for implementing OctoPocus [BM08], a visual guide that displays all available gestures in response to partial input. More recently, I have developed an incremental recognizer for a large vocabulary of multi-touch gestures that relies only on the last events in the finger input stream, meaning that users can switch between different gestures without resorting to explicit delimiters [17]. These gestures can thus be used to activate discrete commands as well as to adjust values of continuous parameters.

In 2011-2014, I coordinated an ANR JCJC project (MDGEST), whose core idea consisted of designing large vocabularies of gestures for small tactile surfaces (smartphones and tablets) that remain easy to memorize and perform. This had to be achieved by using gestures that remain simple in their shape by rather relying on other characteristics. Within the context of MDGEST, we have designed several novel vocabularies of gestures that rely, *e.g.*, on additional input channels: *tilt* [25], *pressure* [24] and *proximity* [23]. These vocabularies of gestures remain simple to perform while offering at least as much expressivity as the whole set of existing graphical widgets, and without consuming any screen space. For all these projects, we have designed a set of gesture primitives and we have implemented the associated recognizer either on a regular smartphone using built-in sensors (*e.g.*, accelerometers and gyroscopes) or on a smartphone that we equipped with some extra sensors (pressure or proximity) while taking care of preserving the device's initial form factor. During the last year of the project, we also studied how to augment the expressivity of multi-touch gestures, which we believe are not used to their full potential. In particular, we have shown that the system can analyze how fingers are positioned relative to each other to infer some user intentions at touch time, *i.e.*, before users actually perform the manipulation [18]. For multi-touch input on regular tablets, we have also proposed a vocabulary of gestures that vary along high-level dimensions (such as fingers' *movements relative* to one another, or the whole gesture's *frame of reference*) in order to offer a rich power of expression to users, while only relying on simple circular and linear shapes [17].

My research on gestures, presented in Chapter 3, aims at limiting the complexity of the vocabulary in order to offer a high power of expression without increasing the cost of learning and using the language. Users can actually perform an infinite number of different gestures but, to be useful in an interactive system, the gestures must remain easy for users to recall, and recognizable by the system. My publications on gesture-based interaction result from collaborative work with two postdocs (Daniel Spelmezan and Halla Olafsdottir), three permanent researchers from my research team (Emmanuel Pietriga, Olivier Chapuis and Theophanis Tsandilas), as well as four colleagues from all over the world (Shumin Zhai (Google, USA), Per Ola Kristensson (University of St Andrews, Scotland), Tue Haste Andersen (University of Copenhagen, Denmark), and Xiang Cao (Microsoft Research Asia, China)).

Direct Manipulation

Pivotal to direct manipulation is the ability to select and move objects in a graphical scene. This usually implies view navigation to adjust the display viewport, target acquisition to grab objects, and potential movements of these objects in the scene. This chapter presents my projects along these three fundamental interaction components: *navigation* (which I already started to investigate during my PhD [12, 20]), *acquisition* and *movement*.

As mentioned in the introduction, multi-scale interfaces can accommodate a large graphical scene featuring numerous objects in a limited display space. In such interfaces, users can pan in the 2D plane as well as move in altitude so as to either get an overview or visualize details [CKB09]. This interaction scheme has become very widespread in today's interfaces, and is especially useful to accommodate rich applications on portable devices. However, without appropriate navigation techniques, even simple tasks such as inspecting a local region or following a path can quickly become cumbersome.

Once users have navigated in the graphical scene to bring objects of interest in the viewport, they must be able to designate them with their pointing device to actually select them. Fitts' law [Fit54] accurately models this task and its associated difficulty in electronic worlds when the representation and the locomotion are similar to what we do in the physical world. However, by making two scales coexist at the same altitude, focus+context representations do not resemble the real world. The simultaneous existence of two scales introduces the quantization problem, *i.e.*, the mismatch between visual and motor precision in the magnified region, and forces us to reconsider what we know regarding target acquisition tasks.

Finally, according to the principles of direct manipulation, a lot of point-based interfaces also require users to move objects through drag-and-drop actions. Being able to change objects' location increases the expressive power of those interfaces but it also introduces some complexity related to undoing such moves, when, *e.g.*, repairing errors or when exploring different solutions. While a target acquisition (or selection) can usually be easily undone by designating the background, putting back an object where it exactly was is much more difficult.

This chapter presents our work about the different focus+context interaction techniques that we designed to offer an efficient means for navigating multi-scale interfaces and acquiring graphical objects. It then shows how our DND^{-1} model tackles the problem of reverting movements of objects performed using direct manipulation.

(a) (b) (c)

Figure 2.1 : Various transitions between focus and context: (a) step transition causing occlusion (MAGNIFYING GLASS), (b) distorting space (FISHEYE), (c) using gradually increasing translucence (BLENDING).

2.1 View Navigation

Typical pan&zoom techniques are based on a navigation scheme that imposes a sequence of zoom operations (typically performed using the mouse wheel or pinch gestures) and pan operations (usually performed using mouse drags or finger slides) [GBLB+04]. Using pan&zoom, reaching an object that is not visible in the current viewport requires changing the whole display's content, which may be cognitively demanding [CKB09]. Focus+Context techniques offer an alternative by providing in-place magnification of a region without requiring users to zoom into the representation. These techniques have been shown to be useful for navigating complex visual representations such as large trees [LRP95, MGT+03], graphs [GKN05], high-resolution bitmap representations [CLP04], and even graphical user interfaces featuring small controls [RCBBL07]. A focus+context representation allows users to concurrently preserve the context that the display offers and navigate at a zoom factor that is higher than that of the display. Contextual information can guide navigation when, *e.g.*, looking for particular localities in a map of a densely populated region, or when exploring the points of interest along an itinerary.

However, magnifying in place also introduces a transition area that can hinder the performance of focus+context techniques. For instance, simple magnifying glasses (Figure 2.1-a) create occlusion of the immediate context adjacent to the magnified region [RM93]; graphical fisheyes [SB94], also known as distortion lenses (Figure 2.1-b), make it challenging for users both to acquire targets [Gut02] and to follow trajectories. This section presents our extensions to Carpendale's framework for unifying presentation space [CM01], providing interface designers with novel types of magnifying lenses that facilitate *focus targeting* [19, 21] and *path following* [1] tasks.

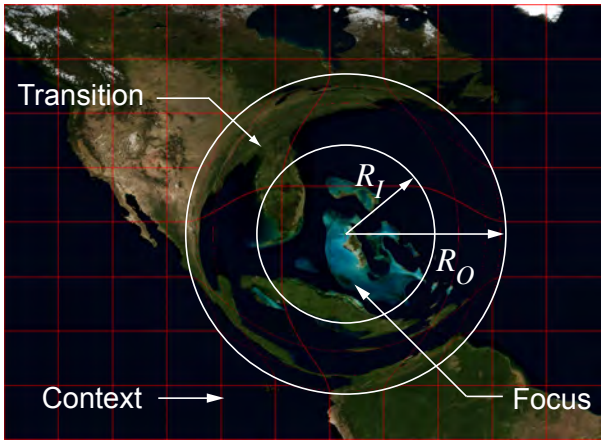


Figure 2.2 : Gaussian distortion lens. The level of detail in the flat-top is increased by a factor of $MM = 4.0$

2.1.1 Sigma Lens

In [19], we introduce the Sigma Lens framework that defines transitions between focus and context as a combination of dynamic scaling and compositing functions. This framework opens a design space to create a variety of lenses that use transformations other than spatial distortion to achieve smooth transitions between focus and context, and whose properties adapt to the users' actions. We identify lenses in this space that facilitate the task that consists in acquiring an object (*focus targeting*) and potentially exploring its surroundings (*local navigation*).

The framework

All constrained magnification lenses featuring a regular shape share the following general properties, no matter how they transition between focus and context (see Figure 2.2):

- R_I : the radius of the focus region (a.k.a the flat-top), which we call *inner radius*,
- R_O : the radius of the lens at its base, *i.e.*, its extent, which we call *outer radius*,
- MM : the magnification factor in the flat-top.

Applying a constrained lens to a representation effectively splits the viewing window into two regions: the *context region*, which corresponds to the part of the representation that is not affected by the lens, and the *lens region*, in which the representation is transformed. Since we want the lens to actually provide a more detailed representation of objects in the magnified region, and not merely duplicate pixels from the previous rendering, our framework relies on two buffers of pixels: the *context buffer*, whose dimensions $w \times h$ match that of the final viewing window displayed to the user, and the *lens buffer* of dimensions $2 \cdot MM \cdot R_O \times 2 \cdot MM \cdot R_O$.

In our approach, the overall process consists in applying a displacement function to all pixels in the lens buffer that fall into the transition zone: pixels between R_I and $MM \cdot R_O$ get *scaled* according to the drop-off function in such a way that they eventually all fit between R_I and R_O. Pixels of the lens buffer can then be *composited* with those of the context buffer that fall into the lens region.

Scaling. The standard transformation performed by graphical fisheyes consists in displacing all points in the focus buffer to achieve a smooth transition between focus and context through spatial distortion. This type of transformation can be defined through a drop-off function which models the magnification profile of the lens. The drop-off function is defined as:

$$\mathcal{G}_{scale} : d \mapsto s$$

where d is the distance from the center of the lens and s is a scaling factor. Distance d is obtained from an arbitrary distance function $\mathcal{D}$. A Gaussian-like profile is often used to define drop-off function $\mathcal{G}_{scale}$, as it provides one of the smoothest visual transitions between focus and context (see Figure 2.2). It can be replaced by other functions (see [CM01, CLP04]).

Compositing. The rendering of a point (x, y) in the final viewing window is controlled by function $\mathcal{R}$ below, where

$$p_{lens} \quad \otimes_{\alpha} \quad p_{context}$$

denotes the pixel resulting from alpha blending a pixel from the lens buffer and another from the context buffer with an alpha value of α. As with scale for distortion lenses, the alpha blending gradient can be defined by a drop-off function that maps a translucence level to a point (x, y) located at a distance d from the lens center:

$$\mathcal{G}_{comp} : d \mapsto \alpha$$

where α is an alpha blending value in [0, α_{FT}], α_{FT} being the translucence level used in the flat-top of the lens.

$$\mathcal{R}(x, y) =$$

$$\begin{cases} \forall (x,y) | \mathcal{D}(x,y) \leqslant R_I, & (x_c + \frac{x-x_c}{MM}, y_c + \frac{y-y_c}{MM}) \bigotimes_{\alpha_{FT}} (x,y) \\ \forall (x,y) | R_I < \mathcal{D}(x,y) < R_O, & (x_c + \frac{x-x_c}{\mathcal{G}_{scale}(\mathcal{D}(x,y))}, y_c + \frac{y-y_c}{\mathcal{G}_{scale}(\mathcal{D}(x,y))}) \bigotimes_{\mathcal{G}_{comp}(\mathcal{D}(x,y))} (x,y) \\ \forall (x,y) | \mathcal{D}(x,y) \geqslant R_O, & (x,y) \end{cases}$$

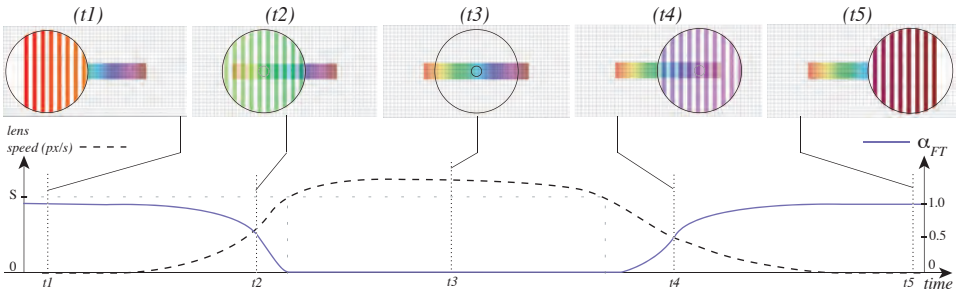


Figure 2.3 : SPEED-COUPLED BLENDING lens moving from left to right, with $\mathcal{S}(t)$ implemented as a low-pass filter.

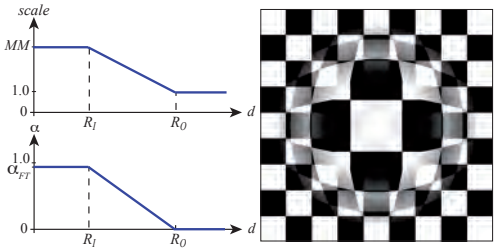


Figure 2.4 : HOVERING Lens

Speed-coupling. In addition to the transition functions $\mathcal{G}_{scale}$ and $\mathcal{G}_{comp}$, the Sigma Lens framework allows for lens properties such as magnification factor, radius or flat-top opacity to vary over time. The first example of lens to make use of dynamic properties was Gutwin's SPEED-COUPLED FLATTENING lens [Gut02], which uses the lens' dynamics (velocity and acceleration) to automatically control magnification. By canceling distortion during focus targeting, SPEED-COUPLED FLATTENING lenses improve the usability of distortion lenses. Basically, MM decreases toward 1.0 as the speed of the lens (operated by the user) increases, therefore flattening the lens into the context, and increases back to its original value as the lens comes to a full stop. Such behavior can easily be implemented by defining a time-based function $\mathcal{S}(t)$ that returns a numerical value depending on the velocity and acceleration of the lens over time. The function is set to return a real value in $[0.0, 1.0]$. Making a lens parameter speed-dependent is then easily achieved by simply multiplying that parameter by the value of $\mathcal{S}(t)$.

Instantiating Specific Lenses

In our approach to the implementation of the Sigma Lens framework, various constrained lenses are obtained easily, only by defining functions $\mathcal{G}_{scale}$, $\mathcal{G}_{comp}$, and $\mathcal{S}(t)$. We have implemented some examples of transitions with static lenses that

rely on scaling (Figure 2.1-b) or on compositing (Figure 2.1-c), and with dynamic lenses that implement a speed-dependent behavior in terms of scaling (such as Gutwin's SPEED-COUPLED FLATTENING lens) or compositing (our SPEED-COUPLED BLENDING lens illustrated in Figure 2.3). We have also implemented more complex transitions as with, *e.g.*, the HOVERING lens (Figure 2.4) that relies on both scaling and compositing, with a dynamic behavior for both its transparency level and flat top size. We conducted a series of experiments to assess the pros and cons of the different dimensions that the Sigma Lens framework features, as detailed next.

2.1.2 Focus Targeting

We first compared the *focus targeting* performance and limits of the five following lenses: a plain MAGNIFYING GLASS, a simple distortion lens (FISHEYE), and BLENDING, SPEED-COUPLED FLATTENING, SPEED-COUPLED BLENDING. We considered five different magnification factors (MM). Higher magnification factors make the task increasingly difficult: (i) the transition area becomes harder to understand as it must integrate a larger part of the world in the same rendering area, and (ii) it becomes harder to precisely position the target in the flat-top of the lens, the latter being controlled in the motor space of the context window. To test the limits of each lens, we included factors up to 14x. Our experiment was a 5×5 within-participant design: each participant had to perform several trials using each of the five lenses with five different magnification factors ($MM \in \{2, 4, 6, 10, 14\}$). A trial in our experiment consisted of a series of focus targeting tasks in every direction, as recommended by the ISO9241-9 standard [ISO00]. All details about our experimental design and statistical analyses are reported in [19]. Figure 2.5 summarizes our main results.

Interestingly, FISHEYE and BLENDING do not significantly differ in their performance. We initially thought that translucence could improve user performance by eliminating the drawbacks of space-based transitions. Transitioning through space indeed introduces distortion that makes objects move away from the approaching lens focus before moving toward it very fast, making focus targeting difficult [Gut02]. But BLENDING does not overcome this problem, as it introduces a new one: the high cognitive effort required to comprehend transitions based on gradually increasing translucence which, as opposed to distortion-based transitions, do not rely on a familiar physical metaphor.

We expected speed-based lenses (SPEED-COUPLED FLATTENING and SPEED-COUPLED BLENDING) to outperform their static versions (FISHEYE and MAGNIFYING GLASS). Each focus targeting task can be divided into two phases: in the first phase, the user moves the lens quickly to reach the target's vicinity, while in the second phase, she moves it slowly to precisely position the target in the focus. In the first phase, the user is not interested in, and can actually be distracted by, information provided in the focus region since she is trying to reach a distant object in the context as quick as possible. By smoothly and automatically neutralizing the focus and transition regions during this phase, and then restoring them, speed-

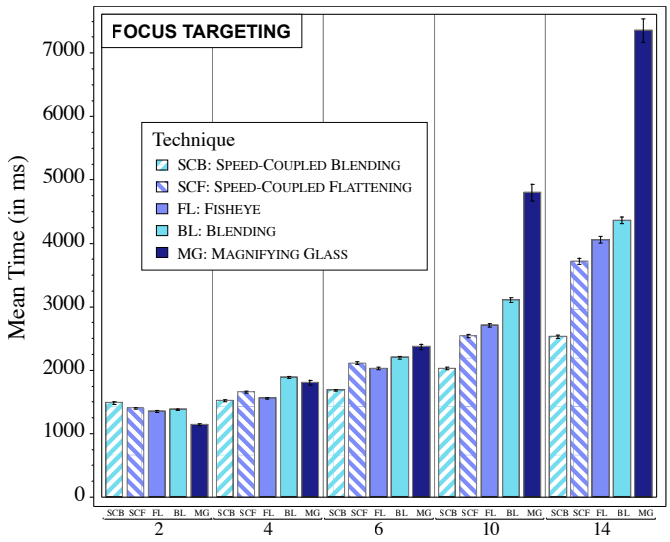


Figure 2.5 : Mean completion time per $Technique \times MM$ condition.

based lenses should help the user. Our results did actually support that this is the case for SPEED-COUPLED BLENDING and MAGNIFYING GLASS: smoothly neutralizing and restoring the focus of a MAGNIFYING GLASS by making it translucent does improve performance. However our participants were not significantly faster with SPEED-COUPLED FLATTENING than with FISHEYE. This was especially surprising since the study conducted in [Gut02] showed a significant improvement in users performance with SPEED-COUPLED FLATTENING. We think this inconsistency is probably due to implementation differences: we implemented SPEED-COUPLED FLATTENING as a constrained lens while it was implemented as a full-screen lens by Gutwin. In full-screen lenses, distortion affects the whole representation, which thus benefits more from the neutralization effect than constrained lenses that only affect a limited area.

2.1.3 Local Navigation

We then further investigated the performance of the two dynamic lenses, SPEED-COUPLED BLENDING and SPEED-COUPLED FLATTENING, by considering a more realistic task where (1) the graphical scene is more complex, and thus potentially causes legibility issues when using distortion or transparency, and (2) the object to explore does not fully fit into the lens' flat top, forcing local navigation, which may be difficult for users to perform with lenses that dynamically change. We conducted two experiments illustrated in Figure 2.6 based on two different types of representation: a network (vector graphics) for Experiment Exp_{graph} ($Bg = graph$), and a high-resolution satellite map (bitmap) for Experiment Exp_{map} ($Bg = map$). In both cases, participants are instructed to memorize a word as they will have to

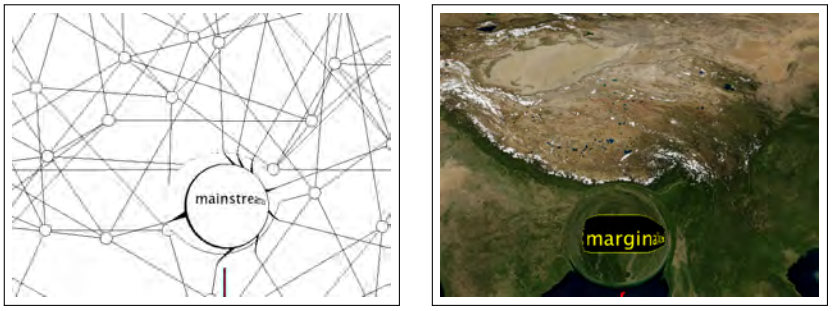
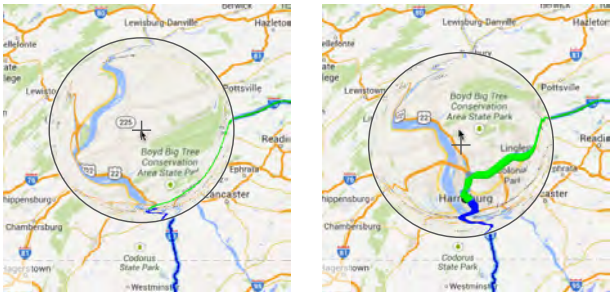


Figure 2.6 : Local navigation tasks in (a) a graph (labels displayed in black) and (b) in a map (labels displayed in yellow over a black background).

search for it in the representation. Once this target word is memorized, participants put the cursor on a red square (20×20 pixels) located at the center of the screen and press the space bar to start the trial. Words (including distractor words) appear successively in the same locations as the circular targets did in our *focus targeting* experiment discussed earlier. However, here, a word can never be fully displayed in the flat-top, forcing participants to perform local navigation. When they recognize the target word, participants press the space bar. We count an error if they press the space bar while the lens is over a distractor word. Here again, to compare lenses both in usual and extreme conditions, we use two magnification factors ($MM \in \{8, 12\}$). Font size is set to 42 pts (at context scale) for $MM = 8$ and 28 pts for $MM = 12$, so that the lens' flat-top can display at most 6 letters at full magnification. We use two word lengths to test the effect of the amount of local navigation on lens performance ($LabLength \in \{8, 12\}$). Finally, we consider two levels of $Opacity$ as we were hypothesizing that background and focus might be perceptually interpreted as one illegible image if contrast is not strong enough when making use of translucence. $Opacity$ was not included as a factor in Experiment Exp_{graph} because sharp edges displayed on a uniform background are strongly contrasted.

Our results revealed that, in terms of completion time, participants were faster using SPEED-COUPLED BLENDING than SPEED-COUPLED FLATTENING. However, this difference was not statistically significant. Differences in accuracy were stronger, with participants being more accurate using SPEED-COUPLED BLENDING than SPEED-COUPLED FLATTENING. Furthermore, differences between lenses in terms of accuracy increased with the magnification factor. In addition, lenses seem to be unequally affected by word length, the comparative gain of SPEED-COUPLED BLENDING over SPEED-COUPLED FLATTENING regarding accuracy is greater for longer words, tending to show that SPEED-COUPLED BLENDING better supports local navigation than SPEED-COUPLED FLATTENING does. This latter effect, observed only in Experiment Exp_{map}, reinforces our intuition that lens usability is affected by the type of representation. We also observed that SPEED-COUPLED FLATTENING is



(a) (b)

Figure 2.7 : Following an itinerary. (a) Conventional lens: the user overshoots at a right turn in Harrisburg; losing the route that falls in the distorted region. (b) RouteLens: the route's attraction compensates the overshoot; the lens remains closer to the route, which remains in focus.

more penalized by background type than SPEED-COUPLED BLENDING. While we were expecting usability problems due to the use of transparency especially with complex representations such as maps, SPEED-COUPLED FLATTENING was actually more affected by the background type than SPEED-COUPLED BLENDING was. We were even more surprised to observe that participants were more strongly affected by label opacity with SPEED-COUPLED FLATTENING than with SPEED-COUPLED BLENDING. As a summary, speed-coupled translucence does not have a negative impact on local navigation in our experiment. The SPEED-COUPLED BLENDING lens then appears as a very efficient technique for navigating even complex scenes that feature a low level of contrast between elements.

2.1.4 Path Following

Speed-based behaviors rely on the hypothesis that users do not seek information at a detailed level when moving a magnifying lens. While this is typically the case in focus targeting tasks when users want to reach a distant graphical object as fast as possible, this hypothesis does not hold for path following tasks such as when inspecting an itinerary. Focus+context techniques are conceptually well-suited to inspect itineraries by allowing users to see the entire route at once, and perform *magnified steering* [GS03] to navigate along the path and explore locally-bounded regions of interest. Navigation based on magnified steering has been shown to outperform regular pan&zoom for large steering tasks [GS03]. Yet, this task remains a challenging one for users, in part because paths have a tendency to "slip off" the side of the lens.

In order to make it easier for users to follow a route, we have designed RouteLens, a new content-aware technique that automatically adjusts the lens' position based on the geometry of the path that users steer through, so as to keep the lens on track in case of overshoot (Figure 2.7). RouteLens makes it easier for users to

follow a route, yet do not constrain movements too strictly. The lens is more or less strongly attracted to the path depending on its distance to it, and users remain free to move the lens away from it to explore more distant areas. RouteLenses decouple the lens' position from the cursor's position to give users the impression that the lens is attracted by the route. This separation between the motor and the visual space is similar to what Semantic Pointing [BGBL04] does when enlarging targets of interest only in motor space while leaving their visual counterparts unchanged.

When using RouteLens, all route segments whose distance to the system cursor is less than Δ apply an attraction force to the lens. The lens' position L is computed as a function of the system cursor's position C by using a weighted mean between all attracting route segments:

$$L = C + d_{min} \cdot \frac{\sum_{i=1}^{n} w_i \cdot A_i}{\sum_{i=1}^{n} w_i}$$

where A_i is the force vector that route segment i applies at position C to attract the lens (see below) and d_{min} is the distance between the cursor and the closest route segment.

To ensure continuous lens movements when a route segment starts or stops having an influence on the lens, w_i is set to $\Delta - d_{c,i}$, where $d_{c,i}$ is the distance between the cursor and route segment i.

For a given route segment, the attraction vector is computed as:

$$A = \alpha(d_c) \cdot (R_c - C)/d_c$$

where R_c is the point on the route closest to the cursor, and d_c the distance between the cursor and the route segment. α is a power function of d_c that parameterizes the force vector a route segment applies to the lens:

$$\alpha(d_c) = \begin{cases} 1 - \left(\frac{d_c}{\Delta}\right)^p & \text{if } d_c \leq \Delta \\ 0 & \text{otherwise.} \end{cases}$$

When steering along a magnified route, users want to minimize the distance d_l between the lens' center and the route. In Accot & Zhai's steering law [AZ97], d_l represents the movement's variability along the tunnel centered on the route, *i.e.*, the tunnel's width. The law stipulates that the larger the variability, the easier the movement. Figure 2.8 shows how RouteLens makes steering easier than a regular fisheye lens does, by allowing for a wider variability in user-controlled cursor movements. To keep a regular lens at a distance d_l from the route, users have to keep the cursor at a distance d_c = d_l. With a RouteLens, this distance can be larger: $d_c = d_l + d_c \cdot \alpha(d_c)$.

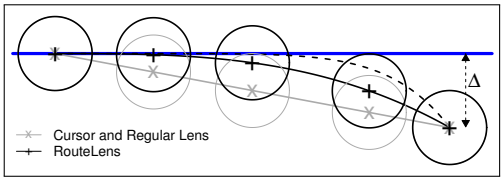


Figure 2.8 : Position of the cursor (grey line), and of the *RouteLens* (black line), that is vertically attracted ($p = 2$) by the route (bold blue line). The dashed black line shows the positions of the *RouteLens* when $p = 6$. In this figure, Δ is equal to the lens' flat-top diameter in motor space, and the black (resp. grey) circles show the part of the context displayed in the lens' flat-top.

We ran a study comparing conventional fisheye lenses (*RegularLens*) with fisheye lenses augmented with the attraction mechanism described above (*RouteLens*). While RouteLenses only affect the motor behavior of lenses, and can thus easily be combined with any type of graphical magnification lens [19][PPCP12], we considered conventional fisheye lenses as a baseline both to isolate the benefits of the attraction mechanism and to avoid a too lengthy experiment. Our experimental task consisted in following a route with a lens, always keeping the route visible in the flat-top. The experiment was a $2\times2\times2\times4$ within-subjects design with factors: TECH, ANGLE, DISTRACTOR, and DIR. TECH was the primary factor with two values: *RegularLens* and *RouteLens*. ANGLE and DISTRACTOR were secondary factors that defined characteristics of the route. ANGLE (*Acute* $= \pi/4$ and *Obtuse* $= 3\pi/4$) defined the angle between two route segments. DISTRACTOR defined the presence or absence of distractor routes, that also attract the cursor. When DISTRACTOR = *With*, additional (grey) routes were added at each turn of the (black) target route. DIR defined the direction of steering: left-to-right, right-to-left, top-to-bottom, or bottom-to-top. This factor was introduced for ecological reasons.

As expected, *RouteLens*' attraction effect made participants steer along the route with a movement that exhibits less variability. The average distance from the lens' center to the route was significantly lower for *RouteLens* than for *RegularLens*. The distance was 13.3 ± 0.9 pixels for *RouteLens* and 33.3 ± 1.7 pixels for *RegularLens* (expressed with respect to the flat-top's coordinate system). Also, *RouteLens* was significantly faster than *RegularLens*, a difference of $\sim 15\%$. Interestingly, the presence of distractors did not negatively affect *RouteLens*' performance. We even observed a comparative improvement of completion time for *RouteLens* over *RegularLens*. This may be due to the specific route layout we considered: a distractor route in the middle of the turn applies additional force vectors, resulting in a stronger global attraction towards the route at the end of the turn. Finally, qualitative feedback from participants revealed that participants hardly notice a difference between the two lenses, and that overall they express a preference for *RouteLens*.

2.2 Object Acquisition and Manipulation

Point-based interfaces rely on selections and movements of graphical objects. The Focus+Context techniques discussed above make it possible to implement graphical interfaces that feature a very large number of graphical objects and that users can still efficiently navigate. However, while navigation can be an end in, *e.g.*, information visualization tasks, it is often only a means to make some objects visible before actually interacting with them through selection and drag-and-drop operations. This section presents our work about facilitating selection, and providing a more flexible model for drag-and-drop interactions.

2.2.1 Acquiring objects with high-precision

The quantization problem

Early implementations of magnification techniques only magnified the pixels of the context by duplicating them without adding more detail, thus severely limiting the range of useful magnification factors (up to 4x). Newer implementations, in Carpendale's original framework [CLP04] or in the Sigma Lens extension introduced earlier, do provide more detail as magnification increases. Theoretically, this means that any magnification factor can be applied, if relevant data is available. In practice, this is not the case as another problem arises that gets worse as magnification increases: *quantization*.

Lenses are most often coupled with the cursor and centered on it. The cursor, and thus the lens, are operated at context scale. This allows for fast repositioning of the lens in the information space, since moving the input device by one unit makes the lens move by one pixel at context scale. However, this also implies that when moving the input device by one unit (dot), the representation in the magnified region is offset by MM pixels, where MM is the focus' magnification factor. This means that only one pixel every MM pixels can fall below the cursor in the magnified region. In other words, some pixels are unreachable, as visual space has been enlarged in the focus region but motor space has not. Objects can thus be *difficult or even impossible to select*; even if their visual size is above what is usually considered a small target (less than 5 pixels). The square representing Arlington station in Figure 2.9-(Left) is 9-pixel wide, yet its motor size is only 1 pixel. Figure 2.9-(Right) illustrates this quantization problem with a space-scale diagram [FB95]: the center of the lens can only be located on a pixel in the focus window that is *aligned* – on the same ray in the space-scale diagram – with a pixel in the context window. The space-scale diagram shows that the problem gets worse as magnification increases.

High-precision Lenses

In [9], we introduce several strategies that decouple the cursor from the lens' center in order to resolve the mismatch between visual and motor space precision

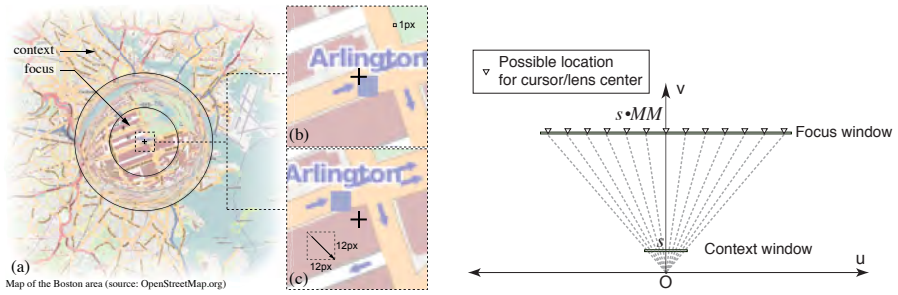


Figure 2.9 : Focus+Context techniques and the quantization problem. (Left) Moving the lens by one unit of the input device South and East makes the cursor jump several pixels in the detailed representation (magnification factor MM = 12). (Right) Space-scale diagram of possible locations for lens center (each ray corresponds to one pixel in context space).

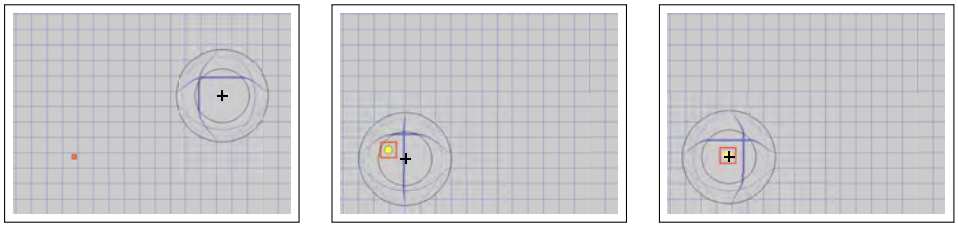
in the focus region. Our techniques' design aims at making it possible to perform both fast navigation for focus targeting and high-precision selection in the focus region in a seamless manner.

- *Key* is a simple mode-switching technique. It uses two control modes: a *context speed* mode and a *focus speed* mode. It requires an additional input channel to perform the mode switch, for instance using a modifier key such as SHIFT. Users can then navigate large distances at *context speed*, where one input device unit is mapped to one context pixel, *i.e.*, MM focus pixels, and perform precise adjustments at *focus speed*, where one input device unit corresponds to one focus pixel.

- *Speed* is inspired by techniques featuring speed-dependent properties (*e.g.*, [CLP09, Gut02, IH00]. We map the precision of the lens control to the input device's speed with a *continuous* function, relying on the assumption that a high speed is used to navigate large distances while a low speed is more characteristic of a precise adjustment (as observed for classical pointing [Bal04]).

- *Ring* is inspired by Tracking menus [FKP+03]. With this technique, the cursor can freely move *within* the flat-top at *focus scale*, thus enabling pixel-precise pointing in the magnified region. When the cursor comes into contact with the flat-top's border, it pulls the lens at *context speed*, enabling fast repositioning of the lens in the information space.

A *pointing task* with a lens is typically divided in two main phases: (i) *focus targeting*, which consists in putting a given target inside the flat-top of the lens (Figure 2.10-(a) and (b)) and (ii) *cursor pointing* to precisely position the cursor over the target (Figure 2.10-(b) and (c)).



(a) (b) (c)

Figure 2.10 : Acquiring a target with a lens: *focus targeting* from (a) to (b) and, *cursor pointing* from (b) to (c).

The *focus targeting* task has an index of difficulty of about:

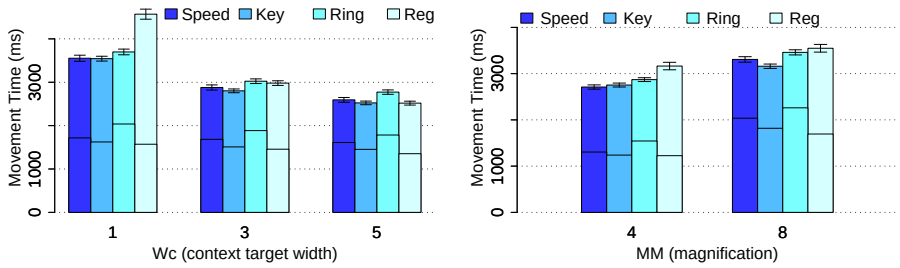
$$ID_{FT} = log_2(1 + \frac{D_c}{(W_{FT_c} - W_c)})$$

where W_{FT_c} and W_c are the respective sizes of the flat-top and the target in context pixels, and D_c is the distance to the target in context pixels as well[1]. This formula clearly shows that difficulty increases as distance increases, as the size of the flat-top decreases, and as the size of the target decreases. The size of the flat-top in context pixels is directly related to the magnification factor of the lens, MM. Indeed, the size of the flat-top is fixed in terms of focus pixels, so the higher MM, the smaller the size of the magnified area in context pixels.

The final *cursor pointing* task mainly depends on the area of the target in focus space that intersects the flat-top after the focus targeting task. The larger this area, the easier the cursor pointing task. We can at least consider the best case, *i.e.*, when the target is fully contained in the flat-top. In this case, the difficulty of the cursor pointing task can be assessed by the ratio $\frac{D_f}{W_f}$ where D_f is the distance between the cursor and the target, and W_f is the motor size of the target when magnified in the flat-top. The distance D_f is small, *i.e.*, smaller than the flat-top's diameter, so we assume that the difficulty of the cursor pointing task is mainly caused by the value of W_f. For regular lenses, the value of W_f is actually the size of the target at context scale because the target is only visually magnified. With our lenses, however, since pixel-precise selections are possible, W_f is the magnified size of the target (at focus scale).

We conducted two experiments that involve pointing tasks. In the first experiment, we considered tasks with an average level of difficulty in order to test whether any of our three techniques degrade performance when compared with regular lenses (*Reg*). In the second experiment, we asked participants to perform tasks with a very high level of difficulty, which involve targets smaller-than-a-pixel wide at context scale and which regular lenses do not support.

[1] ID_{FT} is the exact index of difficulty when the target must be fully contained in the flat-top. Here the task is slightly easier because the target just has to intersect the flat-top.



(a) (b)

Figure 2.11 : First experiment. (a) Movement time per TECH × WC. (b) Movement time per TECH × MM. The lower part of each bar represents focus targeting time, the upper part cursor pointing time.

Figure 2.11 illustrates the main results for our first experiment. As expected, regular lenses (*Reg*) performed worse than the three other techniques. This is likely because, as we explain above, the target's motor size is in context pixels for *Reg* whereas it is in focus pixels for *Key*, *Speed* and *Ring*. The comparative differences between the three other techniques are quite small. The only significant difference is actually between *Key* and *Ring*, with *Key* being faster than *Ring*.

More interestingly, the interaction effect TECH×MM on movement time suggests that *Ring* suffers more than the other techniques from an increasing magnification factor. A closer look reveals that the time for performing the focus targeting phase is proportionally longer for *Ring*. This is probably due to the cost of repairing overshoot errors during this phase: changes in direction are costly with *Ring* since the user first has to move the cursor to the opposite side of the flat-top before being able to pull the lens in the opposite direction.

The interaction effect TECH×WC on movement time is also interesting as it highlights that the differences really matter for small targets (WC= 1 and WC=3). *Key*, *Speed* and *Ring* are significantly faster than *Reg* only for WC=1 and WC=3. The difference is not significant for WC=5. In the latter case, only *Speed* is significantly faster than *Reg*. Moreover *Ring* is faster than *Key* for WC= 1, while *Speed* is not. These results suggest that *Ring* is particularly efficient for very small targets and that *Speed* is more appropriate for larger ones.

Our first experiment thus supports that, in comparison with regular lenses, our precision lenses improve user experience when pointing at small targets. Our second experiment aimed at assessing the comparative performance of those precision lenses in extreme cases that regular lenses cannot support: very small target sizes (less than one pixel in context scale) and high magnification factors. We used the same experiment set up, but discarded the *Reg* technique as it is not capable of achieving sub-pixel pointing tasks, and considered targets that have a size in focus (WF) $\in \{3, 5, 7\}$.

Figure 2.12 illustrates our main observations. *Ring* and *Key* are significantly

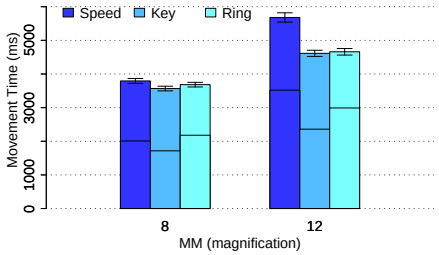


Figure 2.12 : Movement time per TECH×MM. The lower part of each bar represents focus targeting time, the upper part cursor pointing time.

faster than *Speed* but only for MM=12 while these differences are not significant for MM=8. This large difference at MM=12 is due to a sharp increase of focus targeting time (*FTT*) for *Speed*. Comments from participants confirm that the speed dependent control of motor precision is too hard when the difference between context scale and focus scale is too high, resulting in abrupt transitions.

As in our first experiment, we observe that focus targeting performance of *Ring* degrades as MM increases. However, good cursor pointing performance compensates for it, resulting in good overall task completion time. During the cursor pointing phase, *Ring* is stationary; only the cursor moves inside a static flat-top. This is not the case for *Key* and *Speed* for which high-precision cursor pointing is achieved through a combination of cursor movement and flat-top offset. As a result, the control-display gain is divided by MM for *Key* and *Speed*, resulting in a loss of precision that makes the pointing task more difficult with those two latter techniques than with *Ring*.

To summarize, when pushed to extreme conditions, the *Speed* lens becomes significantly slower than the other precision lenses while *Ring* remains as fast as *Key* without requiring an additional input channel for mode switching.

Finally, we designed a family of four high-precision lenses by combining our mechanisms for solving the quantization problem with visual behaviors from the Sigma Lens framework. We chose the two Sigma lens visual designs that we observed as the most efficient ones (SPEED-COUPLED BLENDING – abbreviated *Blend*, and SPEED-COUPLED FLATTENING – abbreviated *Flat*), and we combined them with either speed-dependent motor precision (*Speed*) or cursor-in-flat-top motor precision (*Ring*). *Key* was discarded because it proved awkward to combine explicit mode switching with speed-dependent visual properties.

Speed + *Flat*: this lens behaves like the original *Speed* design, except that the magnification factor decreases toward 1 as speed increases. The main advantage is that distortion no longer hinders focus targeting. Additionally, flattening provides indirect visual feedback about the lens' precision in motor space: it operates in context space when flattened, in focus space when not flattened.

Ring + Flat: This lens behaves like the original *Ring* design, with the magnification factor varying as above. As a consequence, the flat-top shrinks to a much smaller size, thus making course corrections during focus targeting easier since the cursor is still restricted to that area. As above, distortion is canceled during focus targeting.

Ring + Blend: This distortion-free lens behaves like the original *Ring* design, except that the restricted area in which the cursor can evolve (the flat-top) is larger. As speed increases, the flat-top fades out, thus revealing the context during the focus targeting phase. An inner circle fades in, representing the region that will actually be magnified in the flat-top if the lens stops moving. The cursor is restricted to that smaller area, making course corrections less costly.

Speed + Blend: This lens behaves like the original *Speed* design without any distortion. As above, the flat-top fades out as speed increases and fades back in as speed decreases. Again, the larger flat-top reduces the focus targeting task's index of difficulty. In a way similar to *Speed + Flat*, blending provides indirect visual feedback about the lens' precision in motor space: it operates in context space when transparent, in focus space when opaque.

In a final experiment comparing those four hybrid lenses to the static *Ring* and *Speed* designs from our first two experiments, our participants saw their pointing performance further improved by the visual designs from the Sigma Lens framework. This supports that the gains from our high-precision mechanisms can be combined with the gains from our advanced visual designs.

2.2.2 Moving objects

The projects presented above facilitate object acquisition, which is the most basic and frequent action that users perform in point-based interfaces. The complementary key component to point-based interaction is object displacement, which is usually enabled through drag-and-drop actions in graphical applications both on desktop computers and touch-sensitive surfaces. Drag-and-drop is used to pan the view in multi-scale interfaces, to move and edit the geometry of elements in graphics editors, to adjust parameters using controllers such as sliders, or to move and resize windows.

Objects manipulated via drag-and-drop often have to be restored to one of their previous positions. For instance, a user will carefully lay out windows on his desktop but will then temporarily move or resize one of them to access content hidden behind it, such as an icon or another window of lesser importance that was left in the background; he will then want to restore the foreground window to its earlier configuration. The reader of a document will scroll down to an appendix or check a reference, and will then want to come back to the section he was reading. Current systems do not enable users to easily restore windows or viewports to their earlier configuration; users have to manually reposition and resize the corresponding objects. Such actions can be costly. From a motor perspective, the cost of *repairing* a drag-and-drop manipulation can be higher than that of the original

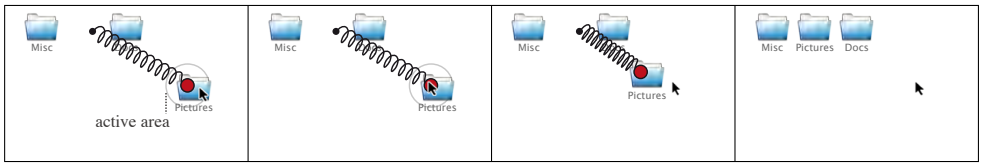


Figure 2.13 : The Dwell-and-Spring technique (DS). A red circular handle pops up close to the cursor when the user presses the mouse button and remains still for 500ms (*i.e.*, dwells) over an icon. Releasing the mouse button while the cursor is over the spring handle will undo the last move of this icon.

manipulation depending on how precisely the object has to be positioned. This is especially true for touch-based interfaces, which can make precise manipulations challenging [SRC05]. The cost can also be high from a cognitive perspective, as users may have difficulty remembering what was the previous state of a particular object [KLDK08].

We studied typical situations where users have to perform a *reciprocal drag-and-drop* in order to restore objects to their past locations. We first designed the Dwell-and-Spring interaction technique [10] that allows users to undo movements of individual objects according to a simple linear undo model. We then introduced the DND^{-1} model [11] that can handle all past locations of individual objects and groups of objects, which led us to redesign Dwell-and-Spring in order to allow users to navigate objects' histories and perform the reciprocal drag-and-drop action of interest.

Reciprocal Drag-and-drop: Simple cases

Situations that call for reciprocal drag-and-drop can be simple: for instance, putting a window back to its last location or reverting it to its previous size.

The basic Dwell-and-Spring technique, as described in [10], readily applies to all simple cases of reciprocal drag-and-drop. Figure 2.13 illustrates it on a very simple case, where an icon gets restored to its last position. A red circular handle pops up close to the cursor when the user presses the mouse button and remains still for 500ms (*i.e.*, dwells) over the icon. Bringing the cursor or finger onto this handle will make a spring appear, showing what the center of the icon will become if the user releases the mouse button or lifts his finger over the spring handle. If the user dwells without having initiated any movement, the spring shows the last move that was applied to the icon. If the user has already initiated a drag-and-drop, the spring proposes the reciprocal drag-and-drop for the current move. The user can either move over the spring handle and select it, activating the spring and thus bringing back the object to its previous location; or he can discard the widget by getting out of the active area.

As illustrated in Figure 2.14, this version of Dwell-and-Spring supports various cases of reciprocal drag-and-drop: manipulating icons on the desktop, navigating

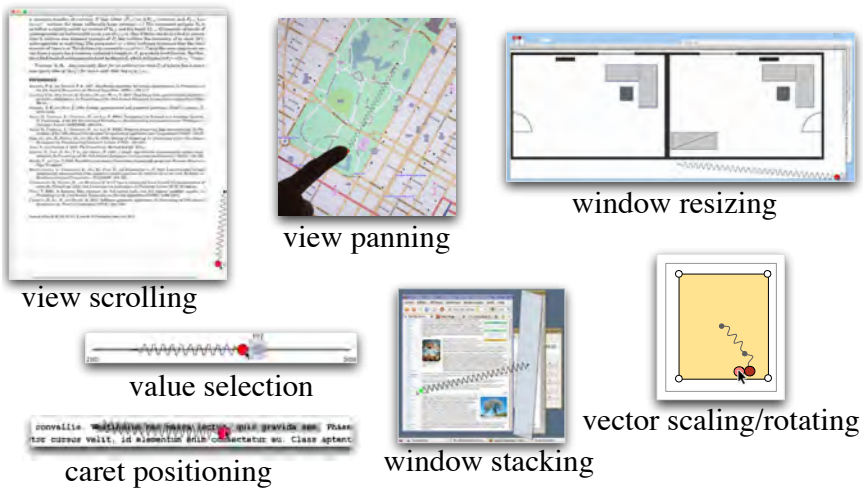


Figure 2.14 : Examples of frequent drag-and-drop actions that may call for reciprocal drag-and-drop actions.

documents using a scrollbar or with a swipe gesture on a touch-sensitive surface, moving and resizing windows, or any other action where the spring's actions are equivalent to what the user would manually do to revert to the original state, like moving a slider knob or a manipulation handle. In this original version, Dwell-and-Spring is only able to revert the current or the last drag-and-drop, as it is only keeping track of the previous location of each object, based on a per-object linear undo model.

We conducted an experiment to capture what users typically do in situations where they want to revert a drag-and-drop. We also wanted to evaluate whether the spring metaphor implemented in Dwell-and-Spring is a viable alternative or not. The experiment contained two parts: (1) an interactive *in-situ* questionnaire to gather data about users' habits when reverting drag-and-drop actions in different contexts of use (view navigation, window management, vector graphics editing, etc), and (2) a formal experiment to evaluate how easy it is to discover and understand Dwell-and-Spring, and how often users would actually resort to it once discovered.

First, our general observation about users' habits was that they always repair their direct manipulation errors manually, except when the direct manipulation acts at the functional level of the corresponding application (*e.g.*, moving a shape back where it was with the undo command in a vector-graphics editor). Second, when the environment proposes Dwell-and-Spring for reverting moves, one third of users spontaneously tried to make use of it. Demonstrating the technique even a single time was sufficient for users to understand and adopt it. Finally, our quantitative analysis highlighted the speed-accuracy trade-off of using Dwell-and-Spring:

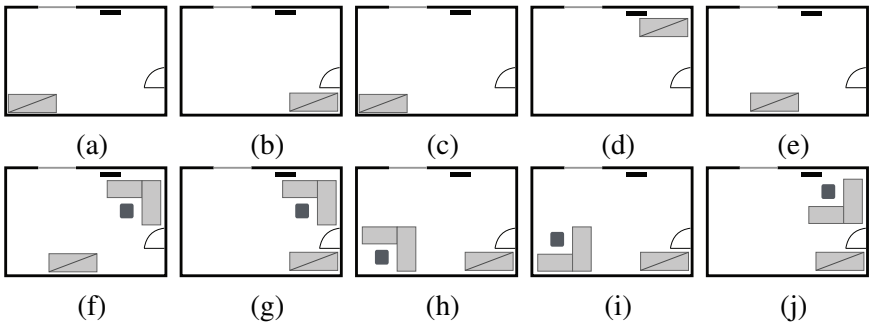


Figure 2.15 : Exploring different office layout alternatives on a floor plan. (a) Placing a cupboard in the SW corner. (b) When moving the cupboard to the SE corner, it is difficult to access it when the door is open. (c) Cupboard back to the SW corner. (d) Cupboard in the NE corner. The heater is partially occluded. (e) Cupboard almost centered along the S wall. (f) Adding a desk in the NE corner, composed of two tables and a chair. The heater is partially occluded. (g) Cupboard back in the SE corner to free space for the desk in the SW corner. (h) Desk in the SW corner. (i) Changing the relative placement of the desk elements. (j) Desk back in the NE corner with the new relative layout between the two tables and the chair.

while it may be a bit slower in some cases, Dwell-and-Spring accurately cancels or undoes any direct manipulation, which can be a significant advantage for precise positioning.

Reciprocal Drag-and-drop: Advanced cases

Situations calling for reciprocal drag-and-drop can be much more elaborate than simply restoring an object's last location: for instance, putting back a group of shapes to an earlier position on the drawing canvas after having manipulated other shapes, while preserving the new relative position that was given to the shapes in the group after they were initially moved away. From a user perspective, such graphical layout tasks are often part of an exploratory process. For instance, Figure 2.15 illustrates a scenario in which a person rearranges furniture in an office and tests alternative layouts. The software allows her to explore different arrangements by selecting and moving either a single piece of furniture, or multiple pieces together. Direct manipulation strongly contributes to making such exploratory design activities easy. But effectively supporting users also entails enabling them to easily revert back to past states from which to try other design options. Most graphical editing software provides an undo command to restore a past state of the entire document but, unfortunately, the underlying undo model is usually a global linear one that does not keep track of branches in the history of manipulations. Such a basic undo mechanism has two strong limitations, as detailed below.

The first limitation is that some previous states in the history can become inac-

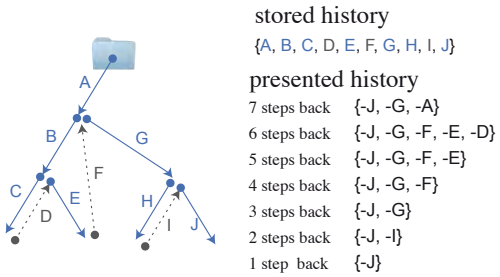


Figure 2.16 : DND^{-1} stores all repositioning actions applied to an object, including those performed via a reciprocal drag-and-drop (D, F and I, shown as dashed black lines). It presents the shortest path to all past locations.

cessible [Ber94, YMK13]. In Figure 2.15, the user moves the cupboard (a-b) but then undoes this move (b-c) when she realizes that this location might not be so convenient because of its proximity to the door. Later, after having considered the different constraints (window, heater, additional furniture), she finally decides that putting the cupboard behind the door (as in (b)) is the best option. She wants to revert it to this location, but as she has moved it to other locations (c-d-e) after her undo operation (b-c), she is no longer able to get back to this configuration other than by manually moving it back there.

The second limitation comes from the lack of integration of object selection mechanisms with the history of direct manipulations. In Figure 2.15, the user moves the two tables and the chair that make her workstation (g-h), and then changes their relative layout, thus breaking the previous multiple selection (i). Because there can only be one single active selection at a time, testing a location of the workstation that has already been explored (f), but with the new relative layout made in (i-j), requires selecting all its elements again and manually dragging-and-dropping them in the right place. Some graphical editors feature a command to group objects together. But this makes the exploratory design process much more cumbersome, as groupings have to be anticipated and created explicitly. In addition, groupings set persistent links between objects, which impede single-object editing operations.

All Past Locations of an Object Applications that support undo typically store the history of actions as a tree whose nodes are the different states of the application. Performing an operation means adding a novel child state to the current node. Undoing an operation means getting back to the parent node. The linear undo model that most applications propose only supports one *single active path*. All nodes outside this path are inaccessible via undo. For instance, in Figure 2.16, the user moves the icon three times successively (displacements A, B then C), reverts C, and then moves the icon again by E. At this point, she can no longer recover the position the icon had after displacement C, since this one no longer belongs to the

active path ({A,B,E}). A few applications, such as Emacs [Gos82, Yan92], make users able to recover any state. However, this might require chaining a long series of interactions to reach a given state, as the history stack is presented to them as a *full sequential path* in the history tree.

Figure 2.16 illustrates DND^{-1}, our local undo model to navigate in the history tree of displacements performed on a graphical object. This model lies in-between the *single active path* and the *full sequential path* models described above. It stores the full history of repositioning actions, but provides users with shortcuts to quickly access nodes in the tree, so as to make them able to recover any past location, in the spirit of the Selective Undo model [Ber94]. Each object remembers the sequence of moves that were applied to it, including reciprocal drag-and-drop actions. All past locations are accessible. Also, when a user invokes a reciprocal drag-and-drop action to restore a past position P of an object O, DND^{-1} adds the straight move between O's current location and P to the end of the history, in the spirit of the *inverse model* for selective undo (used in *e.g.*, [Ber94, Mye98, YMK13]), rather than inserting all reciprocal moves after the corresponding moves in the history, as the *script model* (used in, *e.g.*, [KF88, MLL$^+$15]) would have done.

However, keeping a trace of all past moves also means that the number of steps to revert to a past location can be very long. In order to present an object's history of past locations in a compact way, we have implemented a navigation algorithm that computes the shortest path in the history to reach any of these past locations. This is basically achieved by removing cycles, *i.e.*, series of moves that bring the object back to a location already present on the path. For instance, navigating two steps back with DND^{-1} after move J in Figure 2.16-b entails following path {-J, -I}; but navigating three steps back entails following path {-J, -G}, as {-J, -I, -H} would have led to the same location than {-J}, which is already proposed for a one-step-back navigation. This simplification decreases the number of steps that should be presented to the user, while ensuring that he can reach any past locations.

Groups of Objects To keep track of all multiple selections an object has belonged to, the DND^{-1} model stores the whole history of moves into a hashtable ($History$) whose keys are groups of objects (which can be singletons) that index lists of timestamped movements. Because there can be some ambiguity when deciding what a group is, as the same objects can be involved in different multiple selections, we have designed a strategy for handling groups that is detailed in [11]. This strategy makes users able to not only keep a trace of previous multiple object selections even if other selections happened afterwards, but also restore objects' locations individually.

The first advantage is that the current relative layout between objects within a group G is preserved in case they want to restore a past location of G, making transitions such as the one in Figure 2.15-(i-j) very easy: the user can put back all desk elements to a past location while preserving the rearrangement of the elements that was made afterwards.

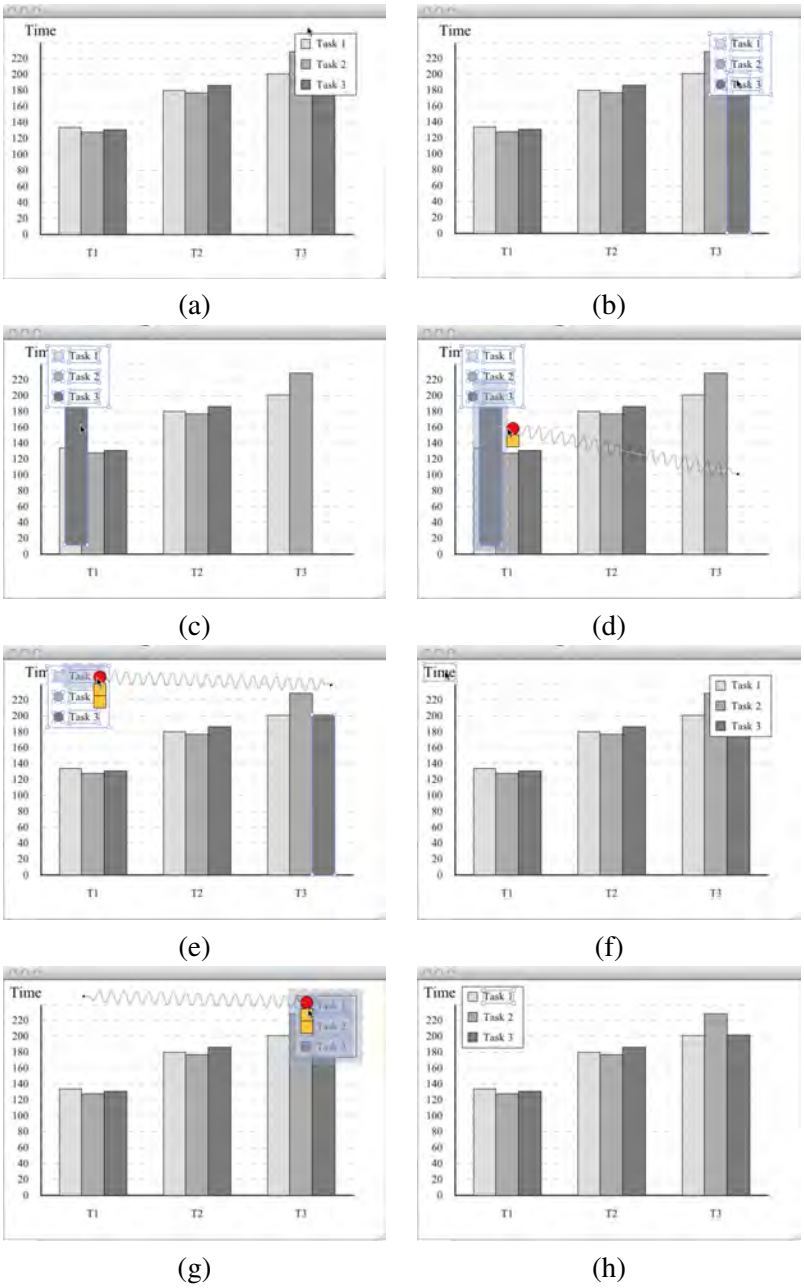


Figure 2.17 : Positioning the legend of a bar chart. (a-b) The user makes a selection of all elements of the legend. (c) He notices that he has accidentally included a bar in his selection and moved it along with the legend. (d) He uses Dwell-and-Spring to restore the bar to its original position. (e) The novel placement of the legend is not satisfying, and he puts the legend back where it was initially. (f) He moves the y-axis label, and (g-h) reverts the legend back to the left position.

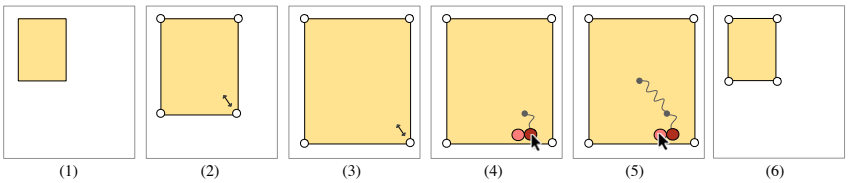


Figure 2.18 : DND^{-1} applied to shape manipulation handles. (1-2-3) The user resizes a rectangle twice. (4) She invokes Dwell-and-Spring on the resizing handle and enters the spring's main handle. This shows where the resizing handle was prior to the last resizing manipulation. (5) She moves the cursor to the next spring handle in the DND^{-1} history. This shows where the resizing handle was prior to the last two resizing manipulations. (6) She releases the mouse button on that second spring handle to revert back to the configuration in (1).

The second advantage is that it overcomes some limits of the *single active selection* model used in most applications. With the latter, the current selection gets cleared as soon as users click on a region or object that does not belong to the selection. DND^{-1} should save a lot of time and effort when trying to revert complex selections (small objects, objects scattered all over the workspace, partially occluded objects, etc.). For example, Figure 2.17 shows a scenario where a user wants to test alternative placements for the legend of a bar chart. The user accidentally selects a bar along with the legend, but notices it only after he has moved the legend to another location (a-c). As each object is also added individually in $History$, he can easily restore the bar's position (d), which has the effect of creating the group that contains only the graphical elements of the legend in $History$.

Navigating the DND^{-1} model

In order to navigate our DND^{-1} efficiently, we have extended the Dwell-and-Spring widget so as to make users able to invoke any reciprocal drag-and-drop on individual objects and groups of objects.

The first enhancement made to the Dwell-and-Spring technique is to provide users with extra spring handles that allow them to apply a series of reciprocal drag-and-drop actions quickly. As illustrated in Figure 2.18, the spring widget features additional handles that are horizontally aligned with the main spring handle (which was the only handle in the original design). Users can navigate through these handles to get a preview of where the selected object(s) would go if they selected them (by releasing the mouse button or lifting their finger). Selecting a handle invokes the series of reciprocal drag-and-drop actions that are associated with this handle. As explained above and illustrated in Figure 2.16, the series of past moves is managed by the DND^{-1} model.

The second enhancement made to Dwell-and-Spring is to provide users with

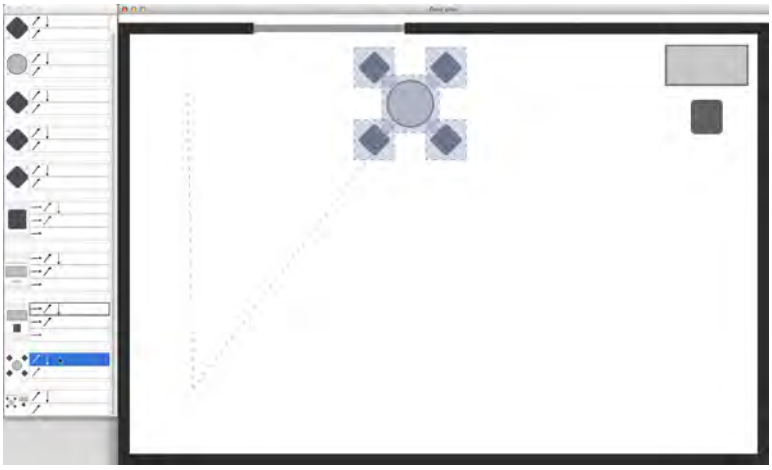


Figure 2.19 : The DnD-List technique.

another type of extra spring handles that allow them to apply reciprocal drag-and-drop actions to groups of objects that were moved simultaneously in the context of a multiple selection. These additional square handles act on the groups that contain object O, on which Dwell-and-Spring has been invoked. For example, in Figure 2.17-(g), the widget features three handles for the "Task 1" text label: a handle for the label itself (red circular handle), a handle for the group G_{legend} consisting of all legend elements, and another handle for the group corresponding to G_{legend} plus the bar that was accidentally moved in Figure 2.17-(c). Handles are organized into several rows, one per group. The primary handle of a row is aligned with the main spring handle that initially popped out. When the cursor enters the handle associated with a group G, additional handles appear on its left. There are as many handles as the number of locations this group has visited. To help users anticipate what will happen if they activate a given handle, Dwell-and-Spring gives some feedforward when entering that handle: objects in the group are highlighted, and the sequence of moves is shown as a series of springs that ends on what will become the center of object O.

We conducted an experiment to test if users could understand and use the DND^{-1} model effectively when they have to restore the position of either a single object or a group of objects. As the DND^{-1} model is novel, we also wanted to gather observations about its usability independently from its combination with Dwell-and-Spring. We thus designed a baseline technique that exposes the full history of DND^{-1} in a standard list presentation, close to the one found in, *e.g.*, Adode Photoshop. This technique, DnD-List, consists of a separate window, that remains always visible on top of other windows. As illustrated in Figure 2.19, this window shows a scrollable list. Each row displays the history of moves for a given group

of objects G. An image of G is on the left, and the list of paths that lead to all past locations of G is on the right. A path is displayed as a series of arrows whose orientation matches that of the actual movements that brought G where it currently is. Also, as for Dwell-and-Spring, when the mouse cursor hovers a path, a feed-forward is provided so as to show the bounding boxes of objects that belong to the associated group, along with a polyline showing what will become the center of this group if the user selects this path. The user can click on the path to actually execute the reciprocal drag-and-drop that reverts this series of movements.

In our experiment, participants had to both perform movements of objects using drag-and-drop operations and apply undo operations in order to reach a specific graphical layout. In the first part of the experiment, the difference between the target and the current layouts could be corrected with a reciprocal drag-and-drop on an individual object, whereas in the second part, solving the difference required performing a reciprocal drag-and-drop on a group of objects. In both cases, if users made an optimal use of the DND^{-1}, they were able to solve the difference with a single (optimal) reciprocal drag-and-drop.

Our empirical results suggest that the DND^{-1} model can effectively support undo for direct manipulations on both individual objects and multiple selections. This does not mean that participants understood all the details of DND^{-1}, but that they were able to use it effectively for the layout problem that they had to solve.

All participants expressed a preference for Dwell-and-Spring over DnD-List. This probably comes from the fact that DnD-List is a global technique that displays the whole history of all objects, consequently suffering from issues related to the difficulty in identifying the right manipulation in the history. By using an object as a reference to provide a contextual history in place, Dwell-and-Spring scales better with long and complex histories, and better fits with the per-object nature of the DND^{-1} model.

2.3 Conclusion

Point-based interfaces encode all information in graphical representations. It makes them efficient for both novice users who can interpret information without much prior knowledge of the interface, and expert users who can exploit their spatial memory [SCG13] to speed up frequent actions. However, actions become difficult from a motor perspective when the number of objects and movements become large. Visual interfaces can be augmented with non-visual feedback based on *e.g.*, auditory [Bre02] or tactile perception [BCB07] in order to facilitate small object acquisition, but both the display's physical size and humans' motor precision impose a lower bound to the size of graphical objects and to the precision of object movements. Interaction techniques that support *multi-scale navigation* and *fine-grained manipulation* can increase the expressive power of point-based interfaces by augmenting the number of objects that displays can accommodate and that users can manipulate.

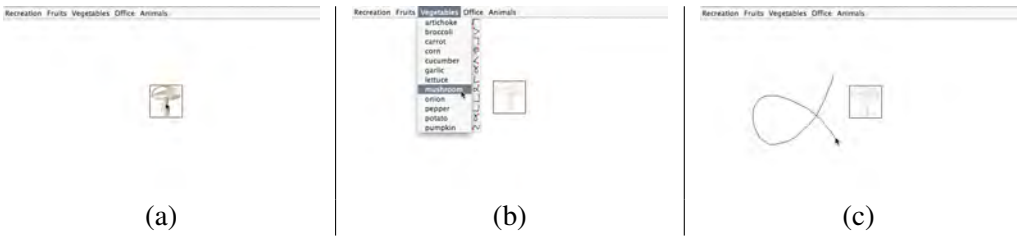
Gestures

Point-based interaction is heavily dependent on the graphical representation, meaning that increasing its expressive power consists in optimizing the use of the display area. However, no matter how good the optimization strategy for screen real-estate use is, the display resource remains limited. Furthermore, today's interactive systems include a large variety of displays, ranging from portable devices to wall-sized screens. Such systems feature graphical objects that are difficult to point at either because of their small size or because of their distant location. Gesture-based interaction then appears as a promising alternative or complement to point-based interaction, and has been my main research interest since my post-doc at IBM Almaden. In particular, I have studied how efficient surface gestures can be for invoking application commands, and how we could design sets of gestures that are highly expressive for small and/or multi-touch tactile surfaces.

Using gestures for invoking application commands is a priori effective to tighten the link between the expressivity of an interface and the amount of graphical objects it can accommodate. However, gesturing also comes with its own usability problems, such as the difficulty for users to discover and learn the available gestures, or their ability to reach a speed-accuracy trade-off for gesturing fast but precisely enough for the system to recognize their input traces. Designing efficient gesture-based interfaces demands both to understand what advantages they can offer over other input channels, and to implement support for users to help them discover the vocabulary of interaction and use it in an efficient way.

Making general statements about gestures is almost impossible given the wide variety of gestures. As mentioned in the introduction, gestures can involve users' whole body or only their finger tips; they can also be executed mid-air or on a surface; etc. Even when considering only surface gestures, *i.e.*, movements that are performed with fingers relative to a surface, the design space of possible gestures is theoretically infinite. However, in practice, the space is much more limited, both because users have limited cognitive and motor abilities, and because the robustness of a recognition system is usually inversely proportional to the number of gestures in the vocabulary. Providing large gesture sets that remain easy to learn and perform is a core challenge for making gesture-based interfaces usable.

This chapter reports on our empirical study that shows the cognitive advantages of gesture commands over more widespread accelerators such as keyboard shortcuts; and our engineering and design solutions to assist users during the discovery and learning phases of a gesture-based interface. It then presents our different projects about designing large gesture sets that can be used for manipulating multiple controls while relying only on simple-shaped gestures in order to limit the cognitive and motor load for memorizing and performing them.



(a) (b) (c)

Figure 3.1 : The task used in the experiment: (a) a command stimulus appears as an icon, the participant clicks on it (this makes the icon become semi-transparent) (b) the participant invokes the command through a menu, (c) or through a shortcut (a stroke shortcut in this case)

3.1 Gesturing for invoking commands

As a first investigation of the efficiency of gestures for invoking application commands, we ran an experiment comparing stroke shortcuts to the standard keyboard shortcuts [13]. We then proposed solutions for integrating gesture shortcuts with conventional widget-based interfaces, and for improving dynamic guides that support users in their adoption of a gesture-based interface.

3.1.1 Stroke shortcuts vs Keyboard shortcuts

Like using gestures to invoke commands, keyboard shortcuts have the advantage of saving screen space as they can activate commands without relying on a graphical widget. However, traditional keyboard shortcuts have their limits in many situations. First, studies show that users often have difficulty to transition from menu selection to keyboard shortcuts [LNPS05]. Second, keyboard shortcuts may not be convenient to use, particularly for portable devices such as smartphones that have no physical keyboard. Enabling users to efficiently trigger a command with a stroke gesture would overcome some of these problems.

Our experiment was motivated by the following hypothesis: stroke shortcuts may have a cognitive advantage in that they are easier to memorize than keyboard shortcuts. To better support recall, designers should make the shortcuts as analogous or mnemonic to the command name or meaning as possible. However, *arbitrary mappings* are unavoidable since many concepts in the digital world do not offer a direct metaphor to the physical world. Interestingly, because strokes are spatial and iconic, which makes richer and deeper processing possible in human memory [CL72] even if the mapping is arbitrary, we hypothesize that stroke shortcuts could have cognitive (learning and recall) advantages over keyboard shortcuts.

Our comparison to keyboard shortcuts was not meant to be a competition, but rather to use keyboard shortcuts as a baseline control condition. Since the use of shortcuts largely depends on their ease of learning, we focused our study on learning aspects involved in both types of shortcuts. We also limited our study to

ICON	Keys	Stroke	ICON	Keys	Stroke
	Shift+W			Ctrl+W	
	Shift+D			Ctrl+D	
...	...	...	...	...	...

Figure 3.2 : An excerpt of the mappings used in the experiment.

the general case of arbitrary mappings between the commands and the shortcuts, namely mappings without direct mnemonic association in either condition. This decision was based on several considerations. First, a learning experiment takes time to do well even when it is focused. Second, the special cases of mnemonic mapping, which should be maximized in actual design, is rather limited in number. For example the usual way of making a keyboard shortcut mnemonic is to use the first letter of the command name. However this rule makes interface developers quickly run into conflicts: in fact the small set of five common commands {Cut, Copy, Paste, Save and Print} already exhibits two conflicts. Also, for non-English speakers, the same command may have different names in different languages yet it has the same keyboard shortcut (which is probably a reasonable design choice for consistency). Third, stroke shortcuts can always be made as mnemonics as keyboard shortcuts by choosing letter-shaped strokes. Learning required in that case is probably limited.

Experiment design

We modeled our experimental task after Grossman et al. [GDB07] which was the most recent and most complete study to date on learning keyboard shortcuts. The task required the participants to activate a set of commands that were accessible through both menus and shortcuts. Once a command stimulus (*i.e.*,a graphical icon, as in [GDB07]) was displayed in the center of the screen, the participant was asked to first click on the icon (Figure 3.1-(a)) and then execute a corresponding command as quickly as possible through either menu selection (Figure 3.1-(b)) or a shortcut activation (by drawing a stroke or pressing hot keys, depending on the experimental condition) (Figure 3.1-(c)). The click on the icon at the beginning of each trial prevented the participant from keeping the mouse cursor in the menu area to only interact with menu items. Both types of shortcuts were displayed *on-line* beside the corresponding menu items. The participant was explicitly told to learn as many shortcuts as possible. In case he did not know or remember a shortcut, he could use the menu to directly select the command or look at the shortcut.

The keyboard shortcuts were assigned in accordance with the rule used in [GDB07]: they were composed of a sequence of a modifier key followed by an alphabetic key that was not the first or last letter of the name of the object. To reflect a necessary difficulty in practical keyboard assignments, the same alphabetic key preceded by two different modifier keys (Ctrl or Shift) constituted two different

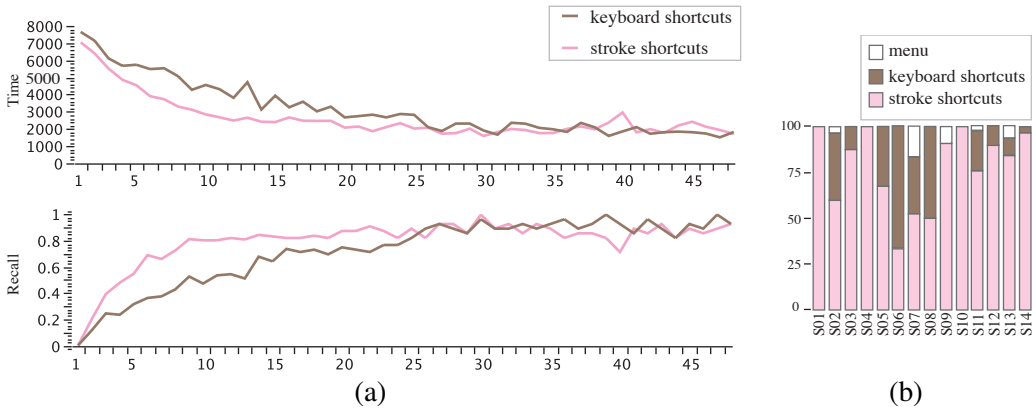


Figure 3.3 : (a) Time and Recall performance according to the number of times a command has been seen in the *test* session - (b) percentage of use of each technique by participant in the *retest* session

commands. To reduce a potential bias, we reproduced this potential pair confusion in stroke shortcuts as well: the same shaped stroke with two different orientations activated two different commands. Table 3.2 shows a sample of the icons and the two types of shortcut we tested. To minimize the influence of the participants' personal experience, commands tested were not those in common software applications but rather objects and activities of everyday life organized into five menus (categories): Animals, Fruits, Office, Recreation and Vegetables. Each menu contained 12 menu items resulting in a total of 60 items. In order to have enough trial repetitions, the participants had to activate a subset of 14 commands during the experiment. Note that the rest of the 60 items were also assigned shortcuts and served as distractors both to the participants and to the stroke recognizer. Finally, to reflect the fact that some commands are invoked more frequently than others in real applications, we assigned different frequencies to different commands.

Because we were interested in memorization aspects, it was important to operationalize the case when users return to an application after a period of not using it. Our experiment consisted of two sessions on two consecutive days. On the first day, participants had to perform a *warm-up* session where the only way of invoking a command was through menu selection, followed by a *test* session where commands could be invoked through either menu selection or shortcuts (they were presented to both stroke and gesture shortcuts condition in a counter-balanced order). On the second day, they participated to a *re-test* session where both types of shortcuts were available and the participants were told to use what was most convenient for each trial. At the end of the experiment, they were given a questionnaire in order to collect qualitative feedback.

Results

In the *test* session on the first day, participants used shortcuts in 96% of the trials when the application was featuring stroke shortcuts. The percentage dropped to 88.5% when the shortcuts were standard keyboard shortcuts. The recall score was significantly higher with stroke shortcuts than it was with keyboard shortcuts. Trials were also completed faster on average with stroke shortcuts than they were with keyboard shortcuts. Finally, our participants made significantly fewer errors with stroke shortcuts than with keyboard shortcuts. Figure 3.3-(a) illustrates the main phenomenon: participants benefited from the use of stroke shortcuts when they were *learning* the mappings. The figure plots the mean $Time$ and $Recall$ performance as a function of the number of times an item was tested, clearly showing that $Time$ decreased faster and $Recall$ accuracy increased faster with stroke shortcuts than with keyboard shortcuts. Note that the performance difference between the two types of shortcuts is primarily cognitive (learning and recalling the shortcuts). With enough practice, when user performance is more likely to be limited by motor execution (around the 25th exposure in this experiment), the difference in both time and recall between the two types of shortcuts becomes indistinguishable.

Data collected in the *re-test* blocks on the second day allowed us to evaluate users' memory retention of the shortcuts learned, and to see which type of shortcuts they preferred. Figure 3.3-(b) shows the percentage of use for each technique ($Keyboard$, $Stroke$ and $Menu$) per participant. Although varied by individual, on average, significantly more stroke shortcuts than keyboard shortcuts were used.

The participants' open remarks confirmed our hypothesis about the power of strokes for encoding mnemonics. Strokes gave participants richer clues to make up an association (more levels of processing) between a command and its arbitrarily assigned stroke. Examples of participant quotes were "I thought of this stroke as fish because the shape's stroke makes me think about a basin" or "I associated this stroke with a jump and I see karate as a sport where people jump". Interestingly, no two people mentioned the same trick to associate a stroke with a command.

In summary, although the purpose of stroke shortcuts is not to replace or compete against either menu selection or keyboard shortcuts, the experiment clearly shows that stroke shortcuts can be as efficient as, or more advantageous than, keyboard shortcuts. While both types of shortcuts have the same level of performance with enough practice, stroke shortcuts have substantial cognitive advantages in terms of learning and recall. With the same amount of practice, users can successfully recall more shortcuts, and make fewer errors with stroke shortcuts than with keyboard shortcuts.

3.1.2 Discovering and learning stroke shortcuts

If using strokes to activate commands looks promising, it also has the well-known and important drawback that strokes are not self-revealing [KM94, HZS$^+$07, BM08]. In other words, as opposed to buttons and menus, users cannot guess which

(a) (b) (c)

Figure 3.4 : Revealing stroke shortcuts (a) *Tooltip*, (b) with *Menu preview* and (c) with a *Strokes Sheet*

stroke-based commands are available and which stroke triggers which command. Often, novel features of an interface are unused not because they are difficult to use, but because users are not aware of them. Therefore, interfaces should offer visual clues about available strokes to enable end users to discover and learn their effect.

Revealing stroke shortcuts using standard widgets

To address the *visibility* problem (*i.e.*, users do not have a way to discover the available strokes and their meaning), we proposed the Stroke Shortcuts Toolkit (SST) that allows developers to implement stroke shortcuts in a wide range of software applications [13]. SST requires only a few lines of code to add stroke shortcuts in any Java Swing application. In particular, with a single line, developers can turn on three types of visual clues (*Tooltip*, *Menu preview* and *Strokes Sheet*). If *Tooltip* is turned on, any graphical component that provides a shortcut will display it in a tooltip that pops up when the mouse cursor dwells over this component. If this component already has a tooltip associated with it, the existing tooltip is augmented with the stroke illustration while preserving its original text (Figure 3.4-(a)). Similarly, if the *Menu preview* is turned on, any menu item that is invokable by a stroke displays a preview of this stroke beside its label, as is usually done with keyboard shortcuts (Figure 3.4-(b)). Finally, a *Strokes Sheet* can be enabled (Figure 3.4-(c)). It is an independent window that displays the list of shortcuts and the name of their associated commands found in the current opened windows. The behavior of the strokes sheet was inspired by the Tivoli system [KM94]: this sheet pops up each time the user pauses during a stroke (at the beginning, or at any moment while stroking) and remains visible until the user enters a shortcut that is successfully recognized, or closes the sheet.

Revealing stroke shortcuts using contextual guides

The mechanisms for making stroke shortcuts visible described above have the advantage of being easy to plug in an existing WIMP-based application. However, they also have important drawbacks such as consuming screen space (*Strokes*

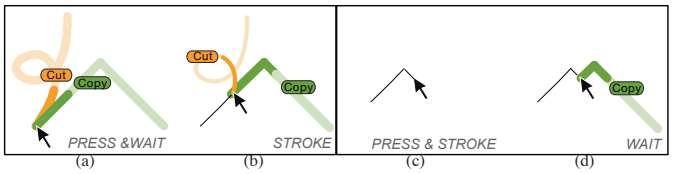


Figure 3.5 : (a-b) OctoPocus in novice mode (tracing *copy* causes *cut* to get thinner). (c-d) OctoPocus in intermediate mode (*cut* disappeared, and there is a scale mismatch for *copy*)

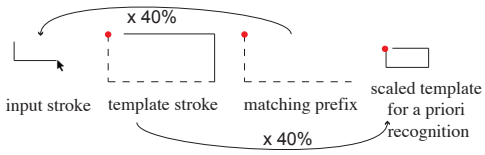


Figure 3.6 : Partial input recognition and scale ratio estimation.

Sheet) or requiring extra actions from users to access visual clues (*Menu preview* and *Tooltip*). Contextual guides, such as OctoPocus [BM08], offer a better user experience as their support is dynamic and integrated into users' gesturing actions. The guide is dynamically updated, as the input grows, to show only the subset of gestures that match the partial input. However, implementing such guides in a robust way is difficult as it demands to incrementally recognize the gesture as the user performs it. Figure 3.5-(c, d) shows what can happen if the incremental recognition process does not handle the variability that exists in different versions of the same gesture. Here, the problem is that partial input recognition is scale dependent while most gesture recognizers support scale independence (*i.e.*, the same shape at different scales actually invokes the same command).

In [3], we propose an algorithm to address this scale mismatch issue. Fig. 3.6 illustrates what our algorithm is capable of: identifying the matching prefix between users' partial input and a gesture template, and computing the scale ratio between both. Our solution is inspired by the "turning angles representation" algorithm used in image analysis [NY95], which has the advantage of representing a shape as a vector of *turning angles*. This representation is actually scale-independent. In this algorithm, the shape is sampled at a given number of equally spaced points to obtain a series of subsegments. A subsegment is defined by the angle it forms with a reference axis, *e.g.*, the x-axis. The distance between two shapes is simply computed as the distance between the two vectors of turning angles. In the case of partial input recognition, users provide only a prefix of the final gesture, making the sampling in equally spaced points irrelevant for comparing an input to a template. Our algorithm addresses this issue.

Given an input stroke S_{input} and a set of n templates S_{t1} ... S_{tn}, our algorithm consists of the following steps:

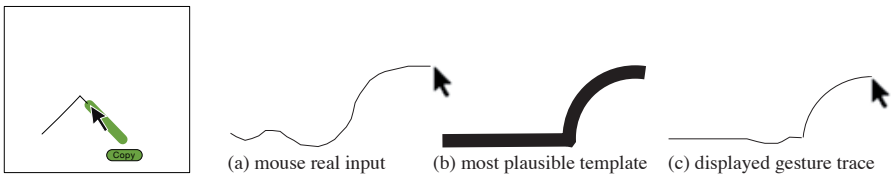


Figure 3.7 : (Left) Rescaling the gesture candidate - (Right) Attracting users' input: (a) Raw input (b) Template (c) Actual gesture trace.

Step 1: Compute a *scale independent representation* (SIR) for S_{input} and all S_{t1} ... S_{tn}. A SIR can be seen as a coarse turning angle representation: it aggregates the subsegments that do not significantly vary in the angle they form with the x-axis. The SIR is thus a series of segments defined by their length (*i.e.*, the sum of aggregated lengths) and their angle (*i.e.*, the mean of aggregated angles).

Step 2: SIRs are convenient to look for the prefix that matches S_{input} in each template S_{t1} ... S_{tn} by simply comparing the successive segments. Two segments are similar if their difference in terms of angle is lower than TOLERANCE[1]. If a segment $SegS_{input}$ from S_{input} is not similar to a segment from a template, $SegS_{input}$ length is recorded as non matching. When the consecutive non matching length exceeds 10% of the length of S_{input}, the compared template is discarded. However a non matching part lower than that threshold is ignored to avoid discarding plausible candidates just because of a small noisy portion. Note that this tolerance is applied only on the S_{input} side and not on the template side, to avoid ignoring angle changes on small portions that are explicitely part of a template (*e.g.*, a curly brace stroke). Our algorithm thus considers templates as *perfect strokes*.

Step 3: Once the matching prefix (if any) is identified for each template, the scale ratio between the input and a matching prefix is computed as the mean of the ratios between lengths of matching pairs of segments.

The algorithm described above allowed us to implement an augmented version of the OctoPocus guide. First, the missing part of gesture candidates is rescaled using the proper ratio (Figure 3.7-(Left)). Second, we have implemented a magnetic effect to facilitate gesturing. We "dig ditches" along the most probable template trajectories in motor space, in the spirit of kinematic templates [FLTL08]. The width and depth of each ditch are proportional to the template's plausibility given the current partial input: the more probable a template, the larger and the deeper its ditch. To provide a smooth and continuous behavior, ditch depth is maximal in the center and progressively decreases towards the boundaries. Users have more control than with a binary snapping mechanism, enabling them to deviate from the most probable trajectory to draw another gesture (in the spirit of our RouteLens technique presented in the previous chapter). Figure 3.7-(Right) shows an interest-

[1] In practice, we use TOLERANCE=$\frac{\pi}{4}$.

ing side effect of this magnetism: the stroke gets *beautified* as it is drawn. Not only may this inform the user that he can draw more quickly since the trace looks good, but it also provides a pleasant aesthetic experience.

We conducted an experiment to test if our augmented guide improves users' experience. Participants had to gesture with a mouse in two conditions: assisted by a regular OctoPocus guide or by an augmented version of it. We measured the completion time and the distance between participants' input and the correct gesture template. Collected data showed that our augmented guide helped participants to perform gestures that are better recognized (because closer to the correct template) without increasing drawing time.

3.2 Gesturing on portable devices

Using gesture-based interaction appears, at first, as an excellent fit to portable devices which have a small screen that can display a limited number of widgets. However, the results presented above, which were conducted in the context of a desktop station operated with a mouse or of a tablet PC operated with a stylus, are not directly relevant to devices that support touch-based input and that are used in a mobile context. On such devices, slide and pinch gestures are already dedicated to navigation, meaning that using other gestures for invoking commands would require to resort to a mode switching action. Furthermore, these devices are often operated in a one-handed context where only the thumb can move on screen with a limited range of motion. The gestures should then remain simple if we want users to be able to perform them in such contexts. In this section, we present three projects that tackle these issues: *SidePress* that augments a device with pressure sensors on its side to allow users to, *e.g.*, delocalize the navigation controls on the device's edge or easily switch between modes; the *Power-up button* that can track simple thumb mid-air gestures that are performed on the side of the device; and *TilTouch* gestures that combine finger sliding with device tilting in order to avoid collision with slide gestures that are used for navigation.

3.2.1 SidePress

SidePress [24], illustrated in Figure 3.8, augments a mobile device with two continuous pressure sensors, co-located on one of its sides. Our prototype consists of three components: an iPod touch 4G (iOS 5.0), a custom-designed plastic casing that hosts two force-sensitive resistors (Interlink Electronics FSR 400), and a custom-built circuit board. Figure 3.8 illustrates a right-handed user applying pressure with either the two groups of digits <index, middle> and <ring, little>, or with her thumb and palm.

Thanks to its two sensors, SidePress provides bi-directional navigation capabilities at different levels of granularity, all seamlessly integrated: continuous

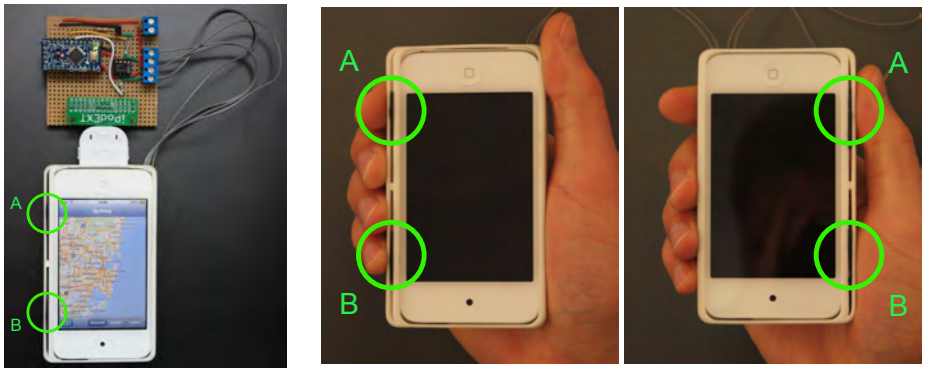


Figure 3.8 : The SidePress prototype has two pressure sensors on one of its sides, providing a rich input vocabulary for interaction. Pressure Sensors can be positioned on either side of the device.

rate-based control (pressure intensity), jumps of different amplitudes (*light-click* or *strong-click*), jump to the minimum or maximum values in the value's range (*strong-press*). Altogether, these events provide users with the same level of expressiveness as a traditional scrollbar, but without cluttering the screen, and without causing visual occlusion or interfering with touch input, as control gets delegated to sensors outside the display area. Users can also press both sensors simultaneously, an additional event that can be useful for, e.g, holding a call, switching between navigation and editing modes, or displaying a control panel to invoke a command or to jump to a bookmark.

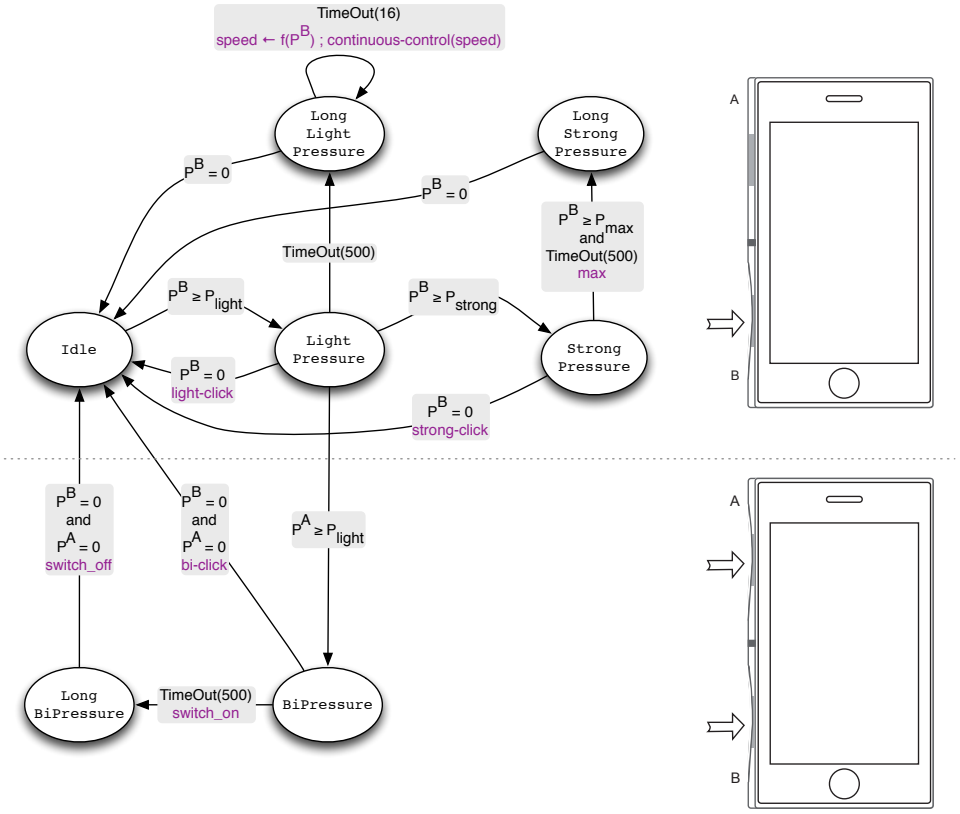
Interaction events

The hardware prototype is driven by software that translates variations of both pressure sensor values into interaction events. Figure 3.9 illustrates the state machine corresponding to one sensor. Two such machines run in parallel, one for each of the two sensors A and B. Whenever the user actuates one of the sensors, *i.e.*, as soon as one of the machines leaves the `Idle` state, the other machine gets disabled.

SidePress is particularly well-suited for navigating large collections of items, such as the pages of a long text document, collections of pictures, a feature film or any other long video. It can also be useful when precisely adjusting the value controlled by any kind of slider, or positioning the caret in a text document. In the following, we describe the interaction techniques enabled by these state machines using a simple example: navigating a document.

When navigating a document, the two sensors provide a set of actions seamlessly integrated together to move upward (sensor A) and downward (sensor B). When the user starts pressing one of the sensors, *e.g.*, sensor B, the corresponding machine goes into the `LightPressure` state and waits a short amount of time (500ms) for one of the following events to occur depending on what the user inputs.

Two state machines run in parallel, one for the upper sensor (P^A) and one for the lower sensor (P^B). Whenever a machine leaves the Idle state, it disables the other machine, which gets re-enabled when the former gets back to the Idle state.



$P_{light} = 1$, $P_{strong} = 5$ and $P_{max} = 6$ (approximate force in Newton)

Figure 3.9 : State machine for sensor B.

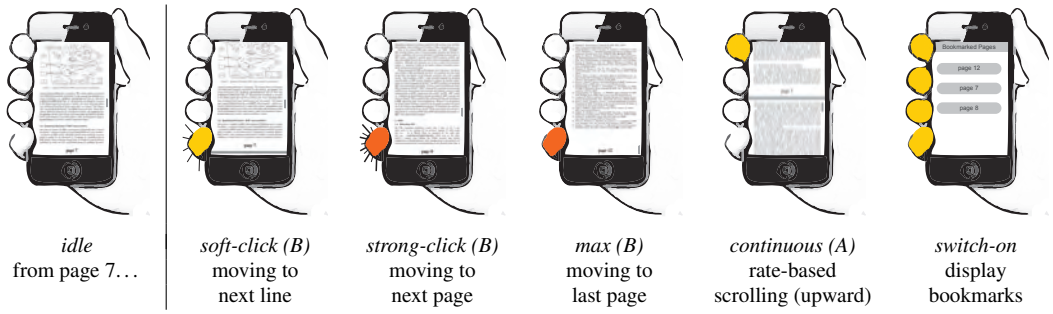


Figure 3.10 : Application to navigation in a document

- She can release sensor B. This takes the machine back in the `Idle` state, and triggers a *light-click* event that translates the document downward by one line.
- She can apply a stronger level of pressure on sensor B. This takes the machine in the `StrongPressure` state. From there, either she releases the sensor within 500ms, which triggers a *strong-click* event that takes her to the next page; or she keeps the sensor pressed for more than 500ms, which triggers a *max* event that takes her to the last page of the document.
- She can press sensor A (in addition to B). This takes the machine in the `BiPressure` state. From there, either she stops applying pressure within 500ms, which triggers a *bi-click* event that bookmarks the current position in the document; or she keeps them pressed for more than 500ms, which triggers a *switch-on* event that toggles another mode and takes the machine in the `LongBiPressure` state until the sensors get released (*switch-off*). In our example, toggling to the other mode pops-up a list of previously-bookmarked locations in the document, that can be selected, *e.g.*, using direct touch or by tilting the device.

If none of the above three events happen within the first 500ms after the user started pressing sensor B, the machine enters state `LongLightPressure`, which allows for pressure-dependent *continuous* control. In our example, this translates to rate-based continuous downward scrolling.

The same events performed on sensor A translate to upward navigation actions in the document: up by one line, previous page, first page, continuous rate-based upward scrolling.

Users can perform additional actions by actuating both sensors simultaneously. Pressing both sensors at the same time consists in squeezing the device, and is a relatively natural gesture. The gesture is interpreted either as a *bi-click* if users release the sensors less than 500ms after press, or as a mode switch if they keep them pressed for a longer period, triggering a *switch-on/off* (lower part of Figure 3.9). These two events can be associated with the invocation of a command, like bookmarking or undoing an action, or entering/leaving another mode, respectively.

Evaluation

We ran a first experiment to evaluate users' ability to trigger, on demand, each of the ten events described above. A trial consisted in triggering one of the events. Because simply displaying the name of the event as a stimulus would have been too artificial and cognitively demanding, we conveyed the stimulus information using a *tank filling* metaphor. A *light-click* adds (A) or removes (B) one volume unit. A *strong-click* sets the volume to the closest upper tick (A) or to the closest lower tick (B). A *max* event fills (A) or empties (B) the tank. A *switch-on/off* makes the tank

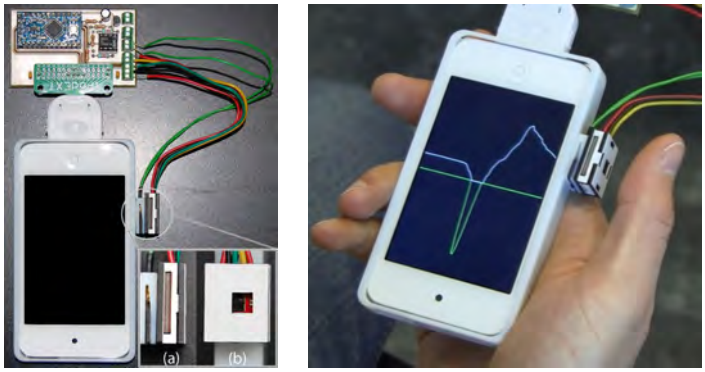


Figure 3.11 : (Left) Our prototype equipped with a Power-up button. The sensors are placed inside custom-built plastic casings. (a) Side view of the pressure and proximity sensors. (b) Front view of the window above the proximity sensor. (Right) Users can perform mid-air and pressure gestures using their thumb.

disappear; the participant then has to wait for a Release instruction that pops up 2 seconds later.

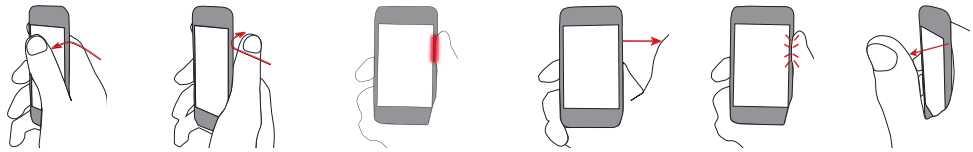
This first empirical evaluation showed that side pressure input is a very promising modality. Participants were able to control the whole vocabulary of events with an overall success rate of 90%, most errors being low-cost or costless ones (for example, performing a *continuous*(B) instead of a *light-click*(B)).

The second experiment tested how users can take benefit of such a new vocabulary for tasks that mainly involve unidimensional or bi-directional navigation, such as scrolling a document or finding a scene in a video. We compared SidePress with the standard interaction technique for one-handed scrolling: drag and flick touch gestures. The experimental task consisted in scrolling a 20-page document (page length = screen height = 920 pixels) to bring a target that was initially out-of-screen inside the viewport.

The most interesting observation was the performance difference between the two techniques that depended on target distance: *Touch* is faster than *SidePress* for small distances, and conversely *SidePress* is faster than *Touch* for long distances.

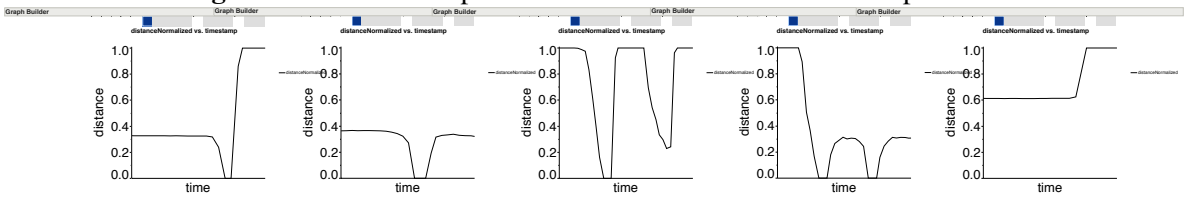
3.2.2 Power-up button

The Power-up Button [23] consists of one proximity sensor superimposed on one continuous pressure sensor. Located on the side of a mobile device, it enables gestural interaction with the thumb that holds the device. Figure 3.11 shows our prototype. The sensing hardware comprises four components: a force-sensitive resistor (Interlink Electronics FSR400 Short), an infrared proximity sensor (Vishay VCNL4000) that is mounted on a Sparkfun breakout board, an Arduino Pro Mini,



(a)	(b)	(c)	(d)	(e)	(f)
discreteDown (proximity)	*discreteUp* (proximity)	*continuousDown* (pressure)	*continuousUp* (proximity)	*click* (pressure)	*quickRelease* (proximity)

Figure 3.12 : Gestures performed on and around the Power-up Button.



(a) *discreteDown* (b) *discreteUp* (c) *discreteDown*$^+$ (d) *discreteUp*$^+$ (e) *quickRelease*

Figure 3.13 : Proximity profiles for discrete events (time scale = 1 sec). *discreteDown*$^+$ and *discreteUp*$^+$ show two consecutive gestures (time scale = 1.5 sec).

and a custom-designed printed circuit board.

Our Power-up Button enables users to perform numerous actions using a single controller that does not occupy more space than a regular button would, but that provides a much higher level of expressive power. Combined with a gesture recognizer that takes the hand's anatomy into account, it allows users to trigger a set of four discrete events and two continuous events using simple gestures performed on and around the button while operating the device with a single-hand (Figure 3.12). A single Power-up Button enables users to control, *e.g.*, a music player eyes-free: pause and resume playback, adjust volume settings, and navigate both within and across songs.

Interaction events

To demonstrate the potential of the Power-up Button, we designed a set of gestures that can be discriminated using the technology described above. We considered both motor and physiological aspects, ending up with a set of gestures that are easy to perform and that will not cause too much fatigue even when performed in sequence. To control the Power-up Button, users move their thumb within the device's plane towards or away from the button (Figure 3.12-(c,d,e)), and out of the device's plane towards either the front or the rear of the device (Figure 3.12-(a,b,f)). We implemented a recognizer to discriminate between these two-dimensional gestures with one-dimensional sensing capabilities only.

Our recognizer is implemented as a finite state machine in which final states are one of the following interaction events: *discreteDown*, *discreteUp*, *click*, *quickRelease*, *continuousDown* and *continuousUp*. Transitions are triggered by input variations on the pressure and proximity sensors. For the two continuous events, a first-order control is initiated as soon as the corresponding state is reached, letting users adjust the rate by varying the pressure intensity or the distance to the button.

Events and Pressure To recognize *click* and *continuousDown* events, the Power-up Button analyzes the profile of the normalized pressure intensity that the thumb applies to the device. When the user starts pressing the force sensor, the state machine waits for a short amount of time (500 ms) to decide if the pressure is released or sustained. Releasing the pressure before the timeout occurs triggers a *click* event if the initial force exceeded 1N. If the pressure is sustained, the state machine triggers *continuousDown*, enabling pressure-dependent *continuous* control.

Events and Proximity The Power-up Button analyzes the profile of the normalized distance between the device and the thumb. The distance is $d = 1$ if the thumb is too far away from the sensor's detection zone (`OutOfRange`), $d = 0$ if the sensor is occluded (`Touch`), and $d \in]0..1[$ if the thumb is close to the button (`Prox`). In addition, our algorithm examines how the distance changes over time ($\Delta d = d_t - d_{t-1}$, uniform sampling frequency) to detect changes in direction ($\Delta d > 0$ and $\Delta d < 0$) and gesture speed.

This low-level input allows our recognizer to identify the *discreteDown*, *discreteUp*, *quickRelease* and *continuousUp* gestures. The main challenge lies in discriminating *orthogonal* gestures *discreteDown* and *discreteUp*, as the proximity sensor only captures the orthogonal distance between the thumb and the edge of the device. We call these gestures orthogonal because the thumb is leaving the device's plane either towards the front of the device (*discreteDown*) or towards the rear of the device (*discreteUp*). Figure 3.13-(a,b) shows typical sensor readings for these gestures. Both gestures must begin in the proximity range (`Prox`), touch the button (`Touch`) and leave the device's plane in one direction or the other. This final part of the movement can be discriminated because of the anatomy of the hand, that makes moving the thumb away from the user (adduction) harder than moving it towards her (abduction). This results in sensing `OutOfRange` for *discreteDown* gestures and `Prox` for *discreteUp* gestures.

The recognizer for *discreteDown* and *discreteUp* accepts several profiles for a single gesture, making the repetition of such gestures in clockwise and counter-clockwise directions easier to perform. The initial position for both gestures becomes optional, allowing users to start their gesture either in proximity range `Prox` (Figure 3.13-(a,b)) or `OutOfRange` (Figure 3.13-(c,d). For *discreteDown*, users can either touch the button or pass close enough to it in mid-air (Figure 3.13-(c)).

Figure 3.13-e shows that the *quickRelease* gesture is easily recognized based on a speed threshold at which the thumb must leave the $\texttt{Prox}_{(\Delta d=0)}$ without touching

the button ($\Delta d > s$ with $s = 0.5 \cdot (1 - d_{t_{initial}})$). Finally, the *continuousUp* gesture is recognized when the thumb remains in the detection zone for more than 500 ms.

Expressive power

To evaluate the expressive power of our button, we have demonstrated how the set of interaction events it generates can be used to control any widget of an existing graphical application without touching the screen. Avoiding touch input implies dedicating interaction events to give the focus to a given widget. We used the (*click*, *quickRelease*) couple of events to navigate the widget hierarchy in depth, and (*discreteUp*, *discreteDown*) to navigate it in breadth. Once at the deepest level of the hierarchy, the four events (*discreteUp*, *discreteDown*, *continuousUp* and *continuousDown*) are used to set the value of the widget that owns the focus.

The exercise that consists of controlling all widgets is a demonstration of what the Power-up Button could do, pushed to the extreme. The more relevant use cases we envision involve a smaller set of controls like, *e.g.*, discarding phone calls or controlling a music player. The Power-up Button can also complement touch input, and can be particularly useful when interacting eyes-free. It also opens up a larger design space for widget organization on screen: the button enables a more compact layout of interface components than what touch input alone would allow. This can be useful when, *e.g.*, filling the numerous fields of a long Web form, or for very small devices.

3.2.3 TilTouch

The last contribution about gesturing on portable devices that we present in this section does not require any additional electronics. TilTouch gestures [25] extend the vocabulary of navigation interfaces by combining motion tilt with directional touch, and can be implemented using the device's built-in gyroscope. TilTouch gestures do not interfere with regular touch and tilt actions, and can thus be used to invoke commands without resorting to explicit mode-switching actions. Our approach lies within the Sensor Synaesthesia design space [HS11], but we focus on directional drags rather than static touch.

Combining touch with tilt

Our goal is to facilitate command invocation while users perform navigation tasks. By combining drags and tilts, we can enhance the number of available gestures but also avoid the use of explicit mode-switching. To that end, we explored how a TilTouch gesture should be defined such that it does not interfere with existing navigation gestures. We conducted a preliminary study to determine the angular range of unintentional tilts that occur during common drag and swipe actions. 10 participants interacted with three representative mobile interfaces: a large

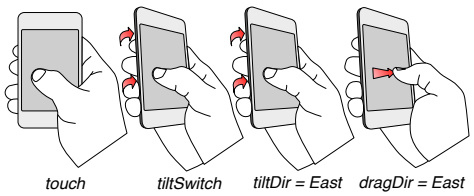


Figure 3.14 : $\langle tiltDir, dragDir \rangle = \langle East, East \rangle$ gesture performed with one hand. Here, the user starts with a tilt followed by a touch action.

2D view, a vertical list of 50 items, and a horizontally aligned view of five screens. We analyzed a total of 3935 events of navigation or directional-drag gestures and defined unintentional *relative tilt* angles to be in the range of $[-18, 18]$ degrees, where tilts are measured relative to the initial orientation of the phone when the finger starts touching the screen. This range corresponds to less than 0.2% of false positives and does not hinder the activation of intentional TilTouch gestures. We also set the minimum drag distance for activating a gesture to 7 mm. When these criteria are met, we detect the dominant direction of the tilt ($tiltDir$) and the drag ($dragDir$) to recognize the TilTouch gesture $\langle tiltDir, dragDir \rangle$. The command associated with this combination is executed after lifting the finger.

Our goal was to keep the shape of gestures as simple as possible while providing a reasonable number of accessible commands. Therefore, we only consider the simplest touch marks and tilt angles along the four cardinal directions: $North$, $South$, $East$, and $West$. Combining such simple gestures along the touch and tilt modality results in a total of 16 commands. Figure 3.14 illustrates an example where the user performs an $\langle East, East \rangle$ gesture.

TilTouch gestures are especially useful as quick triggers of commands, *e.g.*, copy-paste, make a call, and add a bookmark on a map. The technique requires the sequential or parallel control of two input modalities. We envision that novice users will start discovering the gestures by controlling tilt and touch in sequence. However, we expect that with practice, expert users will be able to internalize a single action and use the two input channels in parallel, combining the two gestures into a single interaction chunk [Bux95]. As other directional gestures, TilTouch gestures have drawbacks: drags are limited near screen edges, and screen visibility is reduced both by the finger and the tilt. Yet, our low threshold value of relative tilts ($\pm 18^{\circ}$) minimizes this problem.

Evaluation

We conducted a laboratory study where participants performed TilTouch gestures in conjunction with regular drags. Our goals were to find: (1) which TilTouch gestures are more effective; (2) whether the same gestures are appropriate for both single and two-handed use; and (3) whether users can control the two input modal-

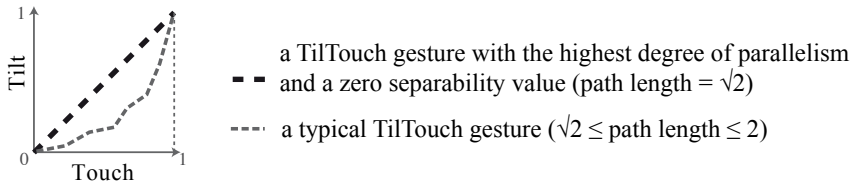


Figure 3.15 : The normalized TilTouch space.

ities in parallel.

The primary factors of our experiment were the *type of gesture* and the *mobile use*. For the *type of gesture*, we considered TilTouch gestures where participants were only instructed about the direction of the drag gesture and were free to combine with a tilt gesture in any direction; and TilTouch gestures where both drag and tilt directions were imposed. For the *mobile use*, we considered both one- and two-handed contexts of use. When using only one hand, participants had to drag and tilt with their dominant hand, while they were tilting with their non-dominant hand and dragging with their dominant hand when using two hands.

Results reported in [25] details what the best combinations of TilTouch gestures were in our experiment, the take-away message being that constraining tilt direction has a great cost to user performance. Combinations with same tilt and drag directions have the best performance. This is especially true for one-handed use. And, when participants are free to tilt in the direction of their choice, most of the TilTouch gestures are a combination of a drag and a tilt in the same direction.

We were also interested in observing whether users are able to control touch and tilt in an integral way [JSMM94] when performing TilTouch gestures, as it would suggest that they perceive a TilTouch gesture as a single interaction chunk [Bux95]. We analyzed the degree of parallelism of the tilt and touch input channels by plotting the collected TilTouch gestures in the normalized tilt+touch space shown in Figure 3.15. The path length of a TilTouch gesture exhibiting the highest degree of parallelism is $\sqrt{2}$ while a TilTouch gesture with no parallel control has a length of 2. By normalizing the path length of each TilTouch gesture, we obtain the *Separability* measure $\in [0, 1]$: the less parallel the control over the two input channels the higher the value of *Separability*. We found a positive correlation between mean *Separability* and mean completion time, with remarkably low separability values for tilts and drags of the same direction for both one- and two-handed contexts.

3.3 Gesturing with multiple fingers

Most tactile screens also support multi-finger input. While using several fingers concurrently quickly becomes uncomfortable on a small device such as a

smartphone, multi-touch offers a rich input channel for tablet-sized screens and tabletops. However, current systems do not use it at its full potential. They make an extensive use of single-finger slides and two-finger pinches for viewport navigation, but the use of other multi-touch gestures remains anecdotal. In the end, interaction still heavily relies on graphical widgets that interpret only single touch input. Some projects propose hand gestures for interacting with tabletops that are based on diffuse illumination technology (*e.g.*, [CWB+08, WBP+11]). But, apart from the specific case of chording gestures (*e.g.*, [BML12, GHB+13]), there is very little work about what can be done with the basic multi-touch API that most tactile screens only support.

This section presents two projects that advocate for a better use of multi-touch input. The first one [18] empirically shows that there is a planning effect when manipulating virtual objects on a multi-touch surface, this effect affecting how users initially grasp objects. This preliminary result suggests that we could possibly anticipate people's intentions as soon as they grab an object, before its actual manipulation starts. The second one [17] presents a large design space for multi-touch gestures that are easy to perform, and a recognition engine to accurately discriminate them. In this space, interface designers can identify a set of gestures to provide users with a large number of both discrete and continuous controls.

3.3.1 Effect of planning on multi-touch grasps

The manipulation of virtual objects has a central role in interaction with tabletops. For example, users move and rotate documents and pictures around the surface to share them with other users. Graphical designers manipulate information and graphical objects to create new content. Multiple users work collaboratively to create schedules, make decisions, or solve complex problems. In all these scenarios, users interact with their hands and their fingers; they grasp, translate, and rotate virtual documents as they would do with physical objects.

Researchers in Psychology have studied how people manipulate objects in the physical world. This work has unveiled that people show strong signs of prospective motor planning, *i.e.*, they choose initial grasps that avoid uncomfortable end postures and facilitate object manipulation. A very recent model that considered continuous tasks, such as rotating a physical knob, has been proposed by Herbort [HB12]. This model, which appeared to us as especially relevant to the continuous manipulation of virtual objects, argues that there are various *biases* that influence people's initial grasp of an object. In its simplest form, the model can be expressed as follows:

$$p_{initial} = \frac{w_{anti} \cdot p_{anti} + w_{default} \cdot p_{default}}{w_{anti} + w_{default}} \tag{3.1}$$

According to the model, two different biases contribute to the initial grasp orientation $p_{initial}$. An anticipatory bias pulls the initial grasp toward a pronated or supinated angular position p_{anti}, depending on the intended direction of rotation. A

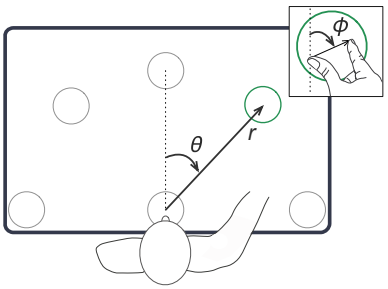


Figure 3.16 : The position of interactive objects expressed in polar coordinates (r, θ), where r is the radial distance and θ is the clockwise angle with respect to the vertical axis of the screen. The grip orientation is expressed by the clockwise angle ϕ defined by the thumb and the index.

second bias pulls the initial grasp toward a preferred task-independent orientation $p_{default}$. The contributing weights w_{anti} and $w_{default}$ of the two biases can vary, for example, depending on the difficulty of the task or the required end precision.

In [18], we describe a series of experiments that test if such prospective motor control also takes place in the context of virtual objects manipulated on a tabletop. In those experiments, participants were asked to grab a circular *start* object with the thumb and the index of the right hand, and to manipulate it to make its position and orientation match another circular *target* object.

In order to test if there is a *default* bias, we considered different locations for the *start* object. Regarding the potential effect of an *anticipatory* bias, we considered two conditions differing in whether users could plan their movements or not:

- *target* is displayed *before* participants grab the *start* (Known Target, planning possible), or
- *target* is displayed *after* participants grab the *start* (Hidden Target, planning not possible).

Figure 3.16 illustrates the six screen positions for both the *start* and the *target* objects that we tested. One was located close to the user, centered on the vertical axis of the display, 35 mm from the front edge. We refer to it as the *User* position. The other five positions were located around the *User* position with an angular position θ_{start} of $-90°$, $-45°$, $0°$, $45°$, and $90°$, and a radial distance of $r = 314$ mm. Our main measure was participants' grip orientation ϕ_{init} (*i.e.*, the angle defined by the thumb and index when participants grab the object).

Default bias In order to observe if there is a preferred task-independent way of grabbing an object (*default* bias), we considered the trials where the *start* and *target* configurations were the same. In such trials, the *start* and the *target* objects were

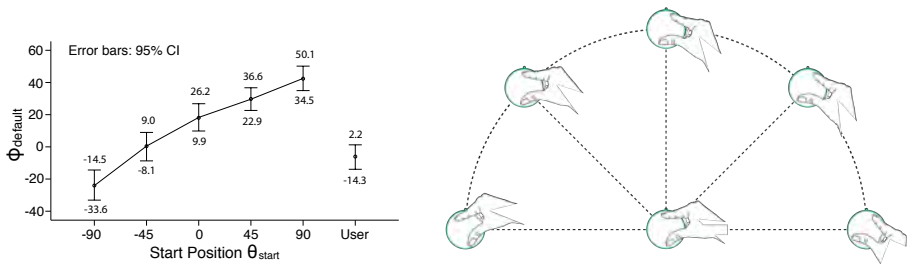


Figure 3.17 : *default* bias. Effect of start position θ_{start} on $\phi_{default}$

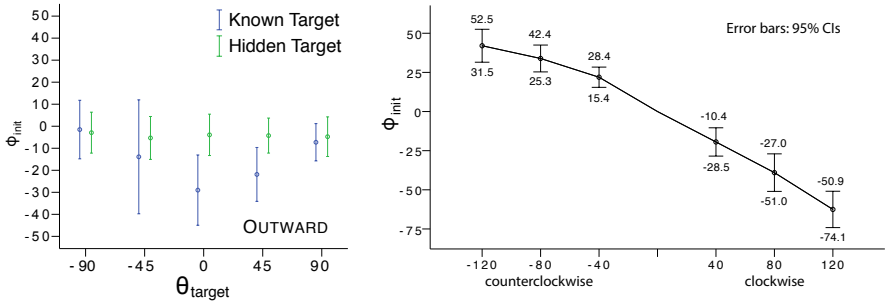


Figure 3.18 : *anticipatory* bias. (Left) Effect of target angular positions θ_{target} on the grip orientation ϕ_{init} for outward translations. (Right) Effect of the rotation angle β on ϕ_{init}

appearing at the same position, and the participant had to hold the *start* object and keep it inside the *target*. Figure 3.17 reports participants' grip orientation for those trials $\phi_{default}$, and clearly illustrates that $\phi_{default}$ varied along different angular positions. This supports the fact that, like with physical objects, there is a *default* bias when manipulating virtual objects.

Anticipatory bias Our observations also corroborate the fact that there is an *anticipatory* bias when manipulating virtual objects on a tabletop. For example, Figure 3.18-(Left) illustrates that ϕ_{init} is different depending on whether participants were able to plan their manipulation or not. In the presence of translations, the anticipatory bias is dependent on the $\phi_{default}$ orientation of the *target* position. In the presence of rotations, the bias is dependent on the direction β_{dir} and the range β of the rotation (Figure 3.18-(Right)). When both rotations and translations are combined, both these factors contribute to ϕ_{init} but with different weights.

Overall, our results support that there is some prospective planning when people perform translations and rotations of virtual objects on a tabletop. We have shown that users choose a grip orientation that is influenced by three factors: (1) a preferred orientation defined by the start object position, (2) a preferred orientation

defined by the target object position, and (3) the anticipated object rotation. We have also shown that our results are consistent with Herbort's WIMB model for physical objects [HB12].

These results suggest that we could possibly infer information about users' prospective movement to improve user experience during the manipulation phase. Interface designers could, for example, develop techniques that adapt their graphical layout to improve visual feedback, avoid potential occlusion issues [HB04, VC12] or reduce interference [HMDR08] when multiple users interact in close proximity in collaborative settings. We could also derive directions about how to design grips and visual guides to facilitate both the acquisition and the manipulation of virtual objects.

3.3.2 A design space for multi-touch gestures

We believe that the use of multi-touch gestures should not be restricted to object manipulations that mimic how we interact with objects in the physical world, but that such gestures also offer an efficient means for invoking commands. Enabling users with a larger set of multi-touch gestures than what current systems do could both make interaction easier and smoother in some scenarios, and make graphical presentations lighter by avoiding graphical widgets. For example, making a text selection on a small touchscreen can be difficult. In particular, if the text to select is longer than what the viewport can accommodate, users need to rely on an autoscrolling mechanism which, as any first-order control, can pose difficulties when, *e.g.*, trying to repair an overshooting error. If users were rather provided with gestures for adjusting the text selection that integrate with navigational finger slides, they could fluidly transition between navigation and selection actions, and thus have a finer control.

In [17], we propose a design space of multi-touch gestures. This space is organized along dimensions that (i) make sense in terms of human anatomy, (ii) do not involve complex shapes and (iii) can be systematically explored. Our goal was to offer a set of gestures that remain simple to execute, and that could ideally be easy to input in a sequence, to provide fluid transitions between the different controls that need to be chained to achieve many tasks. In addition to the text selection example mentioned above, users may want to pan a representation at different speeds to reach quickly and precisely a given paragraph in a PDF document, or a geographical area in a map. They might also want to manipulate different degrees of freedom of a graphical object, as in a 3D docking task [NBBW09], or activate a command and immediately set the value of its parameters in a fluid manner [GW00].

Design space

Our multitouch gesture design space is defined along four dimensions: *Contact Point (CP)*, *Constraint*, *Reference* and *Shape*.

Constraint	FREE				ANCHORED			
Reference	EXTERNAL		INTERNAL		EXTERNAL		INTERNAL	
Shape	LINEAR	CIRCULAR	LINEAR	CIRCULAR	LINEAR	CIRCULAR	LINEAR	CIRCULAR
1 CP								
2 CP								
3 CP								
4 CP	...	...	...	...				
5 CP	...	...	...	...				

Figure 3.19 : Design space for multi-touch gestures. Pictures show a subset of this space by illustrating only gesture classes that involve from 1 to 3 contact points (CP) and that feature at most one anchor (the thumb).

In the ***Contact Point*** dimension, the number of contact points involved in a gesture is defined. For a single-handed interaction, its values ranges from 1 to 5. To be compatible with current technology, our design space does not consider finger identification. For example, a 2-CP gesture may involve the thumb and the index finger, middle and ring finger or any other two-finger combination. ***Constraint*** refers to the behavior of the contact points, that are either active or static. A gesture is *free* when all contact points are active. It is *anchored* when it has at least one static and one active CPs. The ***Reference*** dimension reflects whether an invariant point serves as a reference for gesture execution or not. Gestures are *internal* if there is a reference, being the centroid of contact points for free gestures or the anchor digit for anchored gestures. The ***Shape*** dimension captures the gestures' form. When creating the design space, we consciously decided to include only simple shapes: *linear* or *circular*.

The design space, illustrated in Figure 3.19 with a subset of gestures, is obtained by crossing the values of the above dimensions. In practice, the space is reduced by two main constraints. Both anchored and internal free gestures cannot be defined when the gesture involves only one contact point. Nevertheless, the design space still contains 18 free and 116 anchored gestures. This number is derived by identifying all possible finger combinations that yield discernible patterns of contact points. For example, a 3-finger anchored gesture can have one or two anchor points which can be either adjacent or divided, resulting in a total of six combinations for these gesture classes.

However, not all gestures are feasible. Strong enslaving of middle and ring fingers [YvDG10, ZLL00] will, for example, make any gesture where those two do not act in unison very difficult or even impossible. The form factor of the device also affects what proportion of the design space can be used. Large devices such

as tabletops allow for gestures with up-to-five contact points, or even the use of both hands. On the other hand, using more than three fingers on a tablet is often cumbersome.

The range of motion of gestures in the design space is not uniform. Limiting factors can be anatomical, such as the length of the involved fingers and the flexibility of the hand. The size of the device may also limit the gesture amplitude. Free external linear gestures can, for example, only be as long as the size of the screen, while free internal circular gestures are limited by how much fingers and wrist can ab- and adduct, as well as by shoulder movement. External circular gestures are, on the other hand, infinite in range, as users can perform as many rotations as they wish. Gestures that have a limited range may suit well for *discrete* controls to, *e.g.*, replace a button. Gestures that can be repeated for an arbitrary duration can be used for *continuous* controls to, *e.g.*, replace a slider.

Recognition engine

The full design space is quite large. As a first evaluation, we study a subset of gestures for a tablet-sized device, GS_{tablet}. We explicitly choose gestures that are the least challenging from the perspective of finger-coordination, have two or three contact points, and a maximum of one anchor point (second and third lines of Figure 3.19). The gestures may involve the thumb, index, middle and/or ring fingers; but participating fingers are always adjacent to each other. The anchor point is always the thumb, as it is one of the most independent fingers [HRS00, YvDG10], in particular when acting in parallel to the other ones [OZL05]. This set consists of 16 gesture classes. Within each gesture class, we consider different *directions*. Linear external gestures are along one of the four cardinal directions (NORTH, EAST, SOUTH, WEST). Linear internal gestures go either TOWARDS or AWAY from the gesture's reference point. Circular gestures can be either clockwise (CW) or counterclockwise (CCW). From this set of 40 gestures, we excluded the 8 anchored external linear gestures based on user feedback collected during informal preliminary tests (grayed out in Figure 3.19). These gestures are actually both uncomfortable and difficult to perform. In the end, GS_{tablet} consists of 32 gestures.

We introduced a recognition algorithm that can discriminate between this set of 32 gestures. Our algorithm is incremental, *i.e.*, it relies only on the most recent finger traces, to enable both early recognition and continuous control during gesture execution. By using only local geometrical features of the last points sampled, users are able to fluidly transition between different gestures without requiring explicit delimiters, like pausing or lifting fingers. Our engine analyzes the finger traces in a recognition loop that starts as soon as one finger touches the surface. To detect static fingers, the loop runs at a frequency of 40Hz (period 25ms), independently from the frequency at which the system delivers events.

In order to avoid unstable recognition due to local noise in the different finger traces, we filter the recognition results by introducing a short lag (100 ms). A gesture is considered as reliably recognized for the first time if it has been recog-

nized at least four times in a row by the recognition loop. It then remains the active gesture until any other gesture has been reliably recognized.

The loop treatment starts by looking at whether the gesture is *anchored or free*. A finger is anchored if its trace is bounded to a 50-pixel (85mm) square over the last 500ms. Otherwise, it is free. A gesture is anchored as soon as one finger in contact is anchored. The algorithm then computes the values of other dimensions by using two local geometrical features: the most recent individual finger traces and the polygon formed by the finger contact points (*i.e.*, the contact envelope). In the following, P_d refers to the polygon formed by the points pt_d of each finger trace. For a given finger, pt_d is the point located d pixels away from its current point pt_0 along its trace (*i.e.*, distance path = d).

Anchored gestures During a circular internal gesture, each free finger moves along a circle centered on the anchor location. Our algorithm considers all points of one free finger trace over the last 200 pixels (339mm), and looks at how much their distance to the anchor varies. It computes distances to the anchor for all captured points since pt_{200}. It considers a gesture as *circular internal* if the standard deviation over these distances is lower than 20 pixels. Considering only one finger trace is sufficient, as in the case of other possible anchored gestures (internal linear and external circular) all free fingers see their distance to the anchor vary.

During a linear internal gesture, all free fingers either get close to, or away from, the anchor along individual linear movements. Our algorithm considers polygons P_{50} and P_0, and the different individual finger traces over the last 50 pixels (85mm). For both polygons, it computes the vertex-centroid distance. It also computes the straightness of each individual 50-pixel trace. A trace between pt_{50} and pt_0 is considered as straight if the ratio between the length of the $pt_{50}pt_0$ segment and the distance path separating pt_{50} from pt_0 is above 0.99. A gesture is recognized as *linear internal* (i) if the vertex-centroid distance has changed by at least 15 pixels and (ii) if all free finger traces are straight. As we do not consider *linear external* gestures, the gesture is recognized as *circular external* otherwise.

Free gestures During a circular internal gesture, each free finger moves along a circle centered on the centroid of the polygon's contact envelope, while the relative position of free fingers remains constant. Our algorithm considers both polygons P_{200} and P_0 and computes their angle of reference. It recognizes a *circular internal* gesture if this angle has changed by more than $\frac{\pi}{6}$. Checking this rotation criterion is enough to discriminate this type of gesture from other gestures: neither linear internal gestures nor linear/circular external gestures involve such a rotation of the contact envelope.

During a linear internal gesture, all free fingers get either close to, or away from, the centroid of the current contact envelope P_0. Our algorithm considers polygons P_{50} and P_0 and recognizes a *linear internal* gesture if the difference between the vertex-centroid distance is higher than 15 pixels (25mm) between P_{50}

and P_0. It eventually discriminates between *circular external* and *linear external* gestures by looking at whether all finger traces are arcs over their last 100 pixels (170mm) or not. A trace between pt_{100} and pt_0 is considered as an arc if (i) it is not straight (according to the straightness criterion mentioned above for anchored linear internal) and (ii) it is not a corner (the ratio between the Mahalanobis distance from pt_{100} to pt_0 and the distance path separating pt_{100} from pt_0 is either below 0.9 or above 1.1).

Evaluation In our experiment, participants were asked to perform either one gesture or two gestures in sequence without lifting their fingers off the tablet between the two gestures. *Discrete* gestures, which are limited by either the size of the tablet or anatomical constraints, had to be consistently recognized for 1.7 cm. *Continuous* gestures, which do not have such limitations, had to be consistently recognized for 1000 ms to validate that users are able to maintain them for a substantial amount of time. The overall recognition score for the 32 tested gestures was $\sim 90\%$ when gestures are performed individually. When participants transition between two gestures, the results were more contrasted. Out of 39 transitions, 11 are recognized with an accuracy greater than 90%, but accuracy falls below 75% for 12 others. We argue that these scores represent a worst-case scenario, as the recognizer chose one gesture among the 32. In the context of a real application, only a subset of candidate gestures would likely be used. Also, we chose the transitions that we tested without considering anatomical constraints related to starting a gesture directly after the end of another. Previous studies [HNK+13, HWO+13] have shown that multi-touch gestures can be uncomfortable when started in some specific postures. Future work should rely on this literature to conduct further studies and identify the most user-friendly transitions.

3.4 Conclusion

As opposed to point-based interaction, gesture-based interaction allows users to express their intentions using movements that are less display-dependent, making this interaction style suited to a large variety of setups. However, this flexibility comes at some cost, as users cannot rely on *recognition* to invoke commands, and rather have to *recall* which gesture to perform in order to invoke a specific command. Even if gestures offer a rich space for building mnemonics, the cognitive effort for memorizing gestures and retrieving them quickly becomes very high. A multi-modal approach [NC93] can help in making gesture-based interaction scale with an increasing number of commands and controls. Combining simple messages from different input channels can offload the gesture channel, enabling users to rely on gestures that remain simple enough to memorize and perform.

Perspectives

While direct manipulation heavily relies on display resources, gestures, on the opposite, heavily depend on users' ability to learn how to use their hands in order to send messages that the system can understand. My research for the last ten years has focused on bringing empirical findings and designing solutions to improve these two types of interaction. However, these channels have intrinsic limitations. On the one hand, direct manipulation does not scale to today's interactive systems, which can involve very small or very large devices, and even combinations of several devices in the same set up. On the other hand, gestures inevitably hit humans' cognitive and motor limits when the number of application commands and controls becomes too large. I argue that combining different input types is a promising approach to efficiently support complex interactive environments. In particular, I strongly believe that *combining gestures and tangibles* offers a very rich input channel that has the potential to improve user experience with applications that feature a large number of controls, or that involve several devices with potentially multiple users. Tangibles afford specific grasps and manipulations, and can thus act as guides for "telling" users what and how gestures can be performed. Also, holding a tangible object decreases fingers' degrees of freedom and will likely lead to movement trajectories that are easier to discriminate with a recognition engine. Finally, objects can make gestures polymorph [BLM00], as the same gesture performed with different objects can have different meanings. Polymorphism could lower users' cognitive load by offering an interaction paradigm that borrows from both *recall* and *recognition* paradigms: the number of different gestures that users need to *recall* can be small and can be reused with different objects that support *recognition* thanks to their physicality.

In the coming years, I want to study in depth this idea of combining gestures and tangibles, that I started to investigate within the context of Rafael Morales González's PhD. The first results are very promising and raise many research questions that go far beyond the context of a single PhD. I want to push this concept further along two main axes: (1) understand where its design limitations are, how it can be brought to the general public, and consider its use for manipulating two specific types of complex datasets (graphs of heterogeneous data and geographical data); and (2) study how they could scale to, and what they can bring in, a multi-display and/or multi-user environment.

4.1 Tangibles for any application and for all users

My hypothesis is that the geometry of a tangible object impacts how users grasp it and what movements they can do while holding it, reducing and structuring the initially-infinite space of gestures. We did some first investigations with tokens featuring carved notches to indicate where fingers should be placed when held on a tactile surface [16]. Our results show that we can design at least six tokens that actually lead to different touch patterns (*i.e.*, the touch points' relative positions).

We are able to discriminate these patterns with more than 98% of accuracy using a recognition engine that we designed, and which does not require any training. In our experiment, participants were able to comfortably execute three types of simple gestures (rotating the token, or sliding it along a linear or a circular trajectory).

The originality of this approach is that, as opposed to other tangible interfaces, it neither requires any special equipment such as a diffuse illumination tabletop (*e.g.*, [JGAK07]) nor does it require augmenting the tokens with conductive circuit (*e.g.*, [KWRE11]) or specific sensors (*e.g.*,[HAW13]). It simply relies on carving notches in the tokens to help users position their fingers when holding the token on the surface. Because it is so simple and low-cost, I believe that it has a strong potential if we can (1) show that we can design a large number of different tokens to cover many usages, and (2) provide developers and end-users with tools that allow them to easily build their own token sets and customize their applications.

4.1.1 Designing tangibles

Our first proof-of-concept shows that we can build at least one set of six tangibles that have simple geometric shapes. Reactions to demonstrations of this first prototype are often about the scalability of the approach in terms of both the number of tokens (*e.g.*, "how many different tokens can I consider for my application?") and the degree of customization of each token (*e.g.*, "can I put decorations like fur or jewels on my token?" or "can I create a token that has the shape of a phone or of my favorite cartoon character?"). Such questions call for empirical research with the aim of increasing the power of expression of such tangibles by identifying and characterizing the space within which we can design them. Both anatomical and cognitive constraints must be considered. Regarding anatomy, tangibles must be comfortable to grasp while leading to distinguishable grasps. I plan to conduct studies to identify the criteria in shape, size and volume that the tangibles must meet. Regarding cognition, variations in the shape, size and material of tokens all play an important role in providing the right manipulation affordances and conveying the proper semantics. As our current approach relies on carving notches that alter the shape of tangibles, it is important to understand how shape and semantics are tied in order to define heuristics that indicate how a shape can be carved-in without altering its meaning. For example, heuristics could tell the amount and the type of detail that can be removed from a token while preserving both its meaning in isolation and its identity within the considered token set.

4.1.2 Implementing tangibles

Implementing innovative interactive systems is a difficult task with conventional UI toolkits. Tangible interfaces are especially challenging in that regard, as their physicality makes them resistant to customization [HAW13]. The approach based on passive tokens is very promising to change this state of affairs because of its low cost in terms of construction and implementation. However,

there is still some important effort to put on development to make developers and end-users able to create or customize their tangible interfaces. I plan to work on providing development solutions that make interface designers and end-users able to develop applications with custom-made tokens that suit their specific requirements. In terms of design, what they need are tools that let either designers or end-users easily specify the tokens' shape, that guide them in the process of placing notches ensuring both comfort and recognition accuracy, and that provide them with recommendations for building recognizable tokens. Ideally, these tools would be developed as plugins for mainstream vector drawing editors such as Adobe Illustrator, in order to be as flexible as possible. These design tools should be able to output token descriptions that can be sent to any construction device (*e.g.*, a laser cutter or a 3D printer) and to the underlying token recognition engine. In terms of programming, this latter engine needs to rely on robust algorithms for accurately tracking the tokens' state (where they are on the surface, what their exact position and orientation are, etc.) with an API that remains simple for developers. This API should be built on top of programming languages that most platforms with tactile screens support (JavaScript and Android appear as good candidates). While appearing simple at first, relying on fully passive tangibles is actually challenging when developing robust tracking strategies, as it implies that the system must infer a lot from very few input data, which are essentially limited to the fingers' contact points. Finally, it would be interesting to allow end-users to customize their tangible applications (especially if we think about personal devices like smartphones) by designing mechanisms that expose mappings between controls and tokens, in order to let users change them on-demand.

4.1.3 Application domains

In addition to the empirical and engineering research mentioned above, there is a need to demonstrate how this type of input can actually be used to control complex applications. I consider two application domains where users have to perform many manipulations on a large amount of data: geographical data and networks of heterogeneous data.

Geovisualization systems allow users to present geographical data that are usually arranged into piles of layers that users navigate. We study this type of navigation in the context of María Jesús Lobo's PhD. We first conducted an empirical evaluation of existing techniques [14], and are currently investigating a novel model to support interactive compositing between layers. We want this model to be very flexible, so as to let users define the area where the compositing should take place and whether it should be done instantaneously or smoothly animated. The number of manipulations can be very high as users can define many different regions with varying compositing parameters.

One of the research axes of the ILDA team is to design interactive visualizations to navigate and manipulate large webs of data. We consider graphs where the different entities can have heterogeneous types and where links express different

semantic relationships between these entities, the more general instance of such graphs being the Web of Data [HB11]. This type of data is very rich and calls for highly expressive input if we want users to be able to directly interact with it in a flexible way. I consider it as the most challenging application domain for demonstrating how scalable my approach can be. We started to investigate this area in the context of Hugo Romat's PhD.

4.2 Gestures and tangibles for multi-display and multi-user environments

As already mentioned in this manuscript, tactile surfaces can have varying form factors, ranging from handheld devices (such as smartphones and tablets) to wall-sized displays and tabletops. Recent interactive platforms combine several such tactile surfaces in a room (*e.g.*, the WILDER platform at Inria Saclay) and offer an environment that is especially well suited to collaborative interaction. Our recent experiences developing advanced applications in the domains of astronomy [22] and GIS[1] revealed that designing coherent input vocabularies for such multi-display and multi-user environments is a significant research challenge. In this section, I discuss how gestures and tangibles could be an efficient means to interact with such advanced set ups, and I present the research questions that I want to work on.

4.2.1 Multi-display environments

Multi-display gestures

In a multi-display environment, information is distributed across the different surfaces. Users need to be able to move from one surface to the other and to remotely interact with a surface when this latter is not within arm's reach. Gestures, both touch and mid-air, are portable and represent an ideal candidate for input, as opposed to traditional mouse and keyboard that require users to remain seated at a desk. However, the varying nature of the diverse surfaces (small vs. large, horizontal vs. vertical, tactile vs. remotely controlled) prevents us from using the same gestures on all surfaces. For example, some surfaces do not support tactile input, and others are too small to accommodate multi-touch gestures. The challenge is thus to design gesture vocabularies that do not require users to learn a completely different set of gestures for each surface. I believe that this can be done by identifying high-level dimensions for describing gestures as we have started to do in [17]. This level of abstraction would go higher than the touch-point level by rather considering clusters of points that get defined depending on their coherence or difference in terms of dynamics (presence or absence of invariants, shared or opposite movement direction, internal or external frame of reference, etc.). With

[1] ANR project MapMuxing: `http://mapmuxing.ign.fr`

this approach, we would define families of gestures that can encompass a version for each type of surface. As a concrete and simple example, defining a pinch gesture as two clusters of points that evolve in opposite directions encompasses both the pinch performed with two fingers on a portable device and the pinch performed with two hands on an interactive wall.

Multi-display tokens

A physical object can be a powerful tool for interacting in multi-display environments. It can be used, *e.g.*, as a versatile controller that can be moved from one surface to another (factorizing the number of controls in the environment), or as a data container to carry information between the different displays (making the dependency between data and displays lower). Designing physical tokens that can be both manipulated and tracked in a multi-surface environment is difficult. For example, handling several tokens on a vertical surface can be uncomfortable if they cannot be conveniently stored when not in use. Investigating how we can combine different materials to fabricate tokens that can stick on wall displays while remaining easy to move is one of the design challenges I would like to work on. Also, different surfaces use different tracking technologies (capacitive screens, light-based touch frames, external cameras), making the design of a token that can be transparently tracked by all of them an interesting technical challenge.

4.2.2 Multi-user environments

Combining gestures and tangibles can also provide efficient support for collaborative interaction. Gestures and objects encode a lot of information in their shape, dynamics and direction, that can be directly interpreted in relation with the user, independently from the display output (as opposed to, *e.g.*, a mouse movement or click that can only be interpreted according to the graphical controller (widget) above which it occurs). On the one hand, physical objects can offer a significantly better user experience for collaborative problem solving. For example, in a study reported in [SJZD11], groups of users were better at designing warehouses with tangibles than with touch input. I also believe that their physicality can greatly improve coordination among different actors for, *e.g.*, handling priorities or assigning specific roles, by simply exchanging tokens among users. On the other hand, direct touch provides a better awareness in group work than indirect input methods [NPSG07] and promotes more equal participation [RLHM09]. Specific collaborative use cases that would benefit from both the tangible and the tactile worlds remain to be identified and investigated.

From a technical perspective, allowing multiple users to work together on the same multi-touch surface brings some difficulty to properly interpret input streams. For example, how can the system know that several points belong to the same user rather than to several users? Here, what could be interesting to investigate is an engine that, based on machine learning approaches, could tell who is gesturing. It is

worth studying the feasibility of such an approach by first empirically observing if users have different ways of gesturing, that could be discriminated using describing features such as spacing between fingers, spacing between hands/fingers at gesture start time, speed of gesturing, contact point mean altitude on a vertical display or mean distance to border on a tableto.

Publications

[1] Alvina, J., Appert, C., Chapuis, O., and Pietriga, E. Routelens: Easy route following for map applications. *AVI '14*, ACM (2014), 125–128.

[2] Appert, C. Augmenter la validité des évaluations des applications graphiques interactives. *Revue Technique et Science Informatiques 29*, 1 (2010).

[3] Appert, C., and Bau, O. Scale detection for a priori gesture recognition. *CHI '10*, ACM (2010), 879–882.

[4] Appert, C., and Beaudouin-Lafon, M. Smcanvas: Augmenter la boîte à outils java swing pour prototyper des techniques d'interaction avancées. *IHM '06*, ACM (2006), 99–106.

[5] Appert, C., and Beaudouin-Lafon, M. Swingstates: Adding state machines to the swing toolkit. *UIST '06*, ACM (2006), 319–322.

[6] Appert, C., and Beaudouin-Lafon, M. Swingstates: Adding state machines to java and the swing toolkit. *Softw. Pract. Exper. 38*, 11 (2008), 1149–1182.

[7] Appert, C., Beaudouin-Lafon, M., and Mackay, W. E. Context matters: Evaluating interaction techniques with the cis model. In *People and Computers XVIII—Design for Life*. Springer, 2005, 279–295.

[8] Appert, C., Chapuis, O., and Beaudouin-Lafon, M. Evaluation of pointing performance on screen edges. *AVI '08*, ACM (2008), 119–126.

[9] Appert, C., Chapuis, O., and Pietriga, E. High-precision magnification lenses. *CHI '10*, ACM (2010), 273–282.

[10] Appert, C., Chapuis, O., and Pietriga, E. Dwell-and-spring: Undo for direct manipulation. *CHI '12*, ACM (2012), 1957–1966.

[11] Appert, C., Chapuis, O., Pietriga, E., and Lobo, M.-J. Reciprocal drag-and-drop. *ACM Trans. Comput.-Hum. Interact. 22*, 6 (2015), 29:1–29:36.

[12] Appert, C., and Fekete, J.-D. Orthozoom scroller: 1d multi-scale navigation. *CHI '06*, ACM (2006), 21–30.

[13] Appert, C., and Zhai, S. Using strokes as command shortcuts: Cognitive benefits and toolkit support. *CHI '09*, ACM (2009), 2289–2298.

[14] Lobo, M.-J., Pietriga, E., and Appert, C. An evaluation of interactive map comparison techniques. *CHI '15*, ACM (2015), 3573–3582.

[15] Mackay, W. E., Appert, C., Beaudouin-Lafon, M., Chapuis, O., Du, Y., Fekete, J.-D., and Guiard, Y. Touchstone: Exploratory design of experiments. *CHI '07*, ACM (2007), 1425–1434.

[16] Morales González, R., Appert, C., Bailly, G., and Pietriga, E. Touchtokens: Guiding touch patterns with passive tokens. *CHI '16*, ACM (2016), 4189–4202.

[17] Olafsdottir, H., and Appert, C. Multi-touch gestures for discrete and continuous control. *AVI '14*, ACM (2014), 177–184.

[18] Olafsdottir, H. B., Tsandilas, T., and Appert, C. Prospective motor control on tabletops: Planning grasp for multitouch interaction. *CHI '14*, ACM (2014), 2893–2902.

[19] Pietriga, E., and Appert, C. Sigma lenses: Focus-context transitions combining space, time and translucence. *CHI '08*, ACM (2008), 1343–1352.

[20] Pietriga, E., Appert, C., and Beaudouin-Lafon, M. Pointing and beyond: An operationalization and preliminary evaluation of multi-scale searching. *CHI '07*, ACM (2007), 1215–1224.

[21] Pietriga, E., Bau, O., and Appert, C. Representation-independent in-place magnification with sigma lenses. *IEEE Transactions on Visualization and Computer Graphics 16*, 3 (2010), 455–467.

[22] Pietriga, E., del Campo, F., Ibsen, A., Primet, R., Appert, C., Chapuis, O., Hempel, M., Muñoz, R., Eyheramendy Duerr, S., Jordan, A., and Dole, H. Exploratory visualization of astronomical data on ultra-high-resolution wall displays. In *Proceedings of the Astronomical Telescopes and Instrumentation conference: Software and Cyberinfrastructure for Astronomy IV*, SPIE (2016).

[23] Spelmezan, D., Appert, C., Chapuis, O., and Pietriga, E. Controlling widgets with one power-up button. *UIST '13*, ACM (2013), 71–74.

[24] Spelmezan, D., Appert, C., Chapuis, O., and Pietriga, E. Side pressure for bidirectional navigation on small devices. *MobileHCI '13*, ACM (2013), 11–20.

[25] Tsandilas, T., Appert, C., Bezerianos, A., and Bonnet, D. Coordination of tilt and touch in one- and two-handed use. *CHI '14*, ACM (2014), 2001–2004.

[26] Zhai, S., Kristensson, P., Appert, C., Andersen, T., and Cao, X. Foundational issues in touch-surface stroke gesture design: An integrative review. *Found. Trends Hum.-Comput. Interact. 5*, 2 (2012), 97–205.

Bibliography

[AZ97] Johnny Accot and Shumin Zhai. Beyond fitts' law: Models for trajectory-based hci tasks. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, CHI '97, pages 295–302, New York, NY, USA, 1997. ACM.

[Bal04] Ravin Balakrishnan. "beating" fitts' law: Virtual enhancements for pointing facilitation. *Int. J. Hum.-Comput. Stud.*, 61(6):857–874, December 2004.

[BBL93] Thomas Baudel and Michel Beaudouin-Lafon. Charade: Remote control of objects using free-hand gestures. *Commun. ACM*, 36(7):28–35, July 1993.

[BCB07] Stephen Brewster, Faraz Chohan, and Lorna Brown. Tactile feedback for mobile interactions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 159–162, New York, NY, USA, 2007. ACM.

[Ber94] Thomas Berlage. A selective undo mechanism for graphical user interfaces based on command objects. *ACM Transactions on Computer-Human Interaction*, 1(3):269–294, September 1994.

[BGBL04] Renaud Blanch, Yves Guiard, and Michel Beaudouin-Lafon. Semantic pointing: Improving target acquisition with control-display ratio adaptation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 519–526, New York, NY, USA, 2004. ACM.

[BIH08] Alex Butler, Shahram Izadi, and Steve Hodges. Sidesight: Multi-"touch" interaction around small devices. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, UIST '08, pages 201–204, New York, NY, USA, 2008. ACM.

[BLM00] Michel Beaudouin-Lafon and Wendy E. Mackay. Reification, polymorphism and reuse: Three principles for designing visual interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '00, pages 102–109, New York, NY, USA, 2000. ACM.

[BM08] Olivier Bau and Wendy E. Mackay. Octopocus: A dynamic guide for learning gesture-based command sets. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, UIST '08, pages 37–46, New York, NY, USA, 2008. ACM.

[BML12] Gilles Bailly, Jörg Müller, and Eric Lecolinet. Design and evaluation of finger-count interaction: Combining multitouch gestures and menus. *Int. J. Hum.-Comput. Stud.*, 70(10):673–689, October 2012.

[Bre02] Stephen Brewster. Overcoming the lack of screen space on mobile computers. *Personal Ubiquitous Comput.*, 6(3):188–205, January 2002.

[Bux95] William A. S. Buxton. Human-computer interaction. chapter Chunking and phrasing and the design of human-computer dialogues, pages 494–499. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[CKB09] Andy Cockburn, Amy Karlson, and Benjamin B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Comput. Surv.*, 41(1):2:1–2:31, January 2009.

[CL72] F. Craik and R. Lockhart. Levels of processing: A framework for memory research. *Journal of Verbal Learning and Verbal Behavior*, 11(6):671–684, 1972.

[CLP04] M. S. T. Carpendale, John Ligh, and Eric Pattison. Achieving higher magnification in context. In *UIST '04: Proc. of the ACM Symp. on User Interface Software and Technology*, pages 71–80. ACM, 2004.

[CLP09] Olivier Chapuis, Jean-Baptiste Labrune, and Emmanuel Pietriga. Dynaspot: Speed-dependent area cursor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1391–1400, New York, NY, USA, 2009. ACM.

[CM01] M. S. T. Carpendale and Catherine Montagnese. A framework for unifying presentation space. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, UIST '01, pages 61–70, New York, NY, USA, 2001. ACM.

[CWB+08] Xiang Cao, A. D. Wilson, R. Balakrishnan, K. Hinckley, and S. E. Hudson. Shapetouch: Leveraging contact shape on interactive sur-

faces. In *2008 3rd IEEE International Workshop on Horizontal Interactive Human Computer Systems*, pages 129–136, Oct 2008.

[FB95] George W. Furnas and Benjamin B. Bederson. Space-scale diagrams: Understanding multiscale interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 234–241, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[Fit54] P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *J. Exper. Psych.*, 47:381–391, 1954.

[FKP+03] George Fitzmaurice, Azam Khan, Robert Pieké, Bill Buxton, and Gordon Kurtenbach. Tracking menus. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*, UIST '03, pages 71–79, New York, NY, USA, 2003. ACM.

[FLTL08] Richard Fung, Edward Lank, Michael Terry, and Celine Latulipe. Kinematic templates: end-user tools for content-relative cursor manipulations. In *Proc. UIST '08*, pages 47–56, New York, NY, USA, 2008. ACM.

[GBLB+04] Yves Guiard, Michel Beaudouin-Lafon, Julien Bastin, Dennis Pasveer, and Shumin Zhai. View size and pointing difficulty in multiscale navigation. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, pages 117–124, New York, NY, USA, 2004. ACM.

[GDB07] Tovi Grossman, Pierre Dragicevic, and Ravin Balakrishnan. Strategies for accelerating on-line learning of hotkeys. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 1591–1600, New York, NY, USA, 2007. ACM.

[GHB+13] Emilien Ghomi, Stéphane Huot, Olivier Bau, Michel Beaudouin-Lafon, and Wendy E. Mackay. Arpège: Learning multitouch chord gestures vocabularies. In *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces*, ITS '13, pages 209–218, New York, NY, USA, 2013. ACM.

[GKN05] Emden R. Gansner, Yehuda Koren, and Stephen C. North. Topological fisheye views for visualizing large graphs. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):457–468, 2005.

[Gos82] James Gosling. *Unix Emacs Reference Manual*. Carnegie-Mellon University, Pittsburgh, PA, USA, 1982.

[GS03] Carl Gutwin and Amy Skopik. Fisheyes are good for large steering tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, pages 201–208, New York, NY, USA, 2003. ACM.

[Gut02] Carl Gutwin. Improving focus targeting in interactive fisheye views. In *CHI '02: Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*, pages 267–274. ACM, 2002.

[GW00] François Guimbretière and Terry Winograd. Flowmenu: Combining command, text, and data entry. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, UIST '00, pages 213–216. ACM, 2000.

[HAW13] Sungjae Hwang, Myungwook Ahn, and Kwang-yun Wohn. Maggetz: Customizable passive tangible controllers on and around conventional mobile devices. In *Proc. UIST '13*, pages 411–416. ACM, 2013.

[HB04] Mark S. Hancock and Kellogg S. Booth. Improving menu placement strategies for pen input. In *Proceedings of Graphics Interface 2004*, GI '04, pages 221–230, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[HB11] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, February 2011. 136 pages.

[HB12] Oliver Herbort and Martin Butz. The continuous end-state comfort effect: weighted integration of multiple biases. *Psychological Research*, 76(3):345–363, 2012.

[HMDR08] Eva Hornecker, Paul Marshall, Nick Sheep Dalton, and Yvonne Rogers. Collaboration and interference: awareness with mice or touch input. In *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, CSCW '08, pages 167–176, New York, NY, USA, 2008. ACM.

[HNK+13] Eve Hoggan, Miguel Nacenta, Per Ola Kristensson, John Williamson, Antti Oulasvirta, and Anu Lehtiö. Multi-touch pinch gestures: Performance and ergonomics. In *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces*, ITS '13, pages 219–222. ACM, 2013.

[HRS00] C. Hager-Ross and M.H. Schieber. Quantifying the independence of human finger movements: comparisons of digits, hands, and movement frequencies. *Journal of Neuroscience*, 20(22):8542, 2000.

[HS11] Ken Hinckley and Hyunyoung Song. Sensor synaesthesia: touch in motion, and motion in touch. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, pages 801–810, New York, NY, USA, 2011. ACM.

[HWO+13] Eve Hoggan, John Williamson, Antti Oulasvirta, Miguel Nacenta, Per Ola Kristensson, and Anu Lehtiö. Multi-touch rotation gestures: Performance and ergonomics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 3047–3050. ACM, 2013.

[HZS+07] Ken Hinckley, Shengdong Zhao, Raman Sarin, Patrick Baudisch, Edward Cutrell, Michael Shilman, and Desney Tan. Inkseine: In situ search for active note taking. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 251–260, New York, NY, USA, 2007. ACM.

[IH00] Takeo Igarashi and Ken Hinckley. Speed-dependent automatic zooming for browsing large documents. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, UIST '00, pages 139–148, New York, NY, USA, 2000. ACM.

[ISO00] ISO. 9241-9 Ergonomic requirements for office work with visual display terminals (VDTs)-Part 9: Requirements for non-keyboard input devices. *International Organization for Standardization*, 2000.

[JF98] Susanne Jul and George W. Furnas. Critical zones in desert fog: Aids to multiscale navigation. In *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*, UIST '98, pages 97–106, New York, NY, USA, 1998. ACM.

[JGAK07] Sergi Jordà, Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner. The reactable: Exploring the synergy between live music performance and tabletop tangible interfaces. In *Proc. TEI '07*, pages 139–146. ACM, 2007.

[JSMM94] Robert J. K. Jacob, Linda E. Sibert, Daniel C. McFarlane, and M. Preston Mullen, Jr. Integrality and separability of input devices. *ACM Trans. Comput.-Hum. Interact.*, 1(1):3–26, March 1994.

[KF88] David Kurlander and Steven Feiner. Editable graphical histories. In *IEEE Workshop on Visual Languages*, pages 127–134. IEEE, 1988.

[KLDK08] Akrivi Katifori, George Lepouras, Alan Dix, and Azrina Kamaruddin. Evaluating the significance of the desktop area in everyday computer use. In *First International Conference on Advances in Computer-Human Interaction*, ACHI '08, pages 31–38. IEEE, 2008.

[KM94] G. Kurtenbach and T. Moran. Contextual animation of gestural commands. *Eurographics Computer Graphics Forum*, 13(5):305–314, 1994.

[Ks05] Maria Karam and m.c. schraefel. A taxonomy of gestures in human computer interactions. Technical Report. 2005.

[KWRE11] Sven Kratz, Tilo Westermann, Michael Rohs, and Georg Essl. Capwidgets: Tangible widgets versus multi-touch controls on mobile devices. In *CHI EA '11*, pages 1351–1356. ACM, 2011.

[LNPS05] D.M. Lane, H.A. Napier, S.C. Peres, and A. Sandor. Hidden Costs of Graphical User Interfaces: Failure to Make the Transition from Menus and Icon Toolbars to Keyboard Shortcuts. *International Journal of Human-Computer Interaction*, 18(1):133–144, 2005.

[LRP95] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *CHI '95: Proc. of the SIGCHI Conf. on Human Factors in Comp. Syst.*, pages 401–408. ACM/Addison-Wesley Publishing Co., 1995.

[MGT+03] Tamara Munzner, François Guimbretière, Serdar Tasiran, Li Zhang, and Yunhong Zhou. Treejuxtaposer: scalable tree comparison using focus+context with guaranteed visibility. In *SIGGRAPH 2003: Proc. of the Conf. on Comp. Graphics and Interactive Tech.*, pages 453–462. ACM, 2003.

[MLL+15] Brad A. Myers, Ashley Lai, Tam Minh Le, YoungSeok Yoon, Andrew Faulring, and Joel Brandt. Selective undo support for painting applications. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 4227–4236, New York, NY, USA, 2015. ACM.

[Mye98] Brad A. Myers. Scripting graphical applications by demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '98, pages 534–541, New York, NY, USA, 1998. ACM Press/Addison-Wesley Publishing Co.

[NBBW09] Miguel A. Nacenta, Patrick Baudisch, Hrvoje Benko, and Andy Wilson. Separability of spatial manipulations in multi-touch interfaces. In *Proceedings of Graphics Interface 2009*, GI '09, pages 175–182. CIPS, 2009.

[NC93] Laurence Nigay and Joëlle Coutaz. A design space for multimodal systems: Concurrent processing and data fusion. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 172–178, New York, NY, USA, 1993. ACM.

[NPSG07] Miguel A. Nacenta, David Pinelle, Dane Stuckel, and Carl Gutwin. The effects of interaction technique on coordination in tabletop groupware. In *Proceedings of Graphics Interface 2007*, GI '07, pages 191–198, New York, NY, USA, 2007. ACM.

[NWP+11] Mathieu Nancel, Julie Wagner, Emmanuel Pietriga, Olivier Chapuis, and Wendy Mackay. Mid-air pan-and-zoom on wall-sized displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 177–186, New York, NY, USA, 2011. ACM.

[NY95] W. Niblack and J. Yin. A pseudo-distance measure for 2d shapes based on turning angle. *Image Processing, International Conference on*, 3:3352, 1995.

[OZL05] Halla Olafsdottir, Vladimir M Zatsiorsky, and Mark L Latash. Is the thumb a fifth finger? a study of digit interaction during force production tasks. *Experimental brain research*, 160(2):203–213, 2005.

[PPCP12] Cyprien Pindat, Emmanuel Pietriga, Olivier Chapuis, and Claude Puech. Jellylens: Content-aware adaptive lenses. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 261–270, New York, NY, USA, 2012. ACM.

[RCBBL07] Gonzalo Ramos, Andy Cockburn, Ravin Balakrishnan, and Michel Beaudouin-Lafon. Pointing lenses: facilitating stylus input through visual-and motor-space magnification. In *CHI '07: Proc. of the SIGCHI Conf. on Human Factors in Comp. Syst.*, pages 757–766. ACM, 2007.

[RLG09] Anne Roudaut, Eric Lecolinet, and Yves Guiard. Microrolls: Expanding touch-screen input vocabulary by distinguishing rolls vs. slides of the thumb. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 927–936, New York, NY, USA, 2009. ACM.

[RLHM09] Yvonne Rogers, Youn-kyung Lim, William R Hazlewood, and Paul Marshall. Equal opportunities: Do shareable interfaces promote more group participation than single user displays? *Human–Computer Interaction*, 24(1-2):79–116, 2009.

[RM93] George G. Robertson and Jock D. Mackinlay. The document lens. In *UIST '93: Proc. ACM Symp. on User Interface Softw. and Tech.*, pages 101–108. ACM Press, 1993.

[SB94] Manojit Sarkar and Marc H. Brown. Graphical fisheye views. *Communications of the ACM*, 37(12):73–83, 1994.

[SCG13] Joey Scarr, Andy Cockburn, and Carl Gutwin. Supporting and exploiting spatial memory in user interfaces. *Found. Trends Hum.-Comput. Interact.*, 6(1):1–84, December 2013.

[Shn87] B. Shneiderman. Human-computer interaction. chapter Direct Manipulation: A Step Beyond Programming Languages, pages 461–467. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[SJZD11] Bertrand Schneider, Patrick Jermann, Guillaume Zufferey, and Pierre Dillenbourg. Benefits of a tangible interface for collaborative learning and interaction. *IEEE Trans. Learn. Technol.*, 4(3):222–232, July 2011.

[SRC05] Katie A. Siek, Yvonne Rogers, and Kay H. Connelly. Fat finger worries: How older and younger users physically interact with PDAs. In *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction*, INTERACT '05, pages 267–280, Berlin, Heidelberg, 2005. Springer-Verlag.

[VC12] Daniel Vogel and Géry Casiez. Hand occlusion on a multi-touch tabletop. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 2307–2316, New York, NY, USA, 2012. ACM.

[WBP+11] Daniel Wigdor, Hrvoje Benko, John Pella, Jarrod Lombardo, and Sarah Williams. Rock & rails: Extending multi-touch interactions with shape gestures to enable precise spatial manipulations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1581–1590, New York, NY, USA, 2011. ACM.

[WMW09] Jacob O. Wobbrock, Meredith Ringel Morris, and Andrew D. Wilson. User-defined gestures for surface computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1083–1092, New York, NY, USA, 2009. ACM.

[Yan92] Yiya Yang. Anatomy of the design of an undo support facility. *International journal of man-machine studies*, 36(1):81–95, January 1992.

[YMK13] YoungSeok Yoon, Brad A. Myers, and Sebon Koo. Visualization of fine-grained code change history. In *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, VL/HCC '13, pages 119–126. IEEE, 2013.

[YvDG10] Wei Shin Yu, Hiske van Duinen, and Simon C Gandevia. Limits to the control of the human thumb and fingers in flexion and extension. *Journal of neurophysiology*, 103(1):278–289, 2010.

[ZLL00] Vladimir M Zatsiorsky, Zong-Ming Li, and Mark L Latash. Enslaving effects in multi-finger force production. *Experimental Brain Research*, 131(2):187–195, 2000.

Selected Publications

High-Precision Magnification Lenses

Caroline Appert[1,2] appert@lri.fr
Olivier Chapuis[1,2] chapuis@lri.fr
Emmanuel Pietriga[2,1] emmanuel.pietriga@inria.fr

[1]LRI - Univ. Paris-Sud & CNRS Orsay, France
[2]INRIA Orsay, France

ABSTRACT

Focus+context interfaces provide in-place magnification of a region of the display, smoothly integrating the focus of attention into its surroundings. Two representations of the data exist simultaneously at two different scales, providing an alternative to classical pan & zoom for navigating multi-scale interfaces. For many practical applications however, the magnification range of focus+context techniques is too limited. This paper addresses this limitation by exploring the *quantization* problem: the mismatch between visual and motor precision in the magnified region. We introduce three new interaction techniques that solve this problem by integrating fast navigation and high-precision interaction in the magnified region. *Speed* couples precision to navigation speed. *Key* and *Ring* use a discrete switch between precision levels, the former using a keyboard modifier, the latter by decoupling the cursor from the lens' center. We report on three experiments showing that our techniques make interacting with lenses easier while increasing the range of practical magnification factors, and that performance can be further improved by integrating speed-dependent visual behaviors.

Author Keywords

Focus+Context, Lenses, Quantization, Navigation, Selection

ACM Classification Keywords

H. Information Systems H.5 Information Interfaces and Presentation H.5.2 User Interfaces (H.1.2, I.3.6)

General Terms

Design, Human Factors

INTRODUCTION

Although display technologies continue to increase in size and resolution, datasets are increasing even faster. Scientific data, e.g., telescope images and microscope views of the brain, and generated data, e.g., network visualizations, geographical information systems and digital libraries, are too big to be displayed in their entirety, even on very large wall-sized displays. In Google Maps, the ratio between extreme scales is about 250,000. Vast gigapixel images, such as the 400,000-pixel wide image of the inner-part of our galaxy from the Spitzer telescope also require huge scale factors between a full overview and the most detailed zoom. Users do not necessarily need to navigate through the entire scale range at one given time, but still, they need interaction techniques that will allow them to fluidly navigate between focused and contextual views of large datasets. Such techniques are typically based on the following interface schemes [8]: overview + detail, zooming, focus + context; none of which offers an ideal solution. The task determines which technique is most appropriate, taking scale range, the nature of the representation, input device, available screen real-estate, and of course, the user's preferences, into account.

This paper introduces techniques designed to improve lens-based focus+context interfaces. Our goals are to extend the range of practical magnification factors, which is currently very limited, and to make low-level interactions easier. For the sake of clarity, we illustrate all of our techniques with one common type of lens: constrained magnification lenses [4, 18, 19]. However, our improvements are generic and apply to all types of lenses. They can also be adapted to other focus+context interfaces, including hyperbolic trees [16] and stretchable rubber sheets [20].

QUANTIZATION IN FOCUS+CONTEXT INTERFACES

Constrained lenses provide in-place magnification of a bounded region of the representation (Figure 1-a). The *focus* is integrated in the *context*, leaving a significant part of the latter unchanged. Typical examples of such lenses include magnifying glasses and many distortion-oriented techniques

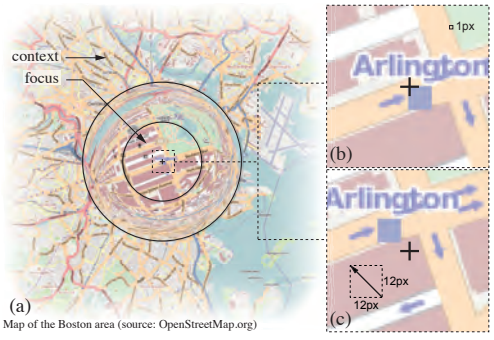


Figure 1. (a) In-place magnification by a factor of 12; (b) center of magnified region with cursor in the middle (detail); (c) same region after moving the lens by one pixel both South and East.

CHI 2010, April 10 – 15, 2010, Atlanta, Georgia, USA

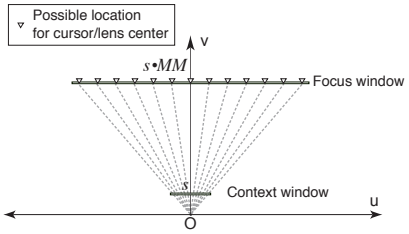


Figure 2. Space-scale diagram of possible locations for lens center (each ray corresponds to one pixel in context space).

such as the so-called graphical fisheyes. Early implementations of magnification techniques only magnified the pixels of the context by duplicating them without adding more detail, thus severely limiting the range of useful magnification factors (up to 4x). Newer implementations [4, 18] do provide more detail as magnification increases. Theoretically, this means that any magnification factor can be applied, if relevant data is available. In practice, this is not the case as another problem arises that gets worse as magnification increases: *quantization*.

Lenses are most often coupled with the cursor and centered on it. The cursor, and thus the lens, are operated at context scale. This allows for fast repositioning of the lens in the information space, since moving the input device by one unit makes the lens move by one pixel at context scale. However, this also means that when moving the input device by one unit (dot), the representation in the magnified region is offset by MM pixels, where MM is the focus' magnification factor. This means that only one pixel every MM pixels can fall below the cursor in the magnified region. In other words some pixels are unreachable, as visual space has been enlarged in the focus region but motor space has not.

This problem is illustrated in Figure 1: between (b) and (c), the lens has moved by 1 unit of the input device, corresponding to 1 pixel in the context, but the magnified region is offset by 12 pixels. Objects can thus be difficult or even impossible to select; even if their visual size is above what is usually considered a small target (less than 5 pixels). The square representing Arlington station in Figure 1 is 9-pixel wide, yet its motor size is only 1 pixel.

Figure 2 illustrates the problem with a space-scale diagram [11]: the center of the lens can only be located on a pixel in the focus window that is *aligned* – on the same ray in the space-scale diagram – with a pixel in the context window. Since the focus window is MM^2 larger than the context window, and since the cursor is located at the lens' center, only one out of MM^2 pixels can be selected. Figure 2 shows that as MM increases, more pixels become unreachable.

Beyond the general problem of pixel-precise selection in the magnified region, quantization also hinders focus targeting, i.e., the action that consists in positioning the lens on the object of interest [12, 18]. This action gets harder as the magnification factor increases, even becoming impossible at extreme magnification factors.

This quantization problem has limited the range of magnification factors that can be used in practice; the upper limit reported in the literature rarely exceeds 8x, a value relatively low compared to the ranges of scale encountered in the information spaces mentioned earlier.

In this paper, we introduce techniques that make it possible to perform both fast navigation for focus targeting and high-precision selection in the focus region in a seamless manner, enabling higher magnification factors than those allowed by conventional techniques. After an overview of related work, we introduce our techniques. *Speed* continuously adapts motor precision to navigation speed. *Key* and *Ring* use a discrete switch between two levels of precision (focus and context), the former using an additional input channel, the latter by decoupling the cursor from the lens' center. We then report the results of two controlled experiments that evaluate focus targeting and object selection performance. Finally, we iterate our designs by integrating speed-dependent visual behaviors from the Sigma Lens framework [18]. The resulting hybrid lenses further improve performance, as shown in a third controlled experiment.

RELATED WORK

Most techniques for navigating multi-scale information spaces are based on either overview + detail, zooming or focus + context (see Cockburn et al. [8] for a very thorough survey). Zooming interfaces, e.g., [21, 14] display a single level of scale and therefore require a temporal separation to transition between "focus" and "context" views. They usually do not suffer from quantization effects, but both views cannot be observed simultaneously. Overview+detail interfaces [13, 22] show both views simultaneously using spatial separation, still requiring some mental effort to integrate the two views. They usually allow pixel-precise selections in the detail region, but focus targeting is also subject to quantization problems in conventional bird's eye views.

Focus+context techniques *"aim to decrease the short term memory load associated with assimilating distinct views of a system"* [8] by integrating the focus region inside the context. This integration, however, limits the range of magnification factors of practical use. Basic magnifying glasses occlude the surroundings of the magnified region [12]. To address this issue, distortion oriented techniques provide a smooth transition between the focus and context views. Distortion, however, causes problems for focus targeting and understanding of the visual scene. Carpendale et al. [4] describe elaborate transitions that enhance the rendering of the distorted area and make higher magnifications comprehensible from a visual perspective. Gutwin's *Speed-coupled flattening lens* [12] cancels distortion when the lens is repositioned by the user, thus removing a major hindrance to focus targeting. The Sigma Lens framework [18] generalizes the idea of speed-coupling to a larger set of lens parameters. For example, the *Speed-coupled blending lens* makes focus targeting easier from a motor perspective by increasing the focus region's size for the same overall lens size, using a dynamically varying translucence level to smoothly transition between focus and context.

Although their primary goal is different, focus+context interfaces share issues with techniques designed to facilitate pointing on the desktop. The decoupling of visual and motor spaces plays a central role in techniques designed to facilitate the selection of small targets, e.g., [6, 7, 17] – see [2] for a detailed survey. Not designed for exploratory multi-scale navigation, but closer to our problem are pointing lenses [19], which punctually enlarge both visual and motor space to facilitate small target selection through stylus input. However, visual space is enlarged by duplicating the pixels of the original representation. The popup vernier [1] enables users to make precise, sub-pixel adjustments to the position of objects by transitioning from coarse to fine-grain dragging mode through an explicit mode switch. The technique provides visual feedback based on the metaphor of vernier calipers to make precise adjustments between both scales.

LENSES WITH HIGH-PRECISION MOTOR CONTROL

The quantization effect is due to the mismatch between visual and motor space precision in the focus region. This mismatch, in turn, is caused by the following two properties of conventional lenses:

(P1) the cursor is located at the center of the lens, and

(P2) the cursor location is controlled in context space.

These properties cause problems with the two low-level actions performed by users: *focus targeting*, and *object selection* within the magnified region. In this section we introduce three techniques that address these problems by breaking the above properties.

For all our techniques, lens displacements of less than MM focus pixels, corresponding to displacements of less than 1 context pixel, are achieved by slightly moving the representation in the focus region while keeping the cursor stationary (see discussion of Experiment 2's results for more detail).

Precision through Mode Switching: the *Key* technique

The first approach to address the problem is to provide a way of controlling the location of the lens in focus space (as opposed to context space). We immediately discard the solution that consists in solely interacting in focus space because of obvious performance issues to navigate moderate to large distances (all distances are multiplied by MM in focus space). The simplest technique uses two control modes: a *context speed* mode and a *focus speed* mode. This requires an additional input channel to perform the mode switch, for instance using a modifier key such as SHIFT. Users can then navigate large distances at *context speed*, where one input device unit is mapped to one context pixel, i.e., MM focus pixels, and perform precise adjustments at *focus speed*, where one input device unit corresponds to one focus pixel.

Figure 3 illustrates this technique, called *Key*: the first case (No modifier) is represented by the topmost grey line; the second case (Shift pressed) by the bottommost grey line. When SHIFT is pressed, (P2) is broken. A similar "precision mode" is already available in, e.g., Microsoft Office to freely position objects away from the intersections formed by the underlying virtual grid using a modifier key.

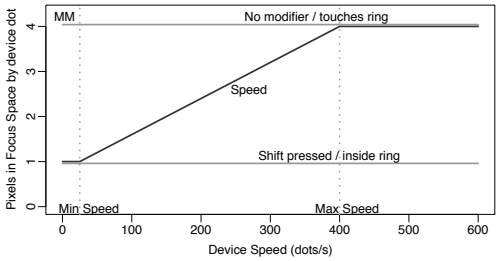


Figure 3. Displacement in focus space (in pixels) for one input device unit move in function of the input device speed ($MM = 4$).

The *Key* technique represents a simple solution. However, as the selection tools based on Magic Lenses [3], an additional channel is required to make the explicit mode switch. Bi-manual input techniques are still uncommon. Modifier keys tend to be used for other purposes by applications, and their use often results in a "*slightly less than seamless interaction style*" [2]. The next two techniques we propose do not require any additional input channel.

Speed-dependent Motor Precision: the *Speed* technique

Following recent works that successfully used speed-dependent properties to facilitate pointing [5] and multi-scale navigation [12, 14, 18], our first idea was to map the precision of the lens control to the input device's speed with a *continuous* function, relying on the assumption that a high speed is used to navigate large distances while a low speed is more characteristic of a precise adjustment (as observed for classical pointing [2]).

The black line (Speed) in Figure 3 illustrates the behavior of our speed-dependent precision lens. Cursor instant speed s is computed as the mean speed over the last four move events. It is mapped to the lens' speed so as to break (P2) as follows:

(i) if $s < MIN\_SPEED$ then the lens moves at *focus speed* ;

(ii) if $MIN\_SPEED \leq s \leq MAX\_SPEED$ then the lens moves by x focus-pixels for 1 input device unit, where x is $1 + (1 - \frac{MAX\_SPEED - s}{MAX\_SPEED - MIN\_SPEED}) \times (MM - 1)$;

(iii) if $s > MAX\_SPEED$ then the lens moves at *context speed* like a conventional lens.

Cursor-in-flat-top Motor Precision: the *Ring* technique

The last technique is inspired by Tracking menus [10]. Consider a large rigid ring (e.g., a bracelet) on a flat surface (e.g., a desk). The ring can be moved by putting a finger inside it and then moving that finger while keeping it in contact with the surface to pull the ring. This is the basic metaphor used to interact with the *Ring* lens: the ring is the lens' focus region (called the *flat-top*) and the cursor is the finger.

The *Ring* lens breaks property (P1): it decouples the cursor from the lens center; the cursor can freely move *within* the flat-top at *focus scale*, thus enabling pixel-precise pointing in the magnified region (bottommost grey line (Inside ring) in Figure 3). When the cursor comes into contact with the flat-top's border, it pulls the lens at *context speed*, enabling fast repositioning of the lens in the information space (topmost

grey line (Pushing ring) in Figure 3). Figure 5 illustrates the lens behavior when the cursor comes into contact with the ring: the segment joining the lens center (g) to the contact point (p) is progressively aligned with the cursor's direction.

Decoupling the cursor's location from the lens' center has a drawback when changing direction: because the user has to move the cursor to the other end of the flat-top before she can pull the lens in the opposite direction. We tried to address this issue by pushing the physical metaphor: we introduced friction in the model to make the ring slide when the cursor stops, with the effect of repositioning the lens' center so as to match the cursor's position. We were not able however to get a satisfying result, and abandoned the idea.

EXPERIMENTS

We conducted two experiments to compare the performance and limits of the three lenses described above. Participants were asked to perform a simple task: selecting an object in the magnified area. The targets were laid out in a circular manner and the order of appearance forced participants to perform the task in every direction, following the recommendations of the ISO 9241-9 standard [9]. Only one target was visible at a time so that participants could not take advantage of the layout to facilitate the task: as soon as the participant clicked on one target, the next target appeared. The recorded movement time is the interval between the appearance of the target and a click on it. The target is presented as a yellow circle on a gray background, and is always surrounded by a 10-pixel red square clearly visible in the context view. The background is also decorated by a grid to help participants understand the transition between context and focus view, and to minimize desert fog effects [15] that can occur with scenes that are too uniform.

Analysis of the Task

A *pointing task* with a lens is typically divided in two main phases: (i) *focus targeting*, which consists in putting a given target inside the flat-top of the lens (Figure 4-(a) and (b)) and (ii) *cursor pointing* to precisely position the cursor over the target (Figure 4-(b) and (c)).

The *focus targeting* task has an index of difficulty of about:

$$ID_{FT} = log_2(1 + \frac{D_c}{(W_{FT_c} - W_c)})$$

where W_{FT_c} and W_c are the respective sizes of the flat-top and the target in context pixels, and D_c is the distance to the target in context pixels as well[1]. This formula clearly shows that difficulty increases as distance increases, as the size of the flat-top decreases, and as the size of the target decreases. The size of the flat-top in context pixels is directly related to the magnification factor of the lens, MM. Indeed, the size of the flat-top is fixed in terms of focus pixels, so the higher MM, the smaller the size of the magnified area in context pixels (see [18] for an analysis of the difficulty of a focus targeting task).

[1] ID_{FT} is the exact index of difficulty when the target must be fully contained in the flat-top. Here the task is slightly easier because the target just has to intersect the flat-top.

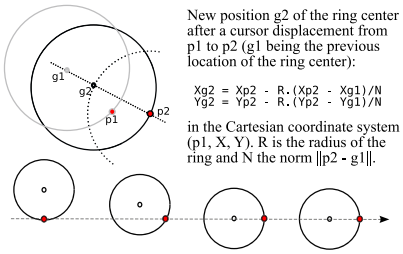


Figure 5. Bottom: behavior of a *Ring* lens when the cursor comes into contact with the flat-top's border at the bottom of the ring and then moves to the right. Top: Computation of the ring's location.

The final *cursor pointing* task mainly depends on the area of the target in focus space that intersects the flat-top after the focus targeting task. The larger this area, the easier the cursor pointing task. We can at least consider the best case, i.e., when the target is fully contained in the flat-top. In this case, the difficulty of the cursor pointing task can be assessed by the ratio $\frac{D_f}{W_f}$ where D_f is the distance between the cursor and the target, and W_f is the motor size of the target when magnified in the flat-top. The distance D_f is small, i.e., smaller than the flat-top's diameter, so we assume that the difficulty of the cursor pointing task is mainly caused by the value of W_f. Note that for regular lenses, the value of W_f is actually the size of the target at context scale because the target is only visually magnified. With our lenses however, since pixel-precise selections are possible, W_f is the magnified size of the target (at focus scale). We provide additional details about the division between the two subtasks in the following sections.

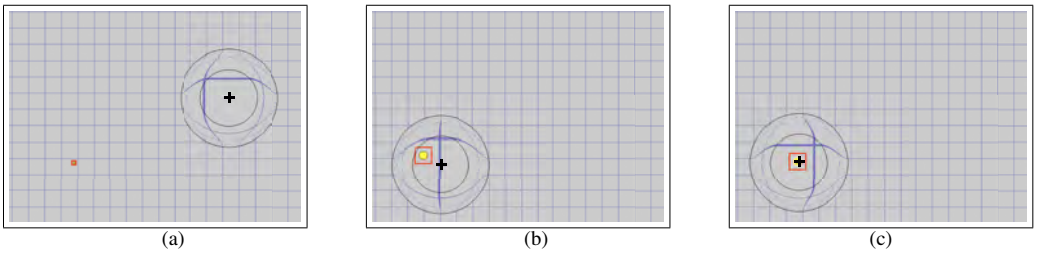
The first experiment tests pointing tasks with an average level of difficulty, while the second one tests pointing tasks with a very high level of difficulty, involving targets smaller-than-a-pixel wide at context scale. Our experimental design involves the three factors that determine the pointing task difficulty introduced above: the distance to the target (Dc), its width (Wc), and the lens' magnification factor MM.

Experiments: Apparatus

We conducted the experiments on a desktop computer running Java 1.5 using the open-source ZVTM toolkit. The display was a 21" LCD monitor with a resolution of 1600 x 1200 ($\approx$ 100 dpi). The mouse was a regular optical desktop mouse at 400 dpi with the default acceleration function.

Experiment 1: Design

The goal of the first experiment is to test whether any of the three techniques we introduced in the previous section degrade performance when compared with regular lenses (*Reg*). We expect them to improve overall performance because the overall task difficulty is theoretically lower. On the one hand, the focus targeting task should not be harder: since we test small targets with lenses having the same flat-top size, the distance in context space is the main factor contributing to difficulty. All our lenses are able to navigate large distances like a regular lens, i.e., move at *context speed* (*Key*:



(a) (b) (c)

Figure 4. Screenshots of our experimental task: *focus targeting* from (a) to (b) and, *cursor pointing* from (b) to (c). Screenshots have been cropped to show details, and cursors have been made thicker to improve readability.

when SHIFT is released; *Ring*: when the cursor pulls the lens; *Speed*: when the lens moves fast enough). On the other hand, cursor pointing should be easier since the difficulty of this second phase mainly depends on the target's motor width in focus space. Since all of our lenses allow to navigate at *focus speed*, they can take benefit of the magnified target size whereas this is not the case with a regular lens: even though it is magnified, the target size in motor space is the same as if it were not magnified.

Sixteen unpaid volunteers (14 male, 2 female), age 20 to 35 year-old (average 26.8, median 26), all with normal or corrected to normal vision, served in Experiment 1. Experiment 1 was a $4 \times 2 \times 2 \times 3$ within-subject design with the following factors:

- Technique: TECH $\in \{Speed, Key, Ring, Reg\}$
- Magnification: MM $\in \{4, 8\}$
- Distance between targets (context pixels): DC $\in \{400, 800\}$
- Target width (context pixels): WC $\in \{1, 3, 5\}$

We grouped trials into four blocks, one per technique (TECH), so as not to disturb participants with too many changes between lenses. The presentation order was counterbalanced across participants using a Latin square. Within a TECH block, each participant saw two sub-blocks, one per value of magnification factor (MM). The presentation order of the two values of MM was also counterbalanced across techniques and participants. For each TECH $\times$ MM condition, participants experienced a series of 12 trials per DC $\times$ WC condition, i.e., 12 targets laid out in a circular pattern as described earlier. We used a random order to present these $2 \times 3 = 6$ series within a sub-block. We removed the first trial of each series from our analyses as the cursor location is not controlled when a series begins. To summarize, we collected 4 TECH $\times$ 2 MM $\times$ 2 DC $\times$ 3 WC $\times$ (12-1) replications $\times$ 16 participants = 8448 trials for analysis. Before each TECH condition, the experimenter took 2-3 minutes to explain the technique to be used next. Participants were told each time the value of MM was about to change, and had to complete 4 series of practice trials for each new TECH $\times$ MM condition.

Experiment 1: Results and Discussion

Our analysis is based on the full factorial model:

TECH$\times$MM$\times$WC$\times$DC$\times$Random(PARTICIPANT)

with the following measures:

- *FTT*, the focus targeting time;
- *CPT*, the cursor pointing time;
- $MT = FTT + CPT$, the time interval between the appearance of the target and a successful mouse press on it (this measure includes penalties caused by errors); and
- *ER*, the error rate (an error is a press outside the target).

Analysis of variance reveals an effect of TECH on *MT* ($F_{3,45} = 15.2, p < 0.0001$). A Tukey post-hoc test shows that *Reg* is the significantly slowest technique and that *Key* is significantly faster than *Ring*. Note that there is no significant difference between *Ring* and *Speed*, nor between *Speed* and *Key*. Participants also made more errors with *Reg* than with our techniques. We expected *Reg* to perform worse since, as we already mentioned, the target's motor size is in context pixels for *Reg* whereas it is in focus pixels for *Key*, *Speed* and *Ring*. The target is thus much harder to acquire in the *CPT* phase. Analysis of variance shows a significant effect of TECH ($F_{3,45} = 18.5, p < 0.0001$) on *ER*. Figures 6-(a) and (b) respectively show the time *MT* and error rate *ER* for each TECH$\times$WC condition.

We find a significant effect of DC ($F_{1,15} = 121.9, p < 0.0001$) on movement time *MT*. It is consistent with our expectations: DC has a significant effect on *FTT* ($F_{1,15} = 165, p < 0.0001$) while it does not on *CPT* (p=0.4). The higher the value of DC, the harder the focus targeting phase. Our techniques do not seem to be at a disadvantage in this phase compared to *Reg* since the effect of DC$\times$TECH on *FTT* is not significant (p=0.9).

MM also has a significant effect on *MT* ($F_{1,15} = 249.6, p < 0.0001$), the effect being distributed across both *FTT* ($F_{1,15} = 515, p < 0.0001$) and *CPT* ($F_{1,15} = 79, p < 0.0001$). Figure 6-(c) shows the three measures per TECH$\times$MM: a bar represents *MT* per condition while the line shows the repartition between *FTT* (lower part of the bar) and *CPT* (upper part)[2]. This clearly shows that a high MM leads to high *FTT* since the flat-top size in context pixels directly depends on MM, as explained in the previous section. A higher MM also means a larger target width in focus pixels. This can explain the effect of MM on *CPT*: *CPT* decreases as MM increases.

The target width in focus pixels is of course also related to WC, which is consistent with our observations: WC has an effect on both (i) *FTT* ($F_{2,30} = 45, p < 0.0001$) and (ii) *CPT* ($F_{2,30} = 1110, p < 0.0001$), and also on *MT* ($F_{2,30} =$

[2]Error bars in the figures represent the 95% confidence limits of the sample mean ($mean \pm StdErr \times 1.96$).

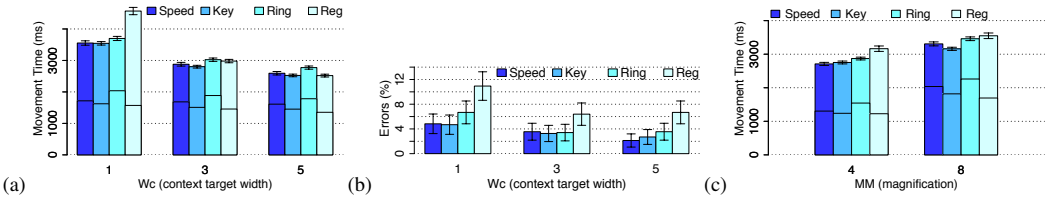


Figure 6. Movement time (a) and error rate (b) per TECH × WC. (c) Movement time per TECH × MM. For (a) and (c), the lower part of each bar represents focus targeting time, the upper part cursor pointing time.

$623.8, p < 0.0001$, Figure 6-(a)). Indeed, as we expected, the smaller WC, the higher the focus targeting time (i). Also, the larger WC, the larger the target in focus pixels to improve focus pointing time (ii). Regarding error rate, WC ($F_{2,30} = 17.5, p < 0.0001$) and MM ($F_{1,15} = 16.8, p = 0.0009$) have a significant effect on *ER*: participants made more errors when the target size was small. This is a simple interpretation that explains the difference in means that we observe; but we have to refine it to reflect the more complex phenomenon that actually takes place.

Coming back to the effect of TECH, we also observe two significant interaction effects that involve TECH on *MT*.

First interaction effect: TECH×MM ($F_{3,45} = 4.7, p = 0.0063$) which can be observed on Figure 6-(c). A Tukey post-hoc test shows that for MM = 4, *Speed*, *Key* and *Ring* are significantly faster than *Reg* but this test also shows that for MM = 8, only *Key* and *Speed* are significantly faster than *Reg* (*Ring* no longer is). A closer look at the focus targeting phase explains why *Ring* seems to suffer from high magnification factors. We know that *FTT* increases as MM increases. We can observe on Figures 6-(c) and (a) that *Ring* is actually slower than the other techniques for this *FTT* phase. This is probably due to the cost of repairing overshoot errors during this phase: changes in direction are costly with *Ring* since the user first has to move the cursor to the opposite side of the flat-top before being able to pull the lens in the opposite direction.

Second interaction effect: TECH×WC ($F_{6,90} = 55.1, p < 0.0001$) which can be observed on Figure 6-(a). A Tukey post-hoc test shows a significant difference in mean for WC=1 between *Reg* and the other techniques, while this difference is not significant for WC=3 and WC=5. To better assess the interpretation of such a result, we consider finer analyses on *CPT*. Figure 7 shows *CPT* for each TECH×MM×WC condition. Analyses reveal significant effects of TECH, MM and WC and significant interactions TECH×MM and TECH×WC (all $p < 0.0001$) on *CPT*. Tukey post-hoc tests show that *Key*, *Speed* and *Ring* are globally faster than *Reg* for cursor pointing. This is not surprising since the motor size of the target is smaller for *Reg* than for the others, as we said earlier. However, this significant difference holds only for WC=1 and WC=3, not for WC=5. In the latter case, only *Speed* is significantly faster than *Reg*. Moreover *Ring* is faster than *Key* for WC= 1, while *Speed* is not. These results suggest that *Ring* is particularly efficient for very small targets and that *Speed* is more appropriate for larger ones.

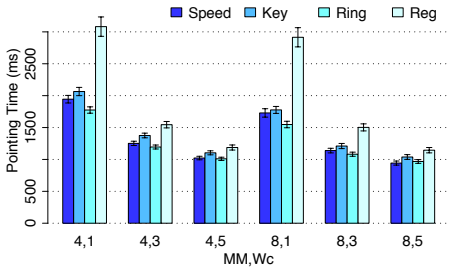


Figure 7. Cursor pointing time per TECH × MM × WC condition.

The latter observations suggest that modeling the movement time *MT* as the sum of *FTT* and *CPT* (*MT=FTT+CPT*) may be too naive to explain the subtle differences between techniques. For instance, this model does not explain the differences between *Ring* and *Speed* that depend on WC. In the same spirit, we observe that the difference between *Reg* and other lenses for WC=5 is very small considering that the target's motor size is 5 for *Reg* and 20 (MM=4) or 40 (MM=8) for *Key*, *Speed* and *Ring*. The additive model based also fails to explain the following observation: *Speed* features significantly higher *FTT* values than *Key* and *Reg* for MM=8 only. We tentatively explain this by the increased difficulty of controlling a lens with speed-dependent precision when the slope of the mapping function is too steep (linear function from MIN_SPEED to MAX_SPEED, i.e., *focus speed* to *context speed* on Figure 3). We tried several variations that, e.g., depend on the difference between these two speeds, without success. Using a gentler slope is frustrating because of the stickiness caused by the large movements required to reach the MAX_SPEED threshold. The more subtle differences we reported in the second part of this section may be explained by the fact that a *transition* phase between the focus targeting phase (*FTT*) and the cursor pointing phase (*CPT*) actually exists for our lenses: pressing a key for *Key*, stop pulling the flat-top for *Ring*, performing speed adjustments with *Speed*.

At the end of the experiment, participant were asked to rank the lenses (with ex-aequo allowed) using two criteria: perceived usability and performance. These two rankings were almost the same for all participants. All but one ranked *Reg* as their least preferred technique (one participant ranked it third with *Speed* fourth). There was no significant difference among other lenses. For instance, 8 participants ranked

Speed first, 3 ranked it second; 6 participants ranked *Key* first, 5 ranked it second, and 5 participants ranked *Ring* first, 7 ranked it second. We also asked participants to comment on the techniques. The main reason for the bad ranking of *Reg* is the great difficulty to acquire small targets, related to the *cursor jumping* effect due to quantization. Regarding *Speed*, most participants found the technique *"natural"*; some found the speed *"difficult to control"*. The participants who ranked *Key* high justified it by a *"transparent control"*; other participants complained about the need to use two hands. Regarding *Ring*, the cursor pointing phase was found easier because the lens is stationary, but participants also raised the overshooting problem discussed earlier.

To summarize, in comparison with regular lenses, precision lenses increase pointing accuracy. They also increase selection speed for small targets and are as fast for larger ones.

Experiment 2: Design

This second experiment evaluates our techniques on extreme tasks: very small target sizes and high magnification factors. We discard the *Reg* technique as it is not capable of achieving sub-pixel pointing tasks, i.e., involving targets that are smaller-than-a-pixel wide in context space. Another difference with Experiment 1 is that we use WF as a factor instead of WC. This allows us to isolate the effects of WF and MM. Indeed, since WF = WC × MM, two values of MM correspond to two different values of WF for the same WC value.

Twelve participants from Experiment 1 (10 male, 2 female), age 20 to 35 year-old (average 27.25, median 26.5), also served in Experiment 2. Experiment 2 was a 3 × 2 × 2 × 3 within-subject design with the following factors:

- TECH ∈ {*Speed*, *Key*, *Ring*}
- MM ∈ {8, 12}
- DC ∈ {400, 800}
- WF ∈ {3, 5, 7}

As in Experiment 1, trials were blocked by technique, with presentation order counterbalanced across participants using a Latin square. The experimenter explained the technique to be used during 2-3 minutes before each TECH condition. For each TECH, participants saw the two values of MM, grouped into two sub-blocks (sub-block presentation order were counterbalanced across techniques and participants). Each sub-block contained 6 series of 8 trials, 1 series per DC × MM condition, presented in a random order. To summarize, we collected 3 TECH × 2 MM × 2 DC × 3 WC × (8-1) replications × 12 participants = 3024 trials for analysis. As in Experiment 1, participants were alerted by a message each time the MM value changed and had to complete 4 practice series for each TECH × MM condition.

Experiment 2: Results and Discussion

Our analysis is based on the full factorial model:

$$\text{TECH} \times \text{MM} \times \text{WF} \times \text{DC} \times \text{Random}(\text{PARTICIPANT})$$

We consider the same measures as in Experiment 1: task completion time *MT*, focus targeting time *FTT*, cursor pointing time *CPT* and error rate *ER*.

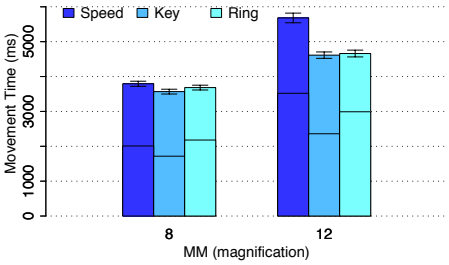


Figure 8. Movement time per TECH×MM. The lower part of each bar represents focus targeting time, the upper part cursor pointing time.

Analysis of variance reveals simple effects of WF ($F_{2,22} = 68$), MM ($F_{1,11} = 393$) and DC ($F_{1,11} = 65$) on *MT* (all $p < 0.0001$). As expected, *MT* increases as WF decreases, as MM increases and as DC increases. Participants also make significantly more errors when WF decreases (3.67% for WF = 7, 5.36% for WF = 5 and 8.82% for WF = 3).

The differences in movement time *MT* among techniques is significant ($F_{2,22} = 21.6, p < 0.0001$) while the difference in error rate is not (6.15% for *Speed*, 6.05% for *Key* and 5.65% for *Ring*).

There is an interaction effect TECH×MM on *MT* ($F_{2,22} = 24.8, p < 0.0001$): Tukey post-hoc tests show that *Ring* and *Key* are significantly faster than *Speed* but only for MM=12 while these differences are not significant for MM=8. Figure 8 shows that this large difference at MM=12 is due to a sharp increase of focus targeting time (*FTT*) for *Speed*. Comments from participants confirm that the speed dependent control of motor precision is too hard when the difference between context scale and focus scale is too high, resulting in abrupt transitions. With *Speed*, participants did not succeed in controlling their speed: either they overshot the target (targeting speed too high) or spent a lot of time putting the target in focus (speed too low). Therefore, *Speed* does not seem to be a suitable lens for pointing with a very high magnification factor: at MM=12, the linear function linking focus speed to context speed is too steep to be usable.

Figure 8 shows that focus targeting performance of *Ring* degrades as MM increases. However, good cursor pointing performance compensates for it, resulting in good overall task completion time. Figure 9 shows *CPT* for each TECH × MM × WC condition. Analysis of variance reveals a significant effect of WF ($F_{2,22} = 230, p < 0.0001$) on *CPT*. As mentioned earlier, the larger WF, the easier the cursor pointing task. However, the effects of MM ($F_{1,11} = 154, p < 0.0001$) and TECH ($F_{2,22} = 64, p < 0.0001$) on *CPT* are less straightforward to interpret. *CPT* is higher when MM=12 than when MM=8, *Ring* is faster than *Key* and *Speed*, and the difference between *Ring* and both *Key* and *Speed* is larger when MM=12 than when MM=8 (the TECH×MM interaction is indeed significant on *CPT*, $F_{2,22} = 9.8, p = 0.0009$).

A plausible explanation for these effects lies in the differences in terms of Control-Display (C-D) gain among tech-

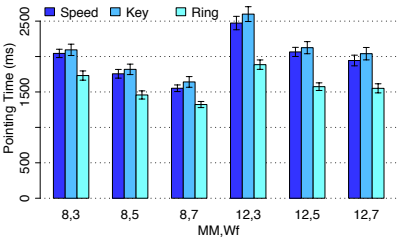


Figure 9. Cursor pointing time per TECH × MM × WF condition.

niques in the cursor pointing phase[3]. Figure 10 illustrates the difference in terms of control-display gain among lenses, all in high-precision mode. During the cursor pointing phase, *Ring* is stationary; only the cursor moves inside a static flat-top. This is not the case for *Key* and *Speed* for which high-precision cursor pointing is achieved through a combination of cursor movement and flat-top offset. In Figure 10, to achieve a mouse displacement of 15 units, the cursor has moved by 1 context pixel (= 8 focus pixels) and the representation has moved by 7 focus pixels to achieve an overall displacement of 15 focus pixels. As a result, the control-display gain is divided by MM for *Key* and *Speed*. This might be the cause for the observed performance degradation. This interpretation is consistent with the stronger degradation for *Key* and *Speed* than for *Ring* from MM=8 to MM=12. Note, however, that there is still a small degradation of *CPT* from MM=8 to MM=12 for *Ring*, that we tentatively explain by a harder focus targeting phase when MM=12 that influences the transition from focus targeting to cursor pointing.

To summarize, when pushed to extreme conditions, the *Speed* lens becomes significantly slower than the other precision lenses while *Ring* remains as fast as *Key* without requiring an additional input channel for mode switching.

MOTOR CONTROL COMBINED WITH VISUAL FEEDBACK

Previous experiments show that techniques with advanced *motor* behaviors enable higher-precision focus targeting and object selection while increasing the upper limit of usable magnification factors. The Sigma Lens framework [18] takes a different approach at solving the same general problem by proposing advanced *visual* behaviors. We now explore how to combine these two orthogonal approaches to create hybrid lenses that further improve performance.

Sigma Lenses with High-Precision Motor Control

The two Sigma lens visual designs reported as the most efficient ones in [18] can be directly combined with our motor designs. The first one is the *Speed-coupled blending* (abbreviated *Blend*): it behaves as a simple magnifying glass whose translucence varies depending on lens speed. Smooth transition between focus and context is achieved through dynamic alpha blending instead of distortion. This enables a larger flat-top for the same overall lens size, reducing the

[3]The ratio between the distances traveled by the *cursor* and the *input device*, both expressed in metric units.

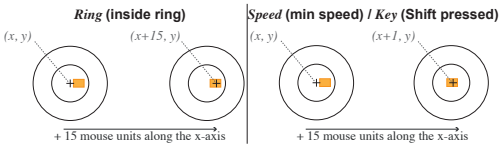


Figure 10. Difference in control-display gain between *Ring* and *Speed/Key* lenses (MM=8). In italic: cursor location on screen.

focus targeting task's index of difficulty. The other design (abbreviated *Flat*) is a variation on Gutwin's original *Speed-coupled flattening* [12]. The lens flattens itself into the context as its speed increases so as to eliminate the problems caused by distortion. Figure 11 illustrates both behaviors.

We designed four new techniques that result from the combination of one of the above two visual behaviors with either speed-dependent motor precision (*Speed*) or cursor-in-flat-top motor precision (*Ring*). *Key* was discarded because it proved awkward to combine explicit mode switching with speed-dependent visual properties.

Speed + Flat: this lens behaves like the original *Speed* design, except that the magnification factor decreases toward 1 as speed increases (Figure 11-a). The main advantage is that distortion no longer hinders focus targeting. Additionally, flattening provides indirect visual feedback about the lens' precision in motor space: it operates in context space when flattened, in focus space when not flattened.

Ring + Flat: This lens behaves like the original *Ring* design, with the magnification factor varying as above. As a consequence, the flat-top shrinks to a much smaller size (time stamp *t3* on Figure 11-a), thus making course corrections during focus targeting easier since the cursor is still restricted to that area. As above, distortion is canceled during focus targeting.

Ring + Blend: This distortion-free lens behaves like the original *Ring* design, except that the restricted area in which the cursor can evolve (the flat-top) is larger (time stamps *t1* and *t5* in Figure 11-b). As speed increases, the flat-top fades out, thus revealing the context during the focus targeting phase (time stamps *t2* to *t4*). An inner circle fades in, representing the region that will actually be magnified in the flat-top if the lens stops moving. The cursor is restricted to that smaller area, making course corrections less costly.

Speed + Blend: This lens behaves like the original *Speed* design without any distortion. As above, the flat-top fades out as speed increases and fades back in as speed decreases. Again, the larger flat-top reduces the focus targeting task's index of difficulty. In a way similar to *Speed + Flat*, blending provides indirect visual feedback about the lens' precision in motor space: it operates in context space when transparent, in focus space when opaque.

Experiment 3: Design

Our goal is to evaluate the potential benefits of combining techniques that enable higher motor precision with visual behaviors based on speed-coupling. We use *Static* versions,

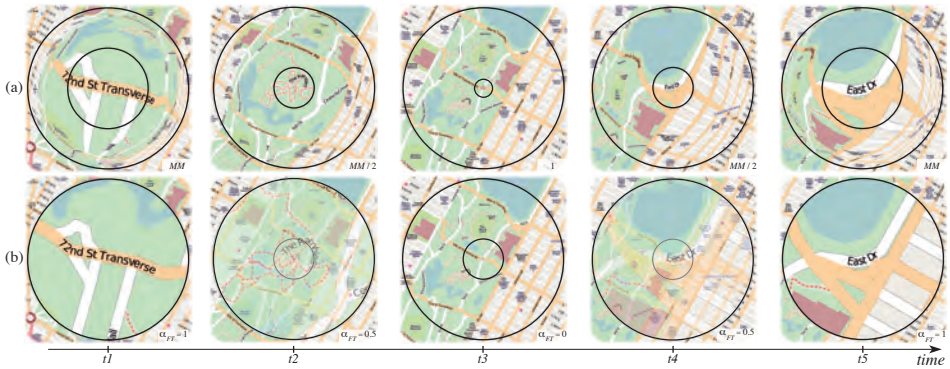


Figure 11. Behavior of two Sigma lenses during a focus targeting task ending on *East Drive* in Central Park. (a) As speed increases, the *speed-coupled flattening lens* smoothly flattens itself into the context (from *t1* to *t3*), and gradually reverts to its original magnification factor when the target has been reached (*t4* and *t5*). The inner circle delimits the region magnified in the flat-top. (b) As speed increases, the *speed-coupled blending lens* smoothly fades into the context (from *t1* to *t3*), and gradually fades back in when the target has been reached (*t4* and *t5*). The inner circle fades in as the lens fades out; it delimits which region of the context gets magnified in the lens. The magnification factor remains constant.

i.e., without any dynamic visual behavior, of our *Ring* and *Speed* techniques as baselines. Experiment 2 revealed that problems arise for the *difficult* tasks. We thus consider here difficult conditions in terms of magnification and target size. To reduce the length of the experiment, we discarded the DC factor (distance between targets) as it did not raise any particular issue for any of the techniques.

Twelve participants from the previous experiments served in Experiment 3. Experiment 3 was a $2 \times 3 \times 2 \times 3$ within-subject design with the following factors:

- Motor precision technique: TECH $\in \{Speed, Ring\}$
- Visual behavior: VB $\in \{Blend, Flat, Static\}$
- Magnification: MM $\in \{8, 12\}$
- Target width in focus pixels: WF $\in \{3, 7, 15\}$

Trials were grouped into two main blocks, one per technique (TECH). These blocks were divided into three secondary blocks, one per visual behavior. The presentation order of TECH main blocks and VB secondary blocks was counterbalanced across participants using a Latin square. Within a TECH×VB block, each participant saw two sub-blocks, one per magnification factor (MM); presentation order was counterbalanced as well. For each TECH × VB × MM condition, participants experienced 3 series of 8 trials, one per value of WF, presented in a random order. We collected 2 TECH × 3 VB × 2 MM × 3 WC × (8-1) replications × 12 participants = 3024 trials for analysis. As with the other two experiments, participants received a short explanation before each TECH × VB condition and performed 3 practice trial series per TECH × VB × MM condition.

Experiment 3: Results and Discussion

As in Experiments 1 and 2, we perform analyses of variances with the full factorial model VB × TECH × MM × WC × Random(PARTICIPANT) for *MT*, *FTT*, *CPT* and *ER*. Tukey post-hoc tests are used for pairwise comparisons.

As expected, we find a simple effect of VB on *MT* ($F_{2,22} = 67, p < 0.0001$) revealing that visual behaviors significantly improve overall performance. Even if *CPT* is significantly degraded, the gain in *FTT* is strong enough (significantly) to decrease *MT* (see Figure 12). The degraded cursor pointing performance observed here is not surprising. It can be explained by the time it takes for a speed-coupled blending lens to become opaque enough or for a speed-coupled flattening lens to revert to its actual magnification factor. The performance gain measured for the focus targeting phase is consistent with previous experimental results [12, 18]. Overall, the gain in the focus targeting phase is strong enough to improve overall task performance.

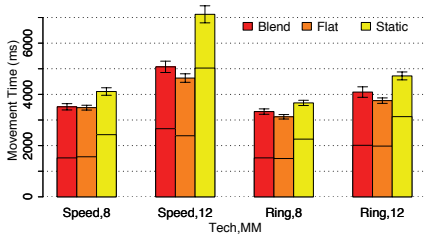


Figure 12. Movement time (*MT*) per VB by TECH × MM condition. The lower part of each bar represents focus targeting time (*FTT*), the upper part cursor pointing time (*CPT*).

The effects of WF and MM on *MT* are consistent with the previous two experiments: *MT* increases as WF decreases and as MM increases. *Ring* is still significantly faster than *Speed* (TECH has a significant effect on *MT*: $F_{1,11} = 153, p < 0.0001$). Even if visual speed-coupling improves the performance of *Speed* more than that of *Ring* (significant interaction effect of TECH×VB on *MT*: $F_{1,11} = 11, p = 0.0005$), *Ring* remains faster than *Speed* for each MM. However, the advantage of *Ring* over *Speed* is significant only for MM=12 when we consider only the two speed-coupling techniques (TECH×MM on *MT* is significant, $F_{1,11} = 227, p < 0.0001$, as well as VB×TECH×MM, $F_{2,22} = 21, p < 0.0001$).

Note that we do not observe a significant advantage of *Blend* over *Flat* as reported in [18]. The main difference is that our targets are much smaller than those tested with Sigma lenses (0.25 to 1.9 context pixels in our experiment vs. 8 context pixels in [18]). Small targets probably cause more overshoot errors that are more expensive to repair with *Blend* than with *Flat*: if the larger flat-top of *Blend* is supposed to make focus targeting easier under an error-free hypothesis, it also causes an area of occlusion that is a significant drawback when trying to correct overshoots. Our participants actually reported that observation; in case of an overshoot they often left the target zone completely to perform a new focus targeting task. However this interpretation should be taken carefully since we did not record the number of overshoot errors. We only measured *ER*, the percentage of clicks outside the target (5.15% for *Blend*, 5.55% for *Flat* and 4.36% for *Static*). As in Experiment 2, the only factor that has an effect on error rate is target width W_F.

SUMMARY AND FUTURE WORK

Large differences in scale between focus and context views cause a quantization problem that makes it difficult to precisely position lenses and to acquire small targets. Quantization severely limits the range of magnification factors that can be used in practice. We have introduced three high-precision techniques that address this problem, making focus targeting and object selection more efficient while allowing for higher magnification factors than regular lenses. This is confirmed by the results of our evaluations, which also reveal that some lenses are more robust than others for extreme conditions, with the *Ring* technique performing the best. Our high-precision techniques can be made even more efficient by combining them with speed-dependent visual behaviors drawn from the Sigma lens framework, as shown in the last experiment.

We analyzed our observations based on a model for target acquisition that sums the *focus targeting* and *cursor pointing* time to get the overall task time. Our results suggest that this model is too simple as it ignores the transition period between the two subtasks. This is especially true for lenses with a speed-dependent behavior, because of the delay to revert back to their stationary configuration. As future work we plan to refine the additive model to better account for these transitions. We also plan to adapt our techniques to other focus+context interfaces and investigate non-circular focus shapes.

REFERENCES

1. Y. Ayatsuka, J. Rekimoto, and S. Matsuoka. Popup vernier: a tool for sub-pixel-pitch dragging with smooth mode transition. In *Proc. UIST '98*, 39–48. ACM, 1998.
2. R. Balakrishnan. "Beating" Fitts' law: virtual enhancements for pointing facilitation. *IJHCS*, 61(6):857–874, 2004.
3. E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses: the see-through interface. In *Proc. SIGGRAPH '93*, 73–80. ACM, 1993.
4. S. Carpendale, J. Ligh, and E. Pattison. Achieving higher magnification in context. In *Proc. UIST '04*, 71–80. ACM, 2004.
5. O. Chapuis, J. Labrune, and E. Pietriga. Dynaspot: Speed-dependent area cursor. In *Proc. CHI '09*, 1391–1400. ACM, 2009.
6. A. Cockburn and P. Brock. Human on-line response to visual and motor target expansion. In *Proc. GI '06*, 81–87, 2006.
7. A. Cockburn and A. Firth. Improving the acquisition of small targets. In *Proc. BCS-HCI '03*, 181–196, 2003.
8. A. Cockburn, A. Karlson, and B. B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM CSUR*, 41(1):1–31, 2008.
9. S. A. Douglas, A. E. Kirkpatrick, and I. S. MacKenzie. Testing pointing device performance and user assessment with the ISO 9241, part 9 standard. In *Proc. CHI '99*, 215–222. ACM, 1999.
10. G. Fitzmaurice, A. Khan, R. Pieké, B. Buxton, and G. Kurtenbach. Tracking menus. In *Proc. UIST '03*, 71–79. ACM, 2003.
11. G. W. Furnas and B. B. Bederson. Space-scale diagrams: understanding multiscale interfaces. In *Proc. CHI '95*, 234–241. ACM & Addison-Wesley, 1995.
12. C. Gutwin. Improving focus targeting in interactive fisheye views. In *Proc. CHI '02*, 267–274. ACM, 2002.
13. K. Hornbæk, B. B. Bederson, and C. Plaisant. Navigation patterns and usability of zoomable user interfaces with and without an overview. *ACM ToCHI*, 9(4):362–389, 2002.
14. T. Igarashi and K. Hinckley. Speed-dependent automatic zooming for browsing large documents. In *Proc. UIST '00*, 139–148. ACM, 2000.
15. S. Jul and G. W. Furnas. Critical zones in desert fog: aids to multiscale navigation. In *Proc. UIST '98*, 97–106. ACM, 1998.
16. J. Lamping and R. Rao. Laying out and visualizing large trees using a hyperbolic space. In *Proc. UIST '94*, 13–14. ACM, 1994.
17. M. J. McGuffin and R. Balakrishnan. Fitts' law and expanding targets: Experimental studies and designs for user interfaces. *ACM ToCHI*, 12(4):388–422, 2005.
18. E. Pietriga and C. Appert. Sigma lenses: focus-context transitions combining space, time and translucence. In *Proc. CHI '08*, 1343–1352. ACM, 2008.
19. G. Ramos, A. Cockburn, R. Balakrishnan, and M. Beaudouin-Lafon. Pointing lenses: facilitating stylus input through visual-and motor-space magnification. In *Proc. CHI '07*, 757–766. ACM, 2007.
20. M. Sarkar, S. S. Snibbe, O. J. Tversky, and S. P. Reiss. Stretching the rubber sheet: a metaphor for viewing large layouts on small screens. In *Proc. UIST '93*, 81–91. ACM, 1993.
21. J. J. van Wijk and W. A. Nuij. A model for smooth viewing and navigation of large 2d information spaces. *IEEE TVCG*, 10(4):447–458, 2004.
22. C. Ware and M. Lewis. The DragMag image magnifier. In *Proc. CHI '95*, 407–408. ACM, 1995.

Reciprocal Drag-and-Drop

CAROLINE APPERT and OLIVIER CHAPUIS, CNRS & Univ. Paris-Sud, Inria
EMMANUEL PIETRIGA, Inria, Inria Chile, Univ. Paris-Sud & CNRS
MARÍA-JESÚS LOBO, Inria, Univ. Paris-Sud & CNRS

29

Drag-and-drop has become ubiquitous, both on desktop computers and touch-sensitive surfaces. It is used to move and edit the geometry of elements in graphics editors, to adjust parameters using controllers such as sliders, or to manage views (e.g., moving and resizing windows, panning maps). Reverting changes made via a drag-and-drop usually entails performing the reciprocal drag-and-drop action. This can be costly as users have to remember the previous position of the object and put it back precisely. We introduce the $D_N D^{-1}$ model that handles all past locations of graphical objects. We redesign the Dwell-and-Spring widget to interact with this history, and explain how applications can implement $D_N D^{-1}$ to enable users to perform reciprocal drag-and-drop to any past location for both individual objects and groups of objects. We report on two user studies, whose results show that users understand $D_N D^{-1}$, and that Dwell-and-Spring enables them to interact with this model effectively.

Categories and Subject Descriptors: H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Graphical user interfaces*

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Undo model, direct manipulation, Dwell-and-Spring

ACM Reference Format:
Caroline Appert, Olivier Chapuis, Emmanuel Pietriga, and María-Jesús Lobo. 2015. Reciprocal drag-and-drop. ACM Trans. Comput.-Hum. Interact. 22, 6, Article 29 (September 2015), 36 pages.
DOI: http://dx.doi.org/10.1145/2785670

1. INTRODUCTION

Most graphical user interfaces rely heavily on drag-and-drop interactions for view management. Drag-and-drop is the primary method for moving and resizing windows on a desktop, for laying out icons, for panning a map or a very large image, for browsing a long document. But objects manipulated via drag-and-drop often have to be restored to one of their previous positions. For instance, a user will carefully lay out windows on his desktop but will then temporarily move or resize one of them to access content hidden behind it, such as an icon or another window of lesser importance that was left in the background; he will then want to restore the foreground window to its earlier configuration. The reader of a document will scroll down to an appendix or check a reference, and will then want to come back to the section he was reading. Current systems do not enable users to easily restore windows or viewports to their earlier configuration; users have to manually reposition and resize the corresponding objects. Such actions can be costly. From a motor perspective, the cost of *repairing* a drag-and-drop manipulation can be higher than that of the original manipulation depending on

Authors' address: Université Paris Sud - Bat. 660, 91405 ORSAY Cedex, France; emails: {appert, chapuis}@lri.fr, {emmanuel.pietriga, maria-jesus.lobo}@inria.fr.

DOI: http://dx.doi.org/10.1145/2785670

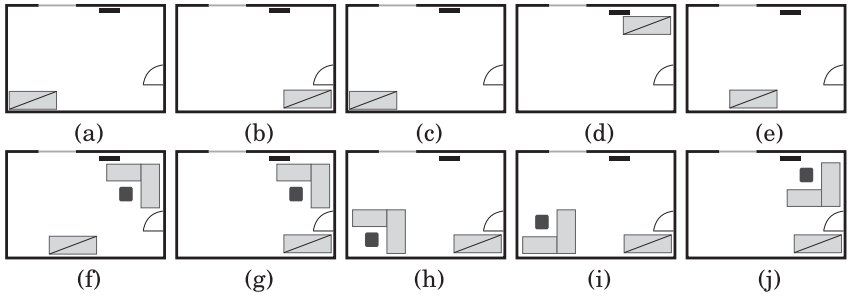


Fig. 1. Exploring different office layout alternatives on a floor plan. (a) Placing a cupboard in the SW corner. (b) When moving the cupboard to the SE corner, it is difficult to access it when the door is open. (c) Cupboard back to the SW corner. (d) Cupboard in the NE corner. The heater is partially occluded. (e) Cupboard almost centered along the S wall. (f) Adding a desk in the NE corner, composed of two tables and a chair. The heater is partially occluded. (g) Cupboard back in the SE corner to free space for the desk in the SW corner. (h) Desk in the SW corner. (i) Changing the relative placement of the desk elements. (j) Desk back in the NE corner with the new relative layout between the two tables and the chair.

how precisely the object has to be positioned. This is especially true for touch-based interfaces, which can make precise manipulations challenging [Siek et al. 2005]. The cost can also be high from a cognitive perspective, as users may have difficulty remembering what was the previous state of a particular object [Katifori et al. 2008].

Users also rely heavily on drag-and-drop for content manipulation in WYSIWYG applications such as vector graphics editors and slide show presentation programs. The precise positioning of elements is particularly important in such contexts, where users perform advanced graphical layout task. But it can be challenging. For instance, graphical shapes vary in their size and may be very close to one another. Accidental selections are likely to occur, and users may want to revert a subset of objects, selected by mistake, back to their original position; and this without having to cancel the manipulation for the objects they actually had intended to move. Some shapes can also overlap other shapes, or even completely cover them. While the shapes below can be visible through alpha blending, they will often be difficult to access. In this situation, users will often temporarily move the shape on top to access the ones below and modify them, but will eventually want to revert to the original layout.

From a user perspective, such graphical layout tasks are often part of an exploratory process. For instance, Figure 1 illustrates a scenario in which a person rearranges furniture in an office and tests alternative layouts. The software allows her to explore different arrangements by selecting and moving either a single piece of furniture, or multiple pieces together. Direct manipulation strongly contributes to making such exploratory design activities easy. But effectively supporting users also entails enabling them to easily revert back to past states from which to try other design options. Most graphical editing software provides an undo command to restore a past state of the entire document but, unfortunately, the underlying undo model is usually a global linear one that does not keep track of branches in the history of manipulations. Such a basic undo mechanism has two strong limitations, as detailed later.

The first limitation is that some previous states in the history can become inaccessible [Berlage 1994; Yoon et al. 2013]. If a user applies a command that turns state A into state B, reverts back to state A using undo, and then applies a new command that turns state A into state C, she will no longer be able to get back to state B. For instance, in Figure 1, the user moves the cupboard (a-b) but then undoes this move (b-c) when she realizes that this location might not be so convenient because of its

proximity to the door. Later, after having considered the different constraints (window, heater, additional furniture), she finally decides that putting the cupboard behind the door [as in (b)] is the best option. She wants to revert it to this location, but as she has moved it to other locations (c-d-e) after her undo operation (b-c), she is no longer able to get back to this configuration other than by manually moving it back there.

The second limitation comes from the lack of integration of object selection mechanisms with the history of direct manipulations. When adjusting layouts, users often want to apply direct manipulation actions to multiple objects simultaneously, typically selected using a rubber-band rectangle or by clicking on all objects in turn while keeping a modifier key pressed. Multiple selection allows users to manipulate groups of objects simultaneously while preserving their relative layout. But this notion of group is transient, as graphical editors usually support only one active selection at a time. Undoing an action performed on multiple objects will no longer be possible once the selection has changed. Users then have to select these objects again, and manually revert them to their earlier position using the reverse drag-and-drop action. Coming back to Figure 1, the user moves the two tables and the chair that make her workstation (g-h), and then changes their relative layout, thus breaking the previous multiple selection (i). Because there can only be one single active selection at a time, testing a location of the workstation that has already been explored (f), but with the new relative layout made in (i-j), requires selecting all its elements again and manually dragging-and-dropping them in the right place. Some graphical editors feature a command to group objects together. But this makes the exploratory design process much more cumbersome, as groupings have to be anticipated and created explicitly. In addition, groupings set persistent links between objects, which impede single-object editing operations.

In all the aforementioned examples, users have to put back individual objects or groups of objects to one of their past locations. In other words, they have to perform *reciprocal drag-and-drop actions*. Such actions occur frequently, and their associated cognitive and motor cost can be high. We present the DND^{-1} history model, that keeps track of all past locations for both individual and multiple object selections. DND^{-1} is based on a direct selective undo model for drag-and-drop actions that works for all past selections of both single and multiple objects. We extend the Dwell-and-Spring widget, that was introduced in Appert et al. [2012], to let users quickly restore single objects and groups of objects to any past location.[1] DND^{-1}, together with this extended version of the Dwell-and-Spring widget, better support exploratory tasks in direct manipulation interfaces, as they let users revert arbitrary objects to a previous configuration, while preserving the result of drag-and-drop actions that happened later in the editing process. For instance, DND^{-1} lets users perform the edits described in Figure 1 in a lesser number of actions, as they can undo actions performed on a selection of objects without having to undo the direct manipulations that were performed later on in the workspace.

We first describe the DND^{-1} model and detail how we extended the Dwell-and-Spring widget to support reciprocal drag-and-drop. We illustrate DND^{-1} on both simple and advanced cases, and discuss implementation details. Finally, we report on two user studies which indicate that: (i) users are faced with situations in which they would like to have better support for reciprocal drag-and-drop, (ii) they can successfully use the Dwell-and-Spring widget in such situations, and (iii) they can take advantage of DND^{-1} to solve advanced layout problems with reciprocal drag-and-drop.

2. RECIPROCAL DRAG-AND-DROP

Situations that call for reciprocal drag-and-drop can be simple: for instance, putting a window back to its last location or reverting it to its previous size. They can also be

[1]The original Dwell-and-Spring technique was supporting only a basic linear undo model for single objects.

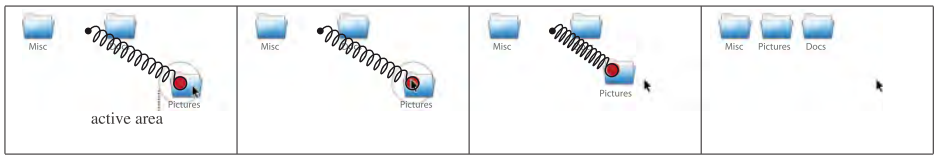


Fig. 2. The Dwell-and-Spring technique (DS). A red circular handle pops up close to the cursor when the user presses the mouse button and remains still for 500ms (i.e., dwells) over an icon. Releasing the mouse button while the cursor is over the spring handle will undo the last move of this icon.

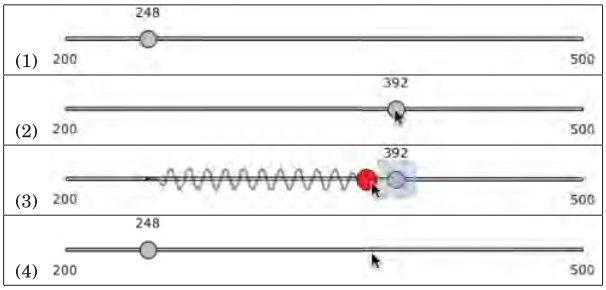


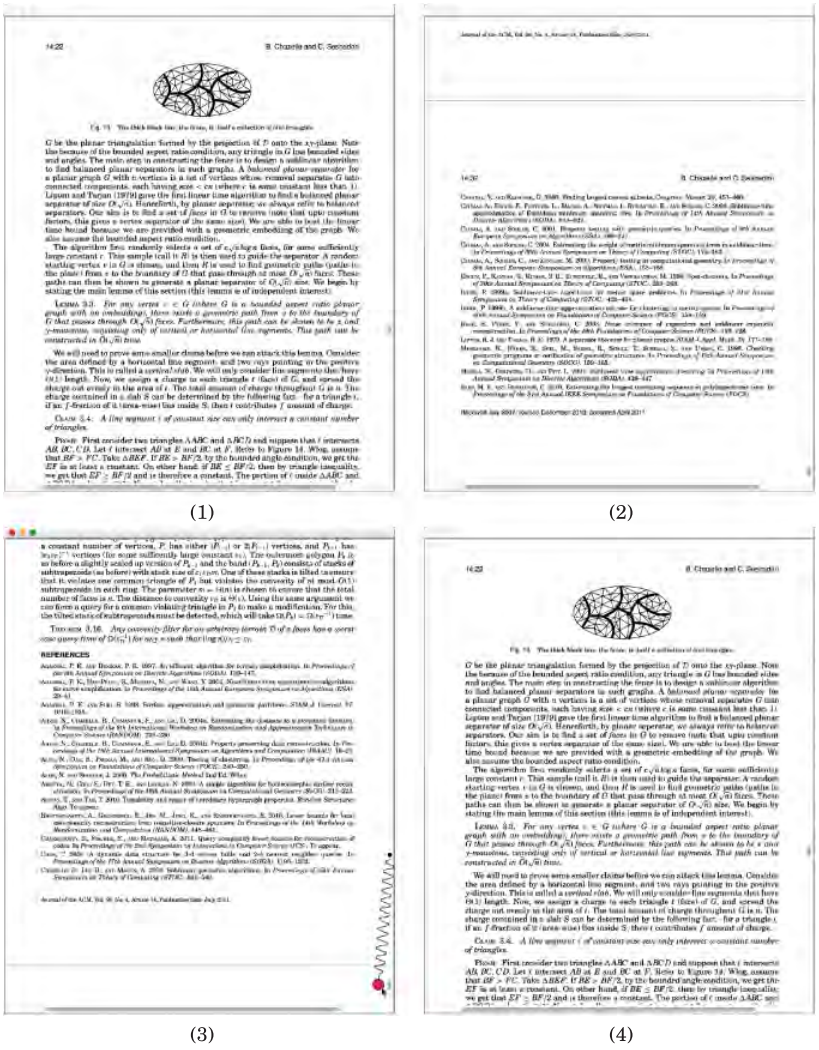
Fig. 3. Evaluating the impact of different values of a parameter controlled with a slider. (1) The current value of the slider is 248. (2) The user tests value 392. (3) Unsatisfied with it, he sets the value back to 248.

much more elaborate: for instance, putting back a group of shapes to an earlier position on the drawing canvas after having manipulated other shapes, and putting them back there while preserving the new relative position that was given to them after they were initially moved away. This section illustrates both simple and more advanced situations, and explains how we have redesigned the Dwell-and-Spring widget on top of the DnD^{-1} model to provide users with a flexible and powerful way of reverting various types of drag-and-drop actions (the companion video shows the technique in action).

2.1. Last Location of an Object

The basic Dwell-and-Spring technique, as described in Appert et al. [2012], readily applies to all simple cases of reciprocal drag-and-drop. Figure 2 illustrates it on a very simple case, where an icon gets restored to its last position. A red circular handle pops up close to the cursor when the user presses the mouse button and remains still for 500ms (i.e., dwells) over the icon. Bringing the cursor or finger onto this handle will make a spring appear, showing what the center of the icon will become if the user releases the mouse button or lifts his finger over the spring handle. If the user dwells without having initiated any movement, the spring shows the last move that was applied to the icon. If the user has already initiated a drag-and-drop, the spring proposes the reciprocal drag-and-drop for the current move. The user can either move over the spring handle and select it, activating the spring and thus bringing back the object to its previous location; or he can discard the widget by getting out of the active area.

This simple technique already applies to many cases of reciprocal drag-and-drop: manipulating icons on the desktop (Figure 2), navigating documents using a scrollbar (Figure 4) or with a swipe gesture on a touch-sensitive surface (Figure 5), moving and resizing windows (Figure 6), or any other action where the spring's actions are equivalent to what the user would manually do to revert to the original state, like moving a slider knob (Figure 3) or a manipulation handle (Figure 7). However, in its



(1) (2)

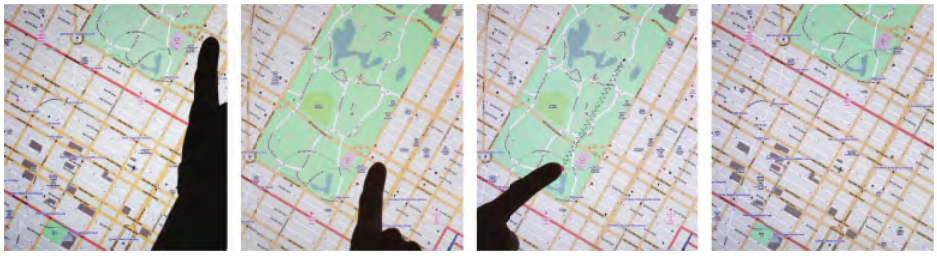
(3) (4)

Fig. 4. Reading a document. (1) The user finds a bibliographic reference while reading. (2) He drags the scrollbar knob to the end of the document to check that reference. (3-4) Once he has checked it, he invokes Dwell-and-Spring on the scrollbar knob to get back to the page he was reading.

original version [Appert et al. 2012], Dwell-and-Spring was only able to revert the current or the last drag-and-drop, as it was only keeping track of the previous location of each object, based on a per-object linear undo model.

2.2. All Past Locations of an Object

The first enhancement made to the Dwell-and-Spring technique is to provide users with extra spring handles that allow them to apply a series of reciprocal drag-and-drop actions quickly. As illustrated in Figure 7, the spring widget features additional handles that are horizontally aligned with the main spring handle (which was the only handle in the original design). Users can navigate through these handles to get



(1) (2) (3) (4)

Fig. 5. Panning a map on a tabletop. (1-2) The user swipes on the touch screen to set the viewport over the region of interest on the map, revealing more of Central Park in New York City. (3-4) She touches the screen and remains still to invoke Dwell-and-Spring on the map and go back to her view on Midtown.

(1) (2) (3) (4) (5) (6)

Fig. 6. Managing the desktop. (1) The user is making a transcription of a sketch into a vector-based document. (2-3) She resizes the window in order to copy and paste some elements that she had already drawn somewhere else in her document. (4-5) She restores the initial window size to get back to an ideal window layout for her transcription task. (6) She adjusts the location of the just pasted elements.

a preview of where the selected object(s) would go if they released the mouse button or lifted their finger on one of them. Releasing the mouse button while the cursor is over a spring handle will invoke the series of reciprocal drag-and-drop actions that are associated with this handle. As explained later, the series of past moves is managed by the DND^{-1} model, which differs from the linear undo stack that is implemented in most systems. With DND^{-1}, users can revert back to any past location, while keeping the length of the history as short as possible.

Applications that support undo typically store the history of actions as a tree whose nodes are the different states of the application. Performing an operation means creating a novel child state to the current node. Undoing an operation means getting back

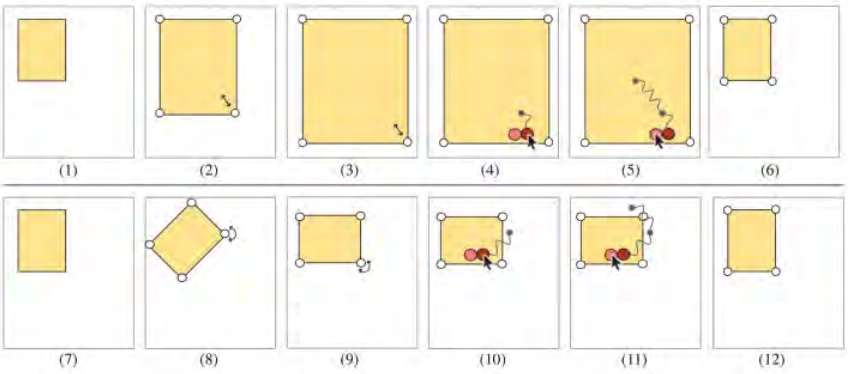


Fig. 7. DND^{-1} applied to shape manipulation handles. (1-2-3) The user resizes a rectangle twice. (4) She invokes Dwell-and-Spring on the resizing handle and enters the spring's main handle. This shows where the resizing handle was prior to the last resizing manipulation. (5) She moves the cursor to the next spring handle in the DND^{-1} history. This shows where the resizing handle was prior to the last two resizing manipulations. (6) She releases the mouse button on that second spring handle to revert back to the configuration in (1). Steps (7–12) illustrate a similar scenario on a rotation handle.

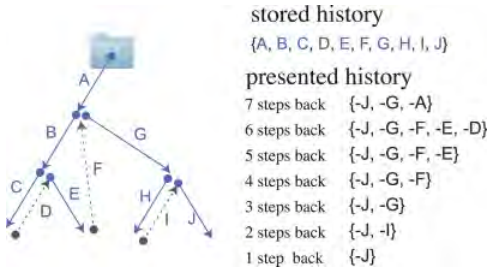


Fig. 8. DND^{-1} stores all repositioning actions applied to an object, including those performed via a reciprocal drag-and-drop (D, F, and I, shown as dashed black lines). It presents the shortest path to all past locations.

to the parent node. Some systems make several branches active at the same time: in collaborative work settings (e.g., Edwards and Mynatt [1997]) or for comparing variations of an image design that have minor differences [Terry et al. 2004]. However, the linear undo model only supports one *single active path*. All nodes outside this path are inaccessible via undo. For instance, in Figure 8, the user moves the icon three times successively (displacements A, B, then C), reverts C, and then moves the icon again by E. At this point, she can no longer recover the position the icon had after displacement C, since this one no longer belongs to the active path ({A,B,E}). A few applications, such as Emacs [Gosling 1982; Yang 1992], make users able to recover any state. However, this might require chaining a long series of interactions to reach a given state, as the history stack is presented to them as a *full sequential path* in the history tree.

Figure 8 illustrates DND^{-1}, the local undo model we propose to navigate in the history tree of displacements performed on a graphical object. This model lies in-between the *single active path* and the *full sequential path* models described previously. It stores the full history of repositioning actions, but provides users with shortcuts to quickly access nodes in the tree, so as to make them able to recover any past location, in the spirit of the Selective Undo model [Berlage 1994]. Each object remembers the sequence of moves that were applied to it, including reciprocal drag-and-drop actions. All past

locations are accessible. Also, when a user invokes a reciprocal drag-and-drop action to restore a past position P of an object O, $\text{D}\text{N}\text{D}^{-1}$ adds the straight move between O's current location and P to the end of the history, in the spirit of the *inverse model* for selective undo (used in, e.g., [Berlage 1994; Myers 1998; Yoon et al. 2013]), rather than inserting all reciprocal moves after the corresponding moves in the history, as the *script model* (used in, e.g., [Kurlander and Feiner 1988; Myers et al. 2015]) would have done.

For example, in Figure 8, the user moves an icon using three standard drag-and-drop actions (moves A, B, and C). He then moves the icon back to the location where it was before reaching C, using a reciprocal drag-and-drop (move D). At this point, there is no difference between $\text{D}\text{N}\text{D}^{-1}$ and the *script model*: both append D (i.e., −C) right after C at the end of the history: {A, B, C, D}. However, when the user moves the object back to the location it had before move B (following move E), $\text{D}\text{N}\text{D}^{-1}$ handles this reciprocal drag-and-drop as any regular drag-and-drop by appending F to the history ({A, B, C, D, E, F}), while the *script model* would have turned the history into {A, B, -B, C, -C, E, -E}. This entails that a drag-and-drop that has been undone with $\text{D}\text{N}\text{D}^{-1}$ can be easily redone. However, keeping a trace of all past moves also means that the number of steps to revert to a past location can be very long. In order to present an object's history of past locations in a compact way, we have implemented a navigation algorithm that computes the shortest path in the history to reach any of these past locations. This is basically achieved by removing cycles, that is, series of moves that bring the object back to a location already present on the path. For instance, navigating two steps back with $\text{D}\text{N}\text{D}^{-1}$ after move J in Figure 8(b) entails following path {−J, −I}; but navigating three steps back entails following path {−J, −G}, as {−J, −I, −H} would have led to the same location than {−J}, which is already proposed for a one-step-back navigation. This simplification decreases the number of steps that should be presented to the user, while ensuring that he can reach any past locations. For the scenario in Figure 8, thanks to this simplification, our extended version of Dwell-and-Spring presents seven spring handles, while it would have presented ten (i.e., the length of the stored history) otherwise.

2.3. Groups of Objects

The second enhancement made to Dwell-and-Spring is to provide users with another type of extra spring handles that allow them to apply reciprocal drag-and-drop actions to groups of objects that were moved simultaneously in the context of a multiple selection. Figure 9 illustrates what Dwell-and-Spring looks like after the user has played the scenario of Figure 10. These additional square handles act on the groups that contain object O, on which Dwell-and-Spring has been invoked. Handles are organized into several rows, one per group. The primary handle of a row is aligned with the main spring handle that initially popped out. When the cursor enters the handle associated with a group G, additional handles appear on its left. There are as many handles as the number of locations this group has visited. To help users anticipate what will happen if they activate a given handle, Dwell-and-Spring gives some feedforward when the mouse cursor or the finger enters that handle: objects in the group are highlighted, and the sequence of moves that will be reverted is shown as a series of springs that ends on what will become the center of object O after completion of the sequence of reciprocal drag-and-drops.

To keep track of all multiple selections an object has belonged to, the $\text{D}\text{N}\text{D}^{-1}$ model stores the whole history of moves into a hashtable (*History*) whose keys are groups of objects (which can be singletons) that index lists of timestamped movements. Figure 10 illustrates this on a simple example. However, there can be some ambiguity when deciding what a group is, as the same objects can be involved in different multiple

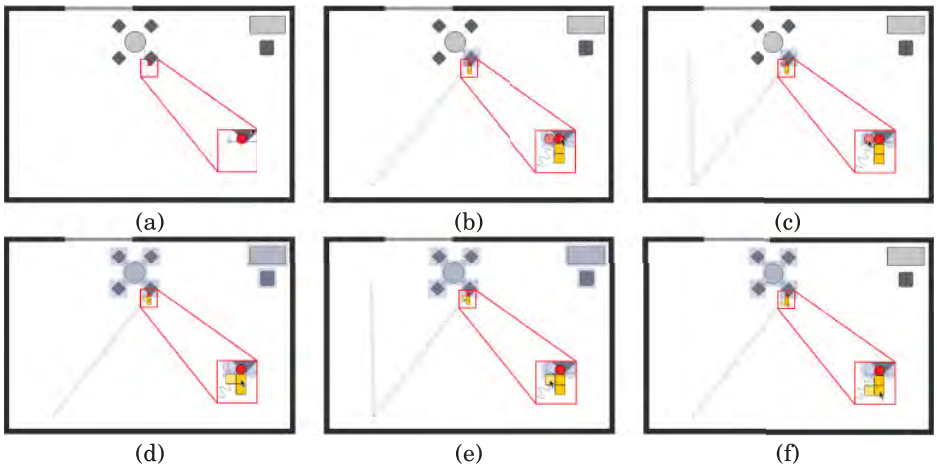


Fig. 9. Extended version of Dwell-and-Spring (the zoomed-in inset is added on top of UI screenshots for illustration purposes only). (a) The user presses the mouse button and remains still over object O; the main circular handle appears after a short delay. (b) The user enters this handle; additional handles appear as a result. (c) Navigating the row of circular handles lets her undo sequences of moves for O as an individual object. (d) Square handles allow her to restore any past location for groups in which O was – or is – involved. The first row of square handles will act on group $G_{\text{meeting+desk}}$, that contains the seven objects in the scene. (e) Navigating along this row lets the user undo sequences of moves for group $G_{\text{meeting+desk}}$. (f) The second row of square handles will act on group G_{meeting}, that contains the four chairs and the circular table.

selections [Figure 10(a-d)]. In order to maintain a coherent history, we have designed a strategy for handling groups that allows users to recover any past state without breaking any previous group.

When a group G has been moved by d, we first add G and all the singletons for objects in G that are not already in *History*. We associate an empty history with each of them (lines 1–8). We then review the history of all groups, as detailed in Algorithm 1. Each group G_i is split into two parts: $G_\cap$, that contains the objects that belong to both G and G_i, and $G_\setminus$, that contains the objects that only belong to G_i (lines 10-11). Each of these groups, if not empty, are updated in *History* (after creation if needed). The history of $G_\cap$, which is empty if just created, is merged with the one of G_i and is then enriched with the last move d (line 15). The history of $G_\cap$, which will be empty if it just got created, is merged with the one of G_i.

Figure 10 illustrates this on a concrete example. Starting from an empty *History*, Group_A, that consists of seven objects, is moved by d_1 as illustrated in step (a). Our algorithm creates seven singletons and Group_A in *History*. Then, during the process of revising existing groups, the individual histories are populated with d_1. In step (b), d_2 is added to all histories. In step (c), the user moves Group_B by d_3. This latter group is created (line 3 in Algorithm 1) and, when revising the existing groups, Group_A is split into Group_B (line 10) and Group_C (line 11). Group_B's empty history is merged with the one of Group_A, which is (d_1, d_2), and d_3 is added to the resulting history (line 15). Group_C is also created (line 18) and its empty history is merged with the one of Group_A (line 19).

The creation of singleton objects (line 4 in Algorithm 1) gives more flexibility to users by allowing them to revert moves on individual objects, as in the scenario of Figure 11, detailed later. However, we do not add all sets that belong to the power set of group G (the one that just moved). This is to keep the number of groups in *History* reasonable,

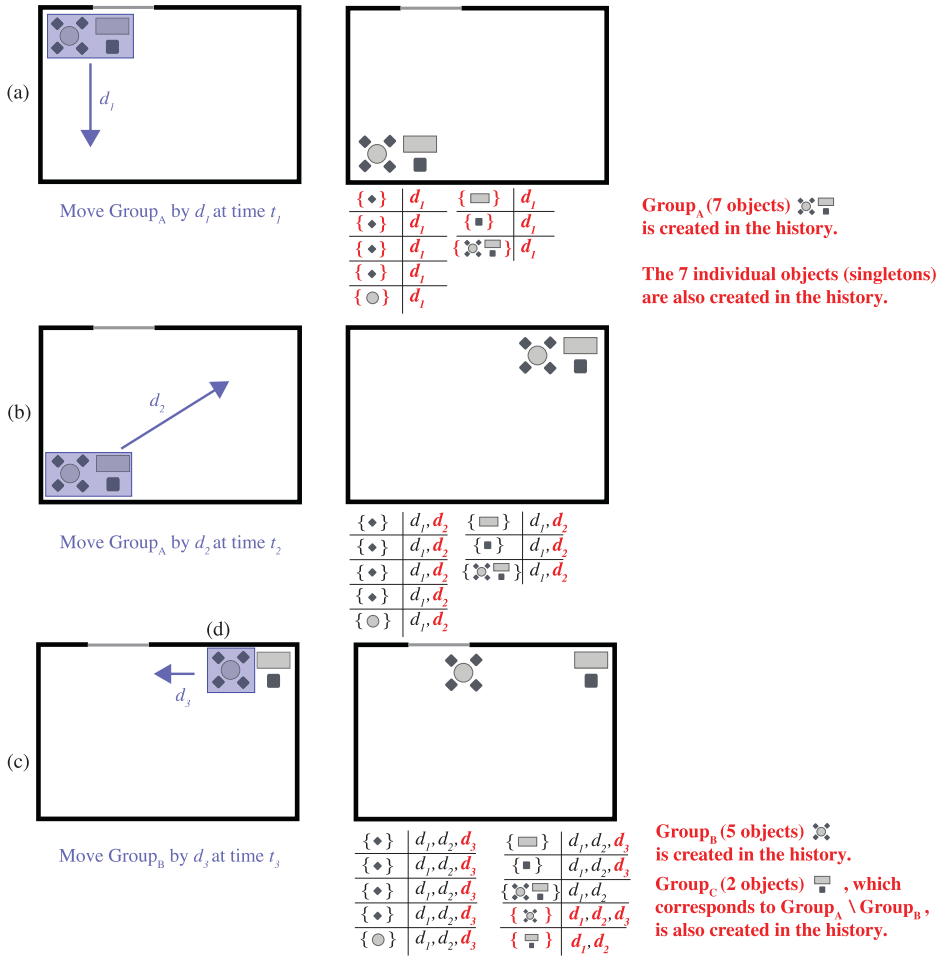


Fig. 10. A scenario involving multiple selections: arranging furniture on an office's floor plan. (a-b) Moving Group$_A$ by d_1 then d_2. Group$_A$ consists of all seven pieces of furniture: the circular table and its associated four chairs + the rectangular table and its associated chair. (c) Moving group Group$_B$ by d_3. Group$_B$ consists of the two pieces of furniture: the rectangular table and its associated chair.

while ensuring that any revert operation is possible. This simplicity versus flexibility tradeoff means that, if users want to revert a move for a subset g of objects that belong to group G, they have to revert that move on each of the individual objects. Finally, when users delete a graphical object, this object is not removed from *History*, but is simply tagged as passive, so as to keep a memory of it in case it gets restored by means of the application's functional undo model or by drag-and-drop across applications. Its passive state makes it ignored by our algorithm for revising groups.

Support for groups in DND^{-1} means that users can keep a trace of previous multiple object selections even if other selections happened afterwards. In particular, the current relative layout between objects within a group G is preserved in case they want to restore a past location of G, making transitions such as the one in Figure 1(i-j) very easy: the user can put back all desk elements to a past location while preserving the

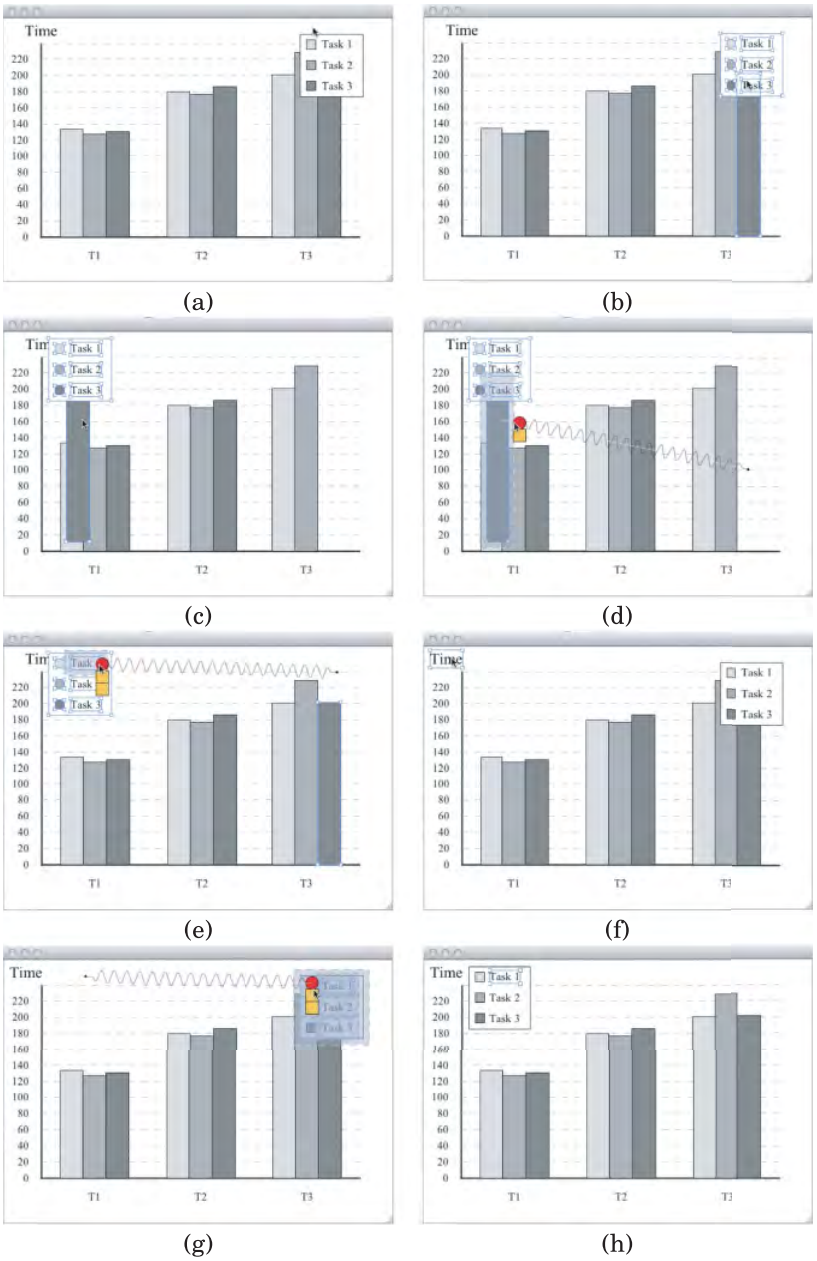


Fig. 11. Positioning the legend of a bar chart. (a-b) The user makes a selection of all elements of the legend. (c) He notices that he has accidentally included a bar in his selection and moved it along with the legend. (d) He uses Dwell-and-Spring to restore the bar to its original position. (e) The novel placement of the legend is not satisfying, and he puts the legend back where it was initially. (f) He moves the *y*-axis label, and (g-h) reverts the legend back to the left position.

ALGORITHM 1: Revising *History* after a group G has been moved by d at time t.

```
// Let h be the function that returns the local history of G stored in History
   as a list of timestamped past movements (d_i, t_i), merge(H1, H2) be a function
   that creates a chronologically ordered list of all timestamped movements of
   H1 and H2 (removing duplicates), and append(H, (d, t)) be a function that
   appends (d, t) at the end of H.

if G does not exist in History then
    h(G) = ∅
foreach object O in G do
    if {O} does not exist in History then
        h({O}) = ∅
end
foreach existing group G_i in History do
    G_∩ = G_i ∩ G
    G_\ = G_i \ G_∩
    if G_∩ ≠ ∅ then
        if G_∩ does not exist in History then
            h(G_∩) = ∅
        h(G_∩) = append(merge(h(G_∩), h(G_i)), (d, t))
    if G_\ ≠ ∅ then
        if G_\ does not exist in History then
            h(G_\) = ∅
        h(G_\) = merge(h(G_\), h(G_i))
end
h(G) = append(h(G), d)
```

rearrangement of the elements that was made afterwards. It also overcomes some limits of the *single active selection* model used in most applications. With the latter, the current selection gets cleared as soon as users click on a region or object that does not belong to the selection. DND^{-1} should save a lot of time and effort when trying to revert complex selections (small objects, objects scattered all over the workspace, partially occluded objects, etc.). For example, Figure 11 shows a scenario where a user wants to test alternative placements for the legend of a bar chart. The user accidentally selects a bar along with the legend, but notices it only after he has moved the legend to another location (a-c). As each object is also added individually in *History*, he can easily restore the bar's position (d), which has the effect of creating the group that contains only the graphical elements of the legend in *History*. As the novel placement of the legend overlaps with the *y*-axis label, he applies a reciprocal drag-and-drop to this group to put it back where it initially was (e). Later, he decides to slightly offset the label of the *y*-axis (f) and can easily test the alternative placement with the legend on the left again (g-h).

2.4. Implementing DND^{-1}

DND^{-1} is designed to be implemented at the system level, in an application-independent manner. We developed a Java prototype using SwingStates [Appert and Beaudouin-Lafon 2008] to demonstrate the approach, in which each client application implements the `ReciprocalDnDProtocol` interface and runs in a `JInternalFrame`. The `ReciprocalDnDManager` communicates with these applications exclusively through messages, while

the interaction techniques to navigate in DND^{-1} (Dwell-and-Spring or DnD-List—described later in Experiment 2) are hosted on a full-screen `GlassPane`.

The central object in the implementation is the `ReciprocalDnDManager`, that handles the entire *History* across client applications. Application developers can add support for DND^{-1} simply by registering their application with the `ReciprocalDnDManager` and implementing the simple protocol described later.

Client applications send a message each time an object is created or deleted. The `ReciprocalDnDManager` can then tag this object as either active or passive. They also send a message `moved(`G`,`d`)` each time a group of objects G is moved by a displacement vector d. The `ReciprocalDnDManager` then updates *History* according to the algorithm described earlier. All these client-to-server messages contain the sender application's id, and the ids of objects that are created, deleted or moved. The `ReciprocalDnDManager` organizes object ids using namespaces (one per client application), meaning that unicity is ensured across applications but that applications still have to guarantee the unicity of their objects' ids.

Techniques for interacting with DND^{-1} are connected to the `ReciprocalDnDManager` to expose *History* to users and let them select a group G and a past location p to restore. When users invoke such a reciprocal drag-and-drop, the `ReciprocalDnDManager` updates *History* by adding δ to the history of G, with δ being the vector between the current location of G and p. It then sends a `translate(`G`,`δ`)` message to the right client application with the ids of objects in G and the translation vector as arguments. The client application is then in charge of applying this displacement. To enable the implementation of feedforward mechanisms such as object highlighting, client applications must also be able to reply to two other messages: `id:pick(screenPoint)` and `rectangle:bounds(`G`)`. The first returns the id of the object that is picked at a given location, the second returns the bounding box of a group of objects. We have used this prototype to implement the experiments that we report on in the next sections.

3. EXPERIMENT 1: UNDERSTANDING USERS' DRAG-AND-DROP HABITS AND EVALUATING THE DISCOVERABILITY OF DWELL-AND-SPRING

For completeness, we include an experiment that was reported in Appert et al. [2012], in which we introduced the original, simpler version of the Dwell-and-Spring widget. We conducted this first experiment to capture what users typically do in situations where they want to revert a drag-and-drop. We also wanted to evaluate whether the spring metaphor implemented in Dwell-and-Spring is a viable alternative or not. The experiment lasted around 45min and contained two parts: an interactive *in situ* questionnaire to gather data about how users currently revert drag-and-drop actions, followed by a formal experiment to evaluate how easy it is to discover and understand Dwell-and-Spring, and how often they would actually use it once discovered.

3.1. Participants and Apparatus

Twelve unpaid volunteers (ten male, two female), aged 24–36 year-old (average 29.1, median 28.5), all daily users of personal computers, participated in this experiment: seven of them used Mac OS X, four Microsoft Windows, and one an X-Window system.

Each session started with a short paper questionnaire asking participants about their familiarity with, and use of, undo operations. Nine participants said that they use the undo operation very often, two often, and one sometimes. All but one participants reported using keyboard shortcuts often (e.g., `Ctrl/Cmd-Z`). Only one said that she mainly uses a toolbar button, with five participants sometimes using such a button. One participant also mentioned using an elaborate menu to navigate in the command history of an image editor (namely Adobe Photoshop).

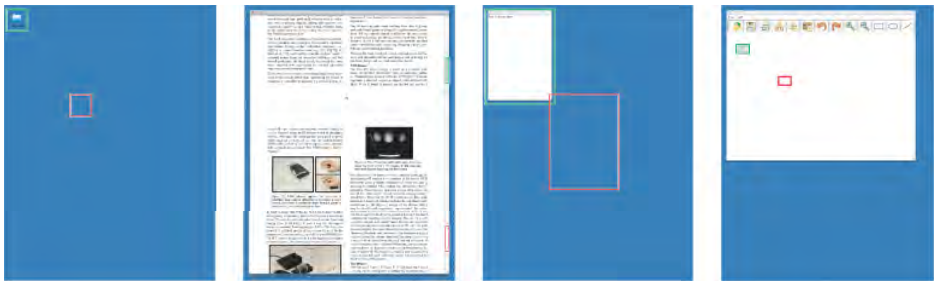


Fig. 12. Scenarios used for collecting users' habits in different situations of reciprocal drag-and-drop. The current location of the object (icon, scrollbar knob, window, or vector shape) is highlighted in green and the past position to restore is highlighted in red.

All sessions were conducted on a workstation with a 30" LCD monitor (2560 × 1600, 100 dpi, 1 pixel is about 0.256mm in width) running Mac OS X. The mouse was a standard optical mouse with 400dpi resolution and default system acceleration.

3.2. Capturing Users' Habits

To gather data about how users revert drag-and-drop actions more specifically, we used an interactive questionnaire where participants actually played several scenarios leading to cognitive states where they want to either cancel an on-going interaction or undo that interaction right after they have completed it. To simulate this cognitive state, participants were instructed to move a graphical object to a target location highlighted on screen. We considered two cases.

—DRAGGING case: an instruction pops up in the middle of the press-drag-release interaction (i.e., the user has not yet released the mouse button) asking the participant to stop and to put the object back where she grabbed it (*Cancel*).
—DROPPED case: an instruction pops up as soon as the participant has dropped the object at the target location (i.e., the user has just released the mouse button) asking her to restore the object to its previous location (*Undo*).

In both cases, we considered four scenarios involving different graphical objects (Figure 12): a desktop icon (S_{icon}), a scrollbar knob ($S_{\text{scrollbar}}$), a window (S_{window}), and a geometrical shape in a vector graphics editor (S_{editor}). As mentioned before, the answer can be highly dependent on the context of use as there is no unified way of doing such a cancel/undo operation across systems. This sample of scenarios was aimed at collecting answers representative of the different contexts of use. We also believe that asking participants to interactively *show us* what they would do in each scenario, as opposed to simply *telling us* in response to a verbal description, captures answers that have higher ecological validity.

We asked questions for the four scenarios, first in the DRAGGING case, and then in the DROPPED case. We decided to use a fixed presentation order for the two cases because the DRAGGING case can always be solved the same way the corresponding DROPPED case is, that is, the user can always decide to commit his current drag by releasing the mouse button and then undo it. Within both cases, the scenario presentation order was counterbalanced using a Latin Square.

Our interactive questionnaire presented the user with a desktop environment where all existing techniques were made available, in all contexts we tested. This means that the drag-and-drop of any graphical object could be cancelled by right clicking,

DRAGGING case (would like to do, usually do)

Strategies \ Scenarios	S_{icon}		S_{window}		S_{editor}		$S_{scrollbar}$	
Manual	9	9	10	12	8	4	12	11
Escape Key	2	2	1	0	1	1	0	0
Menubar drop	0	0	0	0	0	0	0	1
Drop-then-Undo	1	1	1	0	3	7	0	0

DROPPED case (would like to do, usually do)

Strategies \ Scenarios	S_{icon}		S_{window}		S_{editor}		$S_{scrollbar}$	
Manual	9	11	8	11	3	1	10	11
Cmd-Z	3	1	4	1	8	10	2	1
Toolbar button	-	-	-	-	1	1	-	-
Menu item	0	0	0	0	0	0	0	0

Fig. 13. Strategies reported in the interactive questionnaire for each scenario in both the DRAGGING case (top) and the DROPPED case (bottom). Each cell corresponds to a strategy and contains two numbers: first the number of participants who effectively used this strategy in the questionnaire, second the number of participants who usually employ this strategy.

dropping in the menu bar, or pressing the `Escape` key. Once committed (i.e., mouse button released), the user could undo the last action by either using the `Cmd-Z` keyboard shortcut or by selecting an Undo item in the menu bar (always displayed at the top edge of the screen). In scenario (S_{editor}), there was an additional possibility: an undo button in a toolbar, as most applications of this kind actually feature one. Our environment offered a kind of "ideal setting" by making all possible techniques available, whatever the scenario. Our goal was to let participants show us both what they *would like to do* (Q_1), and what they *usually do with their current system* (Q_2) in each situation.

Figure 13 summarizes the answers from our participants. It first shows a clear difference between the graphical editor scenario (S_{editor}), in which participants manipulate content (functional level), and the other scenarios, in which they modify the view configuration (view level). With S_{editor}, many more participants employed another technique than manually reverting. Ten participants reported using the `Cmd-Z` keyboard shortcut once they have DROPPED the object. Seven participants usually choose to drop and then undo when they are still DRAGGING. Very few participants used the `Escape` key and no participant used the right-click technique to abort and cancel the current action. In the other three scenarios, participants mainly restored the object to its original position manually, that is, by performing the same action in the opposite direction. This is even more pronounced in the DRAGGING case, where participants almost never used another technique.

There was almost no difference between answers to what participants *would like* to do and answers to what they *usually* do. However, we did collect a few surprising answers. For instance, one participant said that she usually used `Cmd-Z` in the DROPPED case under all the presented scenarios while her system only supports undo for the S_{editor} scenario. This indicates that some users might expect their system to be consistent over these four scenarios. Three participants told us that they usually use `Cmd-Z` to move back a desktop icon (S_{icon}) to its original position, and four thought they were using it to restore the past position of a window (S_{window}). They might have been thinking they can do this because undo works when the user drops an icon to a new folder, that is, a command at the functional level. This reinforces our intuition that the distinction

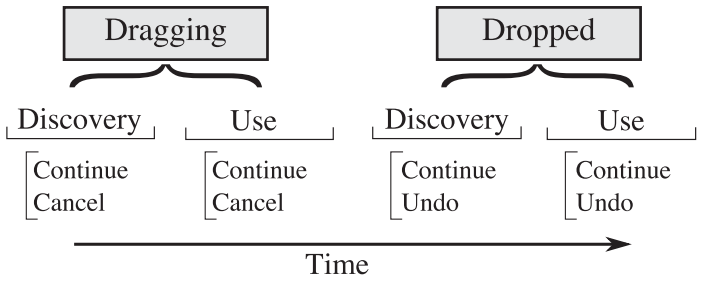


Fig. 14. Outline of the discovery-and-use experiment's design.

between the view level and the functional level is not always clear to users: in one case, the path of the file remains unchanged in the file system, while in the other case, the same interaction causes the file to be moved to a new folder.

Some participants made interesting comments during this interactive questionnaire. In particular, four participants told us that they would like to have an undo mechanism when scrolling, such as a button or an implicit bookmarking system. Expression of such a need for better revisitation mechanisms supports findings reported in previous studies (such as, e.g., [Ko et al. 2006]). Topaz [Myers 1998] includes scrolling operations in its history to allow users to selectively undo them. But users have to locate these specific navigation actions within the whole history of commands. As we will see later, Dwell-and-Spring is particularly well suited to canceling on-going, or undoing just performed, scrolling actions directly on the scrollbar, which could be additionally augmented with some colored marks, as in Alexander et al. [2009].

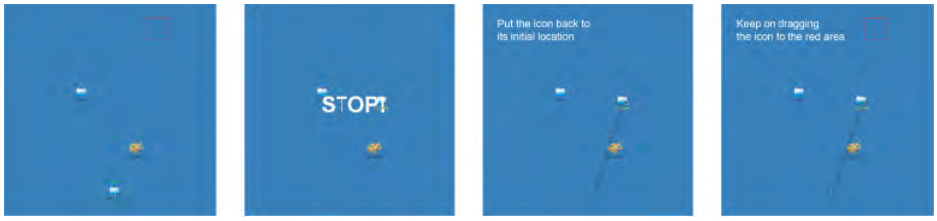
3.3. Discovery and Use: Experiment Design

After the interactive questionnaire, participants ran an experiment whose purpose was to study the following questions.

(1) Is Dwell-and-Spring easy to discover?
(2) Will people be willing to use Dwell-and-Spring once they have discovered it?

Figure 14 outlines our experimental design. As in the questionnaire, we considered both cases DRAGGING and DROPPED. Trials were blocked by case, with the DRAGGING case always presented first. As mentioned before, we chose this fixed presentation order because the DRAGGING case can always be considered as a DROPPED case. We also expect that, in a real context of use, there should be transfer from the situations modeled by the DRAGGING case to the situations modeled by the DROPPED case. Dwelling in the middle of a movement that the user finally wants to cancel seems rather natural: consider, for example, the scenario where the user takes a quick look at a given object in a scrollable view before coming back to the location where she was editing; or the scenario where the user temporarily moves a window to look at the graphical scene under it. We expect this case to lead to discovery of Dwell-and-Spring so that users will more easily understand they can adopt a similar approach in the DROPPED case.

To limit the length of the experiment, we only considered the desktop icon scenario (S_{icon}). The task consisted in moving the icon to a target location shown as a red rectangle (Figure 15). In the DRAGGING case, participants were told before starting that they would be interrupted in the middle of their move by a pop-up message that would give them further instructions about how to finish the trial. The instruction would be either to put the icon back to its original location (*Cancel* condition) or to finish the current operation, that is, move the icon to drop it at the intended target location



(a) (b) (c)-*Cancel* (c)-*Continue*

Fig. 15. Discovery task used in Experiment 1. (a) Participants are instructed to move an icon, highlighted in green, toward a target location, which is highlighted in red. (b) In the middle of their drag-and-drop, a message pops up to ask them to stop their movement. (c)-*Cancel*–(c)-*Continue* Participants are instructed to either put back the icon to its initial location or keep on dragging the icon to the target location.

(*Continue* condition). Once they had followed the new instructions, participants had to press the Space bar to end the trial.

In the DROPPED case, participants also had to drag-and-drop a desktop icon to a target location. As soon as they had dropped the icon, they got a message asking them to either put it back where it was before they moved it, and then press the Space bar (*Undo* condition), or to just press the Space bar immediately (*Continue* condition). In both cases, when they were told to restore the icon to its original location (*Cancel* and *Undo* conditions), the instruction explicitly mentioned that "various techniques may be available" to help them.

In both cases, trials were organized into four blocks of 24. For instance, when DRAGGING, a block contained 12 trials in the *Cancel* condition and 12 trials in the *Continue* condition, presented randomly. The 12 trials in the same condition always involved an icon of 48×48pixels, located 800pixels from the target location. Only the direction of movement varied across trials to take into account the fact that the spring's orientation depends on the movement direction. To vary movement direction, we laid out icons and target locations in a circular way.

There were two phases for each case: *discovery* and *use*. For the first two blocks, participants did not receive any indication about available techniques. They were simply encouraged to explore the interface. After completion of these two *discovery* blocks, the experimenter demonstrated each available technique before participants ran into the two other *use* blocks. These last two blocks were aimed at observing what strategy participants adopted once they had been exposed to all techniques, with clear instructions about how to use them.

Because we were interested in observing how people behave with the Dwell-and-Spring technique in a traditional desktop environment, but also in contexts where the hardware does not feature additional physical buttons or keys (e.g., a touchscreen as in Figure 5), the environment only proposed techniques that rely on "single-point input." The environment only proposed: the Dwell-and-Spring technique (*Dwell-and-Spring*), the technique that consists in dropping the icon in the top menu bar (*MenuBar*), an undo menu item (*EditMenu,*) and, of course, the manual technique that basically consists in dragging the icon back manually (*Manual*).

3.4. Discovering Techniques

We first analyze data we collected in the *discovery* phase of both cases DRAGGING and DROPPED.

3.4.1. DRAGGING Case. 4 out of 12 participants discovered how to use the *Dwell-and-Spring* technique. This is less than we expected since the spring popped up in 96% of the

trials and we thought that the spring offered powerful feedforward. The experimenter's observations help explain this low rate of discovery: several participants only used the spring as a visual guide, and not as a reactive graphical object. When the spring popped up after a dwell, some participants brought the cursor over the spring handle but did not activate the spring by releasing the mouse button as soon as the cursor was over the handle. Instead, they brought the spring handle towards the spring's origin (thus compressing the spring) and only then released the mouse button. The same participants spontaneously said to the experimenter that the spring was useful because it showed the icon's original position. Quantitative evidence backs this interpretation: participants who did not discover the technique grabbed but dropped the spring in about 70% of all cases (under the *Cancel* condition; this happened in only 3% of the cases under the *Continue* condition).

The four participants who discovered the *Dwell-and-Spring* technique understood how to use it during the first block: at first try for two of them, at second and sixth try for the two others. Once discovered, they used the technique a lot: in 100%, 92%, 79%, and 70% of all cases (*Cancel* condition). They also made a few errors in the first block where they activated the spring under the *Continue* condition, but no such accidental spring activation was observed in the second block.

This suggests that feedforward about spring activation should be stronger. A simple solution consists in making the spring more difficult to drop, to offer more opportunities to activate it (e.g., by enlarging the area where the spring is visible around the activation point or by making it more difficult to compress). Another approach would consist in removing the need for a release event to activate the spring (the spring would get activated as soon as the cursor enters the spring's handle). However, these design solutions would make the spring hard to discard in case the user does not want to activate it. An interesting tradeoff might be to come up with a way of discarding the spring that is a function of expertise: the spring could be made difficult to drop only the first few times the user explicitly interacts with it.

3.4.2. DROPPED *Case.* Seven participants discovered how to use *Dwell-and-Spring*. This may seem like a lot, given that contrary to case DRAGGING, the spring would not spontaneously pop up; participants had to explicitly press and dwell on the object to see the spring. But once a participant had found how to invoke the spring, they already knew how to use it as they had all learnt how to do so in the DRAGGING case. This observation tends to support our expectation of a learning effect between the cancel and undo conditions during the experiment.

As in the DRAGGING case, participants discovered the spring technique in the first block: two at first try, two at third try, two at fourth try, and one at eighth try. Then, as in the DRAGGING case, they used the *Dwell-and-Spring* technique a lot: 95%, 92%, 68%, 100%, 88%, 87%, and 86%.

3.5. Using Techniques

The aforementioned analysis reveals that users who discovered the *Dwell-and-Spring* technique made extensive use of it. In the following, we analyze data collected in the *use* phase (after *discovery*), to find out whether the other participants, who did not discover the technique by themselves, eventually adopted *Dwell-and-Spring* once exposed to it and to the other techniques (*MenuBar*, *EditMenu,* or *Manual*) by the experimenter.

3.5.1. Frequency of Use and Qualitative Results. Figure 16 shows the frequency of use of *Dwell-and-Spring* in the last block[2] for conditions *Undo* (DROPPED case) and

[2]We analyze data in the last block only, as it is more likely that participants had made a "definitive" choice by then.

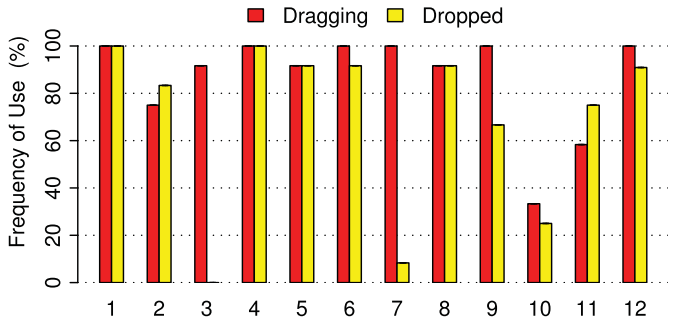


Fig. 16. Use of Dwell-and-Spring in the last block for both DRAGGING and DROPPED, per participant.

Cancel (DRAGGING case). The only other technique that was used significantly is *Manual*: *MenuBar* was used only twice and *EditMenu,* was used six times.

Except for P_3, P_7, and P_{10}, participants used *Dwell-and-Spring* very often, with P_1 and P_4 using it systematically. The three participants who used *Dwell-and-Spring* in less than 50% of the trials in the DROPPED case said that they were not willing to wait for the spring to pop up to precisely reposition the icon, as precision did not matter much. They also stated that they would have used *Dwell-and-Spring*, had precision been an issue, for example, had the task been to reposition a scrollbar knob. We discuss this speed-accuracy tradeoff in the next section. We can also observe that the frequency of use is a bit lower in the DROPPED case than in the DRAGGING case. This is probably due to the fact that doing a long press on an object to undo its last move is less natural than making a pause during a movement the user wants to cancel.

The aforementioned results show that most participants quickly adopted *Dwell-and-Spring*. Of course, the Hawthorne effect [Landsberger 1958] may have led to higher frequency of use than we would have observed in a real setting. However, the qualitative comments we collected at the end of the experiment were very positive and showed a real interest for the technique. Several participants spent a lot of time discussing design issues with the experimenter. Interestingly, more than half of the participants suggested that *Dwell-and-Spring* should enable users to trigger multiple undos in a single "spring step."

3.5.2. Outliers and Errors. For our analyses, we first filter out trials that end while the icon is more than 400pixels away from the ideal position it should have been put at. As the distance between start and target icon locations is initially 800pixels and the message pops up either at the end of the movement in the DROPPED case or in the middle of the movement in the DRAGGING case, this 400pixel criterion captures outlier trials where something unexpected occurred. In the DROPPED case, these are trials where participants pressed the space bar before putting the icon back to its original position in the *Undo* condition (i.e., instruction ignored, possibly because of mechanical routine). This happened in 2.09% of the trials in the DROPPED case. In the DRAGGING case, these are trials where participants either ignored the instruction that asked them to put the icon back where it was (*Cancel* condition) or activated the spring while they should have continued their current move (*Continue* condition). This happened in 2.78% of the trials in the DRAGGING case ($\sim$2% in the *Cancel* condition and $\sim$0.75% in the *Continue* condition). This shows that participants activated the spring while they should not have in less than 1% of the cases. All remaining trials, which ended with the icon being less than 400pixels away from the target location, are kept for further analyses.

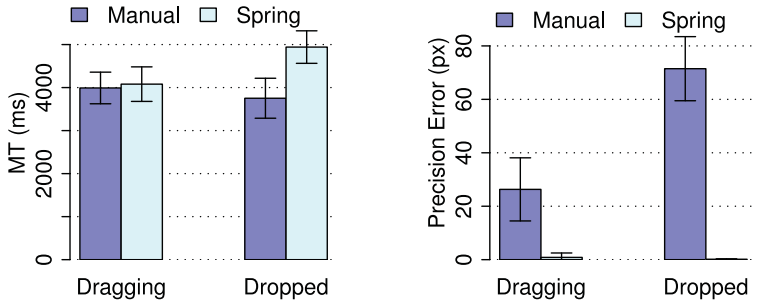


Fig. 17. Movement time (left) and precision error in pixels (right) for *Manual* and *Dwell-and-Spring*, in both cases DRAGGING and DROPPED (under *Cancel* and *Undo*). Error bars show the 95% confidence limits.

Regarding accidental interactions with the spring, we also recorded occurrences of participants grabbing the spring without activating it while they should not have used it (*Continue* condition). This happened in 3.86% of the trials in the DRAGGING case and in only one trial in the DROPPED case. All these trials ended without any error, indicating that participants were able to drop the spring. In the DROPPED case, a cancel spring popped up in about 6.62% of the trials under the *Continue* condition (typically just before the end of the task), but participants never activated it. These observations tend to show that the Dwell-and-Spring technique minimizes accidental triggers and enables easy repair.

3.5.3. Movement Time and Precision. Figure 17 shows movement time and precision (distance between the icon's original location and its position at trial end time) for trials in the *Cancel* and *Undo* conditions, after having removed the outliers mentioned earlier. In the DRAGGING case, we observe very similar movement time for *Dwell-and-Spring* and *Manual*, and a better precision (distance close to zero) for *Dwell-and-Spring* than for *Manual*.[3] In the DROPPED case, *Manual* was about 1.2s faster than *Dwell-and-Spring*, but *Manual* was far less precise than *Dwell-and-Spring*. It is not surprising that *Dwell-and-Spring* offers a much better precision, since it automatically performs the ideal reciprocal manipulation, putting the object back to its exact original location. The average precision error with *Manual* was 71.5pixels (median 69pixels). Our experimental design allowed participants to choose which technique they wanted to use. This resulted in an unbalanced number of collected measures between the *Manual* and *Dwell-and-Spring* conditions, thus violating the assumptions made when running a statistical test comparing conditions. However, Figure 17 suggests a tradeoff between movement time and precision when comparing *Dwell-and-Spring* and *Manual* in the DROPPED case. The precision error is lower in the DRAGGING case than in the DROPPED case as the spring was always popping up at the moment participants had to pause in the middle of their drag movement. In such cases, the spring visually helped participants revert the icon back to its exact previous location.

3.6. Summary

Collecting users' habits in different contexts of use revealed that they always repair their direct manipulation errors manually, except when the direct manipulation acts at the functional level of the corresponding application. Observing users when they are in an environment where *Dwell-and-Spring* is available revealed that one third of users spontaneously tried to make use of it, and that demonstrating the technique even a

[3]The small imprecision observed in Figure 17-right is due to accidental clicks before pressing the space bar.

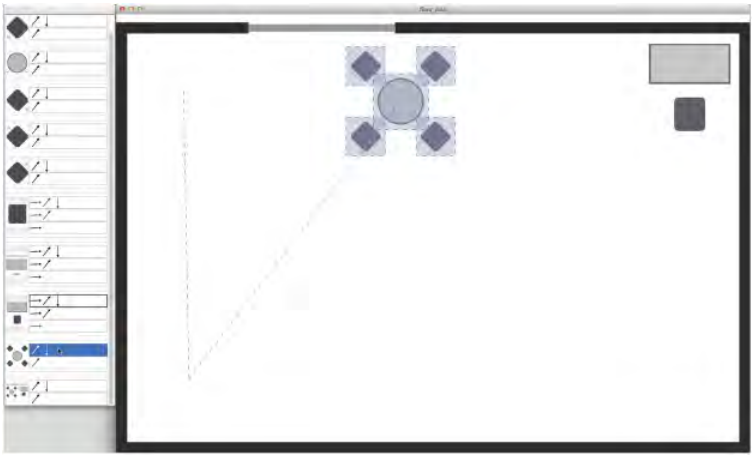


Fig. 18. The DnD-List (*List*) technique (after playing the scenario in Figure 10).

single time is sufficient for users to understand and adopt it. Our quantitative analysis highlighted the speed-accuracy tradeoff that users may face with such a technique. While it may be a bit slower in some cases, *Dwell-and-Spring* allows users to accurately cancel or undo a direct manipulation, which can be a significant advantage for precise positioning.

4. EXPERIMENT 2: USABILITY OF DND^{-1} COMBINED WITH DWELL-AND-SPRING

We conducted a second experiment to test if users could understand and use the DND^{-1} model effectively when they have to restore the position of either a single object or a group of objects. We also wanted to assess the relevance of the Dwell-and-Spring technique (abbreviated *Spring*) in comparison with a baseline technique, which exposes the full history (abbreviated *List*).

4.1. The DnD-List (*List*) Technique

As the DND^{-1} model is novel, we wanted to gather observations about its usability independently from its combination with Dwell-and-Spring. We thus designed a baseline technique that exposes the full history of DND^{-1} in a standard list presentation, close to the one found in, for example, Adode Photoshop. This technique, DnD-List (*List*), consists of a separate window, that remains always visible on top of other windows. As illustrated in Figure 18, this window shows a scrollable list. Each row displays the history of moves for a given group of objects G. An image of G is on the left, and the list of paths that lead to all past locations of G is on the right. A path is displayed as a series of arrows whose orientation matches that of the actual movements that brought G where it currently is. The user can click on any of these paths to actually execute the reciprocal drag-and-drop that reverts this series of movements. As DnD-List is implemented according to the DND^{-1} model, a reciprocal drag-and-drop is appended to the history, following the inverse model of selective undo (as Dwell-and-Spring does—see Section 2.2).

As illustrated in Figure 19, *List* makes a screen capture each time users press the mouse button, and creates a thumbnail picture of a group G when users select and move it for the first time. *List* crops the minimal square that fully contains the group and adds a white translucent mask on top of this image to emphasize the objects that

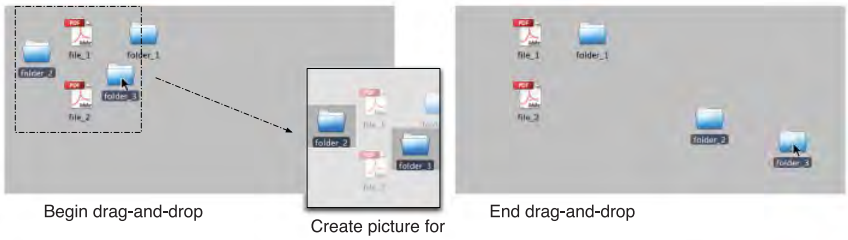


Fig. 19. Thumbnail creation process for DnD-List. When users initiate a drag-and-drop on a multiple selection (here, {folder_2, folder_3}) for the first time, DnD-List takes a screen shot, crops it and applies a mask to emphasize the objects that actually belong to the group.

actually belong to the group. Objects that fall within the bounding box of the multiple selection but that are not part of the group are still visible, but faded out.

Groups are sorted by the number of objects they contain (largest groups at the bottom of the list). Groups that feature the same number of objects are sorted according to the timestamp of their last move (a group is below another group if it has been moved more recently). Each time a move occurs, resulting from either a manual drag-and-drop or a reciprocal drag-and-drop, the path item that has just been inserted is highlighted and the window auto-scrolls to make it fully visible inside the list's viewport. Also, when the mouse cursor hovers an item, the bounding boxes of objects that belong to the associated group appear, along with a polyline showing what will become the center of this group if the user selects this item. Figure 18 illustrates this. G_{desk} has just been moved (Figure 10(e)) and the cursor hovers a path item to restore a past position of G_{meeting}. This hovering feedforward mechanism, which is intended to prevent errors, is hosted on a transparent layer on top of any application that implements the communication protocol described in the implementation section earlier.

4.2. Hypotheses

Our main hypothesis is that users can interact with DnD^{-1} using both techniques (factor TECH $= \{Spring, List\}$). We hypothesize that DnD^{-1} enables them to minimize the number of operations required to revert a sequence of moves.

Our secondary hypothesis is that the cost of using a technique depends on the structure of the *History* relative to the `reciprocalDnD(`G,P`)` operation to do. This cost is a function of both local and global depths. Let us consider a group G at position P. The performance of *Spring*, which operates object wise, will be mainly impacted by local depth, that is, the number of positions that G visited until it was eventually put in P (LOCAL-DEPTH). The performance of *List*, on the contrary, will also and mainly be impacted by the number of manipulations that have been performed on other objects since G left position P. Indeed, this entails that the right item is deeper in the list and that retrieving it requires more scrolling. To operationalize this notion of cost, we consider two secondary factors: LOCAL-DEPTH and LIST-SCROLL, as detailed later.

The experiment was divided into two parts and took 70min on average. In each part, participants had to both perform movements of objects using drag-and-drop operations and apply undo operations in order to reach a specific graphical layout. In the first part, the difference between the target and the current layouts can be corrected with a reciprocal drag-and-drop on an individual object, whereas in the second part, solving the difference requires performing a reciprocal drag-and-drop on a group of objects. In both cases, if users make an optimal use of the DnD^{-1}, they are able to solve the difference with a single (optimal) reciprocal drag-and-drop.

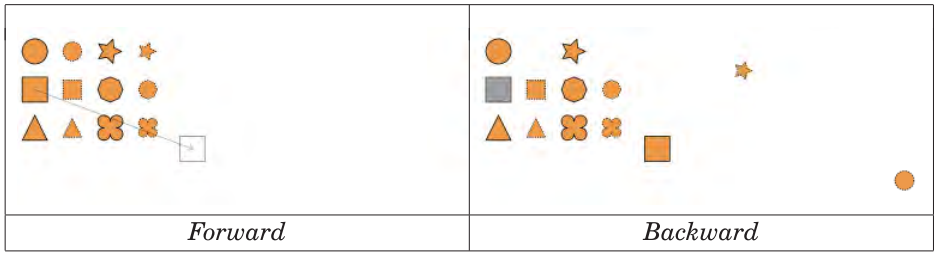


Fig. 20. Move and Undo instructions for a single object.

4.2.1. Participants. Twelve volunteers (five female), all right handed, aged 24–40 years old (average 30.0, median 28.5), daily computer users, participated in the experiment.

4.2.2. Apparatus. We used a Mac Pro workstation running Mac OS X, equipped with a high-end graphics card connected to a 30" LCD display (100 dpi, 2560 × 1600pixels) and an Apple Mighty Mouse set with the default transfer function.

4.3. Part 1: Multistep Reciprocal Drag-and-Drop for Individual Objects

4.3.1. Experiment Scene and Task. Starting from a graphical scene composed of 16 different shapes (Figure 20), participants are instructed to follow a scenario that consists of a series of *Forward* and *Backward* instructions. Participants have to press the space bar in order to get the specific instruction and perform it. In the case of a *Forward* instruction [Figure 20(a)], the next position P_{next}, where the shape S has to be moved is shown by displaying an outline of S in P_{next} with an arrow linking the center of S to P_{next}. In the case of a *Backward* instruction [Figure 20(b)], the past position P_{past} (that needs to be restored) of shape S is illustrated with a grayed out copy of S centered on P_{past}. To avoid any ambiguity when interpreting instructions, any two shapes either differ in their geometry (circle, star, square, hexagon, triangle, or clover) or in both their size (small or large) and outline stroke (dotted or solid). In the *List* condition, the size of the history list window is set to 250 × 300pixels, meaning that a maximum of four objects with up to two past locations fit in its viewport at the same time. To avoid introducing variability in the way participants navigated the list, its window could not be resized and it could only be browsed using the scroll bar.

The experiment program also controls the *History*'s structure, ensuring that it is exactly the same across participants. Drag-and-drop interactions are enabled only for *Forward* instructions. Conversely, the technique for restoring a past position (*Spring* or *List*) is enabled only for *Backward* instructions. During a *Forward* instruction, participants are allowed to move only the shape that has to be moved. Any other move makes the application beep and cancel the move without recording it in *History*. Similarly, during a *Backward* instruction, participants are allowed to restore a past position only for the shape that differs from the indicated layout. Any other reciprocal drag-and-drop causes a beep and is ignored. However, participants are not forced to perform the reciprocal drag-and-drop in a single (optimal) operation. They can choose to perform a series of reciprocal drag-and-drop operations as long as they perform them on the right shape. The program logs the number of operations and proposes the next instruction as soon as the shape is placed in the indicated position. Each time the program beeps, it counts a misinterpretation and reinitializes the timer used to record task completion time. We logged 3.8% such errors for *Spring* and 3.6% for *List*.

4.3.2. Design and Procedure. The experiment is divided into two blocks. One group of six participants see the *Spring* block first, followed by the *List* block, while the other

Participant	Block 1		Block 2	
P01	*Spring*	$S_{pratice}$, S_4, S_2, S_5	*List*	$S_{pratice}$, S_6, S_1, S_3
P07	*List*	$S_{pratice}$, S_4, S_2, S_5	*Spring*	$S_{pratice}$, S_6, S_1, S_3
P02	*Spring*	$S_{pratice}$, S_1, S_6, S_4	*List*	$S_{pratice}$, S_3, S_2, S_5
P08	*List*	$S_{pratice}$, S_1, S_6, S_4	*Spring*	$S_{pratice}$, S_3, S_2, S_5
P03	*Spring*	$S_{pratice}$, S_3, S_5, S_1	*List*	$S_{pratice}$, S_2, S_4, S_6
P09	*List*	$S_{pratice}$, S_3, S_5, S_1	*Spring*	$S_{pratice}$, S_2, S_4, S_6
P04	*Spring*	$S_{pratice}$, S_5, S_4, S_6	*List*	$S_{pratice}$, S_1, S_3, S_2
P10	*List*	$S_{pratice}$, S_5, S_4, S_6	*Spring*	$S_{pratice}$, S_1, S_3, S_2
P05	*Spring*	$S_{pratice}$, S_6, S_3, S_2	*List*	$S_{pratice}$, S_4, S_5, S_1
P11	*List*	$S_{pratice}$, S_6, S_3, S_2	*Spring*	$S_{pratice}$, S_4, S_5, S_1
P06	*Spring*	$S_{pratice}$, S_2, S_1, S_3	*List*	$S_{pratice}$, S_5, S_6, S_4
P12	*List*	$S_{pratice}$, S_2, S_1, S_3	*Spring*	$S_{pratice}$, S_5, S_6, S_4

Fig. 21. Experiment 2. Presentation order of the six scenarios and TECH conditions across participants.

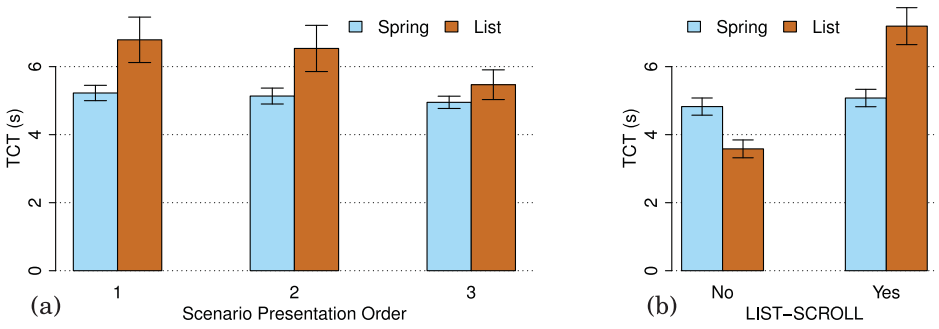
half start with *List* and then see *Spring*. At the beginning of each block, the operator introduces the technique that participants are about to use (3min). Participants then have to complete four scenarios, with the first one serving as a practice session (S_{practice}).

We collect measures (for analysis) on six scenarios ($S_1, \ldots, S_6$) per participant, that is, three scenarios per TECH block. We use the six same scenarios for all participants. We compute six possible scenario presentation orders with a Latin Square. Each order is assigned to two participants, one participant starting with *List* and another participant starting with *Spring* (Figure 21). Overall, each scenario is played six times with *List* and six times with *Spring*.

As explained earlier, a scenario consists of a series of *Forward* and *Backward* instructions. More precisely, a scenario contains 12 *Backward* instructions interleaved with some *Forward* instructions, 4.75 in average (min=1 and max=8), for a total of 57 *Forward* instructions. Each scenario is generated in a pseudorandom manner, ensuring a balanced number of measures per LOCAL-DEPTH × LIST-SCROLL and satisfying three criteria: (1) the 12 *Backward* instructions of a scenario are distributed among the same four shapes; (2) each shape is involved in three *Backward* operations with LOCAL-DEPTH $= \{1, 3, 5\}$; (3) half of the *Backward* instructions can be performed by clicking an item that is initially visible when using *List* (LIST-SCROLL=*NO*), while the other half requires scrolling the viewport (LIST-SCROLL=*YES*).

4.3.3. Results for Part 1. Among the 864 (12 instructions × 6 scenarios × 12 participants) *Backward* instructions measured, 850 were completed by invoking a single reciprocal drag-and-drop. This shows that participants were able to use $\text{D}N\text{D}^{-1}$ in an optimal way for 98.38% of the *Backward* instructions they had to perform. The distribution of nonoptimal trials plays against *List* (ten for *List* and four for *Spring*), and a pairwise Wilcoxon test ($n = 12$) confirms a significant effect of TECH on the number of nonoptimal trials ($p = 0.048$). However, there is no significant effect of the structure of *History* on the number of such trials: neither LOCAL-DEPTH nor LIST-SCROLL has a significant effect.

For trial completion time (*TCT*) analyses, we only keep trials completed in an optimal way. Among the 850 trials that were completed with a single reciprocal drag-and-drop, participants scrolled the viewport even though it was not required in 11.11% of the cases in the *List* condition. In these trials, participants failed to recognize the right shape in the list's thumbnails even though they were visible without scrolling (LIST-SCROLL=*NO*). This is not surprising, as searching through a list of graphical objects to find a target is a difficult task, that is costly in terms of both visual and cognitive processing [Salvucci 2001]. We filter out these trials, as they may be an artifact related to our experimental task. In a real context of use, objects may be either easier or harder to recognize in these thumbnails, depending on the type of graphics in the scene.



Effect for *TCT*	n, d	$F_{n,d}$	p	η^2_G
TECH	1,11	4.70	0.0530	0.04
LOCAL-DEPTH	2,22	8.89	0.0015	0.12
LIST-SCROLL	1,11	102	< 0.0001	0.45
TECH × LOCAL-DEPTH	2,22	1.55	0.2346	0.02
TECH × LIST-SCROLL	1,11	105	< 0.0001	0.38
LOCAL-DEPTH × LIST-SCROLL	2,22	1.74	0.1949	0.01
TECH × LOCAL-DEPTH × LIST-SCROLL	2,22	2.73	0.0873	0.02

(c) TCT ∼ TECH × LOCAL-DEPTH × LIST-SCROLL × Rand(PARTICIPANT)

Fig. 22. (a) *TCT* by scenario presentation order for each TECH. (b) *TCT* by TECH × LIST-SCROLL for the third scenario. (c) ANOVA results.

We look at the evolution of performance over the three scenarios per technique in order to check for a potential learning effect. As illustrated in Figure 22(a), we observe such an effect only for *List*, not for *Spring*. An ANOVA shows a significant interaction between TECH and the scenario presentation order on *TCT* ($F_{2,22} = 3.76$, $p = 0.0392$, $\eta^2_G = 0.06$). A *post hoc* (Holm corrected) *t*-test reveals that *TCT* significantly differs between each pair of conditions that vary in their scenario presentation order for *List*, while this is not the case for *Spring*. A *t*-test also shows that *Spring* is significantly faster than *List* in the first ($p = 0.0133$) and second ($p = 0.0188$) scenarios, but not in the third scenario ($p = 0.2136$).

So as not to disadvantage *List* by ignoring the fact that users' performance will benefit from learning, we analyze effects on *TCT* only for trials collected in the third scenario. Figure 22(c) reports the results of an ANOVA for model *TCT* ∼ TECH × LOCAL-DEPTH × LIST-SCROLL × Rand(PARTICIPANT). It shows that, in our experiment, the structure of *History* impacts the performance of both *Spring* and *List*.

More specifically, both LOCAL-DEPTH and LIST-SCROLL have an impact on *TCT* for both techniques. First, the local depth of *History* for an object (LOCAL-DEPTH), which sets either the number of spring handles to traverse with *Spring* or the number of items per object thumbnail with *List*, has an impact on the performance of both *Spring* and *List*. Second, LIST-SCROLL, which is related to the global depth of *History*, has a large effect on *TCT*. This mainly comes from the fact that the performance of *List* strongly degrades when users need to scroll the list of items, as illustrated in Figure 22(b). Indeed, the effect of TECH is marginal, whereas interaction effect TECH × LIST-SCROLL is very large. A *post hoc t*-test (with Bonferroni correction) actually shows that (i) *List* is significantly faster than *Spring* when LIST-SCROLL=*NO* ($p = 0.0003$) and that (ii) *Spring* is significantly faster than *List* when LIST-SCROLL=*YES* ($p = 0.0001$).

While our analyses consider trials that better reflect the behavior of expert users, results are similar when analyzing the entire set of measured trials. The only

difference lies in the performance of *List*, which is slightly worse than that of *Spring*. For instance, interaction effect TECH × LIST-SCROLL remains very strong, but there is a significant difference between both techniques only when LIST-SCROLL=*YES*, whereas *List* is not significantly faster than *Spring* when LIST-SCROLL=*NO*.

In summary, $D_N D^{-1}$, combined with either *Spring* or *List*, lets users restore past locations of individual objects in an optimal way. However, the performance of *List* suffers from a high variability depending on the reciprocal drag-and-drop considered and on the history structure. On the opposite, by adopting a per-object navigation strategy, *Spring* limits this cost, yielding more constant performance figures across reciprocal drag-and-drop actions.

4.4. Part 2: Reciprocal Drag-and-Drop for Groups of Objects

4.4.1. Experiment Scene and Task. In the second part of the experiment, the application enables users to perform multiple rectangular selections through rubber-band interaction. It also allows them to add and remove individual objects using Shift+Click. The graphical scene is the same as in Part 1, but every *Forward* and *Backward* instruction involves several objects. Also, as opposed to Part 1, in which *Forward* and *Backward* instructions were interlaced, a scenario in Part 2 gives a series of *Forward* instructions and ends with a single *Backward* instruction. This final instruction indicates a target layout that can always be reached with a single (optimal) $D_N D^{-1}$ operation.

As in Part 1, the application ensures that *History* is the same across participants by constraining their interactions. For a *Forward* instruction, participants have to move all shapes involved at the same time, using the correct multiple selection. Otherwise, the application beeps and ignores the last move. For the final *Backward* instruction, the application enables any sequence of reciprocal drag-and-drop actions. It records them until the target layout is reached. Participants were also allowed to skip a trial on demand, if they were confused and unable to figure out how to reach the target layout. This could have happened if they had chained a large number of reciprocal drag-and-drop actions.

4.4.2. Design and Procedure. As in Part 1, each participant has to complete two blocks, one per TECH. When a block starts, the operator gives a 2min introduction about how to use the technique on a group of objects. Participants then complete a practice scenario, followed by six measured scenarios. We generate 12 scenarios (2 series of 6) in advance, that we present to all participants. To make sure each scenario is performed with each technique across participants, the presentation order between the two series of six scenarios is always the same, but the presentation order between the two TECH conditions varies (six participants start with *Spring*, the six others start with *List*). We also counterbalance the presentation order of the six scenarios within a series by using a 6 × 6 Latin square.

In Part 2, each scenario contains from 9 to 11 *Forward* instructions that involve three or four subgroups of six specific graphical shapes among the 16 shapes in the scene. A subgroup consists of 2, 3, 4, or 6 shapes and is involved in 1 to 5 *Forward* instructions. The direction and amplitude of the drag-and-drop is randomly generated while avoiding overlap between shapes. The final *Backward* instruction is generated by picking one of a subgroup's past locations (the chosen subgroup must have been moved at least four times). The past location is either 1, 2, or 3 step(s) backward (LOCAL-DEPTH $\in \{1, 2, 3\}$). Half of the scenarios require scrolling with *List* (LIST-SCROLL=*YES*), while the other half does not (LIST-SCROLL=*NO*). In order to ensure an equivalent difficulty among the two sets of six scenarios, we generated six pairs of scenarios. Paired scenarios feature the same number of subgroups, which have the same size, and a similar sequence of

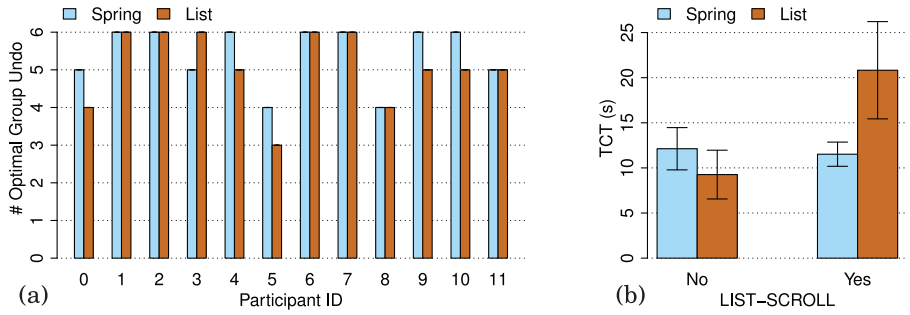


Fig. 23. (a) Total number of trials completed with a single optimal DND^{-1} among the six trials each participant performed with each TECH. (b) *TCT* by TECH × LIST-SCROLL.

instructions. They only differ in the shape of objects they contain, and in the amplitude and direction of each drag-and-drop.

4.4.3. Results for Part 2. As explained earlier, each final *Backward* instruction can be performed in a single optimal reciprocal drag-and-drop. The percentage of trials that were completed in such an optimal way was high in our experiment, with no significant difference between *Spring* (90.3%) and *List* (84.7%). Figure 23(a) reports the number of trials that were completed in such an optimal way, per participant. Eight of the twelve participants made either 0 or 1 error. Moreover, among the 18 trials that were not completed with an optimal use of DND^{-1}, 9 required only 2 or 3 reciprocal drag-and-drop operations, and only 3 were considered as too difficult and skipped at the participant's request. These results show that, even if the notion of group necessarily introduces some difficulty in the task, participants succeed in using the DND^{-1} model to apply reciprocal drag-and-drop on groups of objects.

For our analysis of *TCT*, we first remove the 18 nonoptimal trials mentioned previously. As we did in Part 1, we also remove 11.45% of trials in the *List* condition, which correspond to cases where participants scrolled the viewport even though this was not necessary (LIST-SCROLL = *NO*). However, as opposed to our earlier observations about reciprocal drag-and-drop on single objects, there was no significant learning effect on *TCT*, neither for *Spring* nor *List*. Figure 23(b) reports *TCT* by TECH for both LIST-SCROLL conditions and shows the same TECH × LIST-SCROLL interaction effect we already observed in Part 1. A *t*-test shows that *Spring* is significantly faster than *List* when LIST-SCROLL = *YES* ($p = 0.006$), while this difference is not significant when LIST-SCROLL = *NO* ($p = 1.0$). Finally, LOCAL-DEPTH did not have a significant effect on *TCT*.

4.5. Qualitative Questionnaire

At the end of the experiment, participants had to fill a questionnaire. They were asked about their preferred technique, and about their perception of how easy it was to use each technique for undoing moves on both individual objects and on groups of objects. All participants said that Dwell-and-Spring was their preferred technique, which is in accordance with the perceived difficulty of the different experimental tasks. Figure 24 illustrates this perception of difficulty as a score on a five-point Likert scale ([1: very easy ... 5: very difficult]). A pairwise Wilcoxon test with Holm correction reveals that this score significantly differs for all six pairs of tasks. In particular, participants found that *Spring* was easier to use than *List* for tasks on both individual objects and groups of objects.

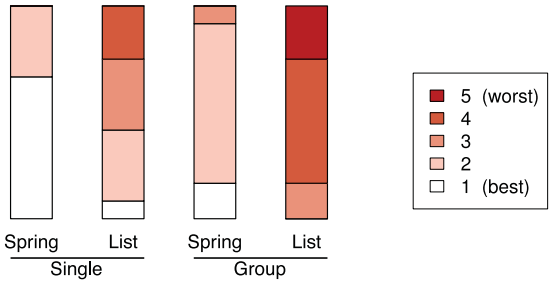


Fig. 24. Participants' subjective rating of difficulty with each TECH for both the single and group *Backward* instructions on a five-point Likert scale (1: best, . . . , 5: worst).

4.6. Summary

Our empirical results suggest that the DND^{-1} model can effectively support undo for direct manipulations on both individual objects and multiple selections. This does not mean that participants understood all the details of DND^{-1}, but that they were able to use it effectively for the layout problem that they had to solve.

All participants expressed a preference for technique *Spring* over *List*. This probably comes from the fact that *List* is a global technique that displays the whole history of all objects, consequently suffering from issues related to the difficulty in identifying the right manipulation in the history. The first issue is the cost of searching in terms of motor control: the longer the history, the larger the number of required scrolling actions. The second issue is the cost of recognizing an object or a group of objects as what can be shown in the thumbnails is limited. Designing more meaningful thumbnails by emphasizing differentiating features between groups would help, but this is difficult to achieve in an application-independent manner since we do not have any information regarding the semantics of the objects manipulated. By using an object as a reference to provide a contextual history in place, *Spring* does not suffer from such problems. In particular, *Spring* scales better than *List* with the size of the history, and better fits with the per-object nature of the DND^{-1} model.

5. RELATED WORK

Most applications handle a history of commands, as well as meta commands UNDO and REDO to navigate it. We review the different models implemented by existing applications to offer such functionalities. However, the DND^{-1} model should not be directly compared to any of these undo models. It is not intended to replace any of them, but rather to complement them. DND^{-1} helps users make a precise drag-and-drop and, if this manipulation actually invokes an application command, it will be handled by the application's undo model. In that regard, DND^{-1} is rather an enhancement to direct manipulation than an undo model.

5.1. Undo Models

In the most common case, the history of commands can be represented as a tree whose nodes are interface states. In the linear undo model, only a single child is associated with each node (the most recently visited one). This makes it impossible to access some previous states of the interface via undo. A few applications like Emacs [Gosling 1982; Yang 1992] provide users with a better experience, in terms of exploration, by making any previous state recoverable. This is achieved by considering UNDO as a regular command that also gets stored in the history. While this model is very powerful—any state can be restored—it remains a bit confusing, as users have to figure out how to

break a flow of undo commands. Also, this model is global: reverting an object to a given past state entails undoing all commands that were performed afterwards, no matter which objects they were performed on. As the US&R model [Vitter 1984], the DnD^{-1} model makes the whole branching of drag-and-drop history accessible. However, DnD^{-1} provides a direct path to any past location, while the US&R model prompts the user each time a redo command is issued on nodes that have several children.

Selective Undo [Prakash and Knister 1994; Berlage 1994] lets users select a command to undo anywhere in the history. The original mechanism [Prakash and Knister 1994] basically removed the command to undo from the history, and then redid the commands that were coming after it in the history. However, this interpretation of the history of commands as a script may not always match users' mental model of the undo command [Cass et al. 2006]. With direct selective undo [Berlage 1994], application developers can implement an undo behavior that is more appropriate in this context, by adding an *inverse* command at the end of the history. The Amulet toolkit [Myers and Kosbie 1996] provides a good architecture to enable support for such undo mechanisms in graphical applications. However, selective undo remains difficult to implement since what the semantics of a reverse command should be is difficult to predict, as it depends on the current interface state. The DnD^{-1} model can be described as direct selective undo for drag-and-drop at the level of individual objects and group of objects. It defines the inverse move to be appended to the history of an object or group as the straight movement from the latter's current location to one of its past locations.

Edwards et al. [2000] experimented with a clever approach to handle undo in Flatland. It consists of introducing commands that will be nested with past commands already stored in the history. As Flatland is a system intended to be used by several users, undo commands can also be region based, regardless of when they happened. This concept of regional undo has also been applied to spreadsheet editors [Kawasaki and Igarashi 2004], interactive large displays [Seifried et al. 2012], code editors [Yoon et al. 2013], VLSI systems [Zhou and Imamiya 1997], sketch-based interfaces [Oe et al. 2013], and painting applications [Myers et al. 2015]. However, to support such an advanced model of undo, applications must take into account a complex system of dependencies and causal relationships. The Causality model [Nancel and Cockburn 2014] identifies the different components and relations that an application should use to store their interaction history in order to support any advanced undo model.

Our model, DnD^{-1}, focuses on direct manipulation. It is not intended to replace the functional undo model of applications, but to run in parallel with them. DnD^{-1} uses a direct selective undo model to provide users with shortcuts that bring objects back to their exact past positions, which is something that would otherwise be difficult to do. In a sense, DnD^{-1} offers a quick way to perform *forward* recovery [Abowd and Dix 1992], that is, to do the right manipulation from the present state to reach a novel state that is similar to a past state for some objects' location. Actually, when users restore object O to a past location P, DnD^{-1} simply sends a message asking the underlying application to translate O from its current location to P. But DnD^{-1} does not have any idea of what the semantics of this object's move are in the application. If this move triggers the invocation of a command in the application, the command will be added to the undo model of that application, no matter the complexity of this model. If this move does not invoke any command, it will simply be ignored by the application's undo model. For instance, in a graphical editor, if DnD^{-1} moves a slider that controls transparency while an object is selected on the canvas, the `setTransparency` command will be stored in the application's command history of the graphical editor. But if there is no object selected when DnD^{-1} moves the slider, the application will ignore it. Object movements performed by DnD^{-1} can have functional effects in a given application. Those effects

can be appended to the application's own history of commands, as any object movement that is performed by users, no matter the complexity of the underlying history model.

5.2. Navigation in Command Histories

A few applications allow users to access their interaction's history. The most basic history representation is a list of text items as in, for example, Adobe Photoshop. Clicking on an item reverts the document back to the state in which it was before this command got invoked, following a linear undo model. The most elaborate history representation is probably the Chronicle system [Grossman et al. 2010] that was also designed for image editing. This system proposes a sophisticated timeline of user actions and serves *chronicle* widgets on demand. A chronicle is an augmented video clip of what happened between two time stamps on a given image area. This approach provides a very exhaustive visualization of commands performed, that supports reflection and communication with others for, for example, creating tutorials. But it is not intended to support undo navigation by restoring a previous state as Rekimoto envisioned with the Time-Machine concept [Rekimoto 1999].

Other approaches have been designed more specifically for selective undo. In the GINA system [Berlage 1994], history is a list of text commands. Users can filter them out using string pattern matching expressions. As it may be difficult to refer to a graphical object textually, users can also drag-and-drop an object onto the list to get the partial history specifically related to it. Other systems represent the history graphically. For instance, Meng et al. [1998] propose two types of history widgets for selective undo, with which users can browse a collection of snapshots that illustrate the different interface states. Representing the history as a picture list has also been investigated in a collaborative web site design tool [Klemmer et al. 2002], in a painting application to perform selective undo [Myers et al. 2015] and for representing users' operations on large and complex visualizations [Heer et al. 2008]. Chimera [Kurlander and Feiner 1992] also exposes the history as a list of graphical panels, allowing users to edit an object in the history and propagate changes to the current state. Chimera relies on an application-dependent visual language to create a graphical representation of the command. This representation conveys the command's semantics better than a scaled-down snapshot of the full interface would, while using less screen real estate. Nakamura and Igarashi [2008] later showed how to adopt such a comic strip metaphor in an application-independent manner, by emphasizing low-level user events and by using the Phosphor afterglow effect [Baudisch et al. 2006] on affected widgets. These systems can be powerful, but browsing collections of pictures requires heavy-weight widgets, which makes them ill suited to performing fast iterations in graphical layout design tasks.

Dwell-and-Spring is a light-weight widget, that works in an application-independent manner, both in terms of graphical rendering and interaction with the widget. This is made possible by the fact that Dwell-and-Spring works at the local level (focus on objects or groups of objects), and has a more focused role: contrary to the heavy-weight widgets mentioned earlier, Dwell-and-Spring is not intended to navigate the whole history of an application, but is designed to navigate the history of past locations of the object it has been invoked on.

5.3. Enhancements to Direct Manipulation

As mentioned earlier, DnD^{-1} focuses on the history of direct manipulations, providing shortcuts to drag-and-drop interactions that move objects and groups of objects. UIMarks [Chapuis and Roussel 2010] also offers shortcuts for graphical interfaces, enabling users to explicitly put some marks on the user interface and configure them to, for example, facilitate invocation of frequently used widgets. The drag-and-pop technique

[Baudisch et al. 2003] also accelerates drag-and-drop interactions by using the direction of movement to predict and bring potential targets close to the dragged object.

Dwell-and-Spring implicitly records the start and end points of any press-drag-release interaction with which users can interact by using a dwell time to trigger a widget. Using the time dimension during a drag-and-drop to avoid having to rely on an additional modality (e.g., the keyboard) is not new and is, for example, used in some systems to reveal the content of a folder when dwelling over its icon. Another example is Scriboli [Hinckley et al. 2005] that suggests the use of dwelling after a lasso selection to pop up a contextual menu.

Allowing users to interact during a press-drag-release interaction can also be addressed with other approaches such as crossing or gesture dynamics. For instance, Fold-and-Drop [Dragicevic 2004] proposes to cross window borders to fold windows during a drag-and-drop to facilitate navigation over windows. The trailing widget [Forlines et al. 2006] follows the cursor and can be grabbed with a quick movement to access a menu. Boomerang [Kobayashi and Igarashi 2007] allows users to suspend a drag-and-drop by using a throwing gesture.

Most graphical editors support an explicit grouping command, but very few systems support more lightweight grouping definitions as $\text{D}_\text{N}\text{D}^{-1}$ does by keeping a trace of multiple selections. Among them, Bubble clusters [Watanabe et al. 2007] automatically cluster objects that have been brought close-by via direct manipulation. The Dynomite notebook application [Wilcox et al. 1997] groups graphical strokes that are spatially and temporally related, to enable users to undo the creation of a set of hand-written letters, in the same spirit as a text editor does with a series of characters that belong to the same word unit. QuickSelect [Su et al. 2009] proposes a similar propagation mechanism for enlarging the current selection based on previous multiple selections. Finally, Handle Flags [Grossman et al. 2009] also proposes extra grouping widgets (handles) that appear when the stylus approaches hand-written strokes. There are as many handles as the number of potential groups, which are computed based on how the strokes are spatially clustered and how the user actually refined proposed groups to perform previous multiple selections. However, if these techniques also keep trace of previous selections as $\text{D}_\text{N}\text{D}^{-1}$ does, they focus on the quick selection of these groups and do not address interaction with their history.

6. DISCUSSION AND LIMITATIONS

$\text{D}_\text{N}\text{D}^{-1}$ provides users with a simple way of putting individual objects and groups of objects back to any location in their respective histories, regardless of what other movements were applied to other objects in the scene. As graphical user interfaces heavily rely on direct manipulation and object movements at large, $\text{D}_\text{N}\text{D}^{-1}$ can be helpful in many cases, at both the system level and the application level: as discussed in this article (Section 2), reciprocal drag-and-drop actions may be useful for desktop and window management, view navigation, direct manipulation in vector graphics editors, and control of widgets such as sliders, scrollbars, color wheels, manipulation handles, etc.

Both the operating system and applications can register object moves to $\text{D}_\text{N}\text{D}^{-1}$. A naive implementation would consist in systematically registering the translation vector between the mouse press and release events (or the finger touch and lift events). However, this will fail in some cases, as graphical objects feature some tolerance with respect to the input movement, in order to make the motor action easier to perform. For example, when a snapping mechanism is implemented, or when users manipulate an object that only has a single degree of freedom (such as a slider using a 2D input device), the input movement performed by the user does not necessarily match the

actual graphical object's movement. In such cases, what should be registered in DND^{-1} is the object's actual move rather than what the input device receives.

Another issue to consider is that some object movements are performed using another input device than the pointing device. For instance, users can scroll a document or move the currently selected object using arrow keys. The resulting object movements can still be registered in DND^{-1}. For example, when scrolling a document by maintaining the `UP` key pressed, the system can send a `moved`$(scrollbar_{knob}, d)$ to the model when the key is released. By doing so, the scrolling operation is featured in DND^{-1}'s history and can be reverted. To avoid registering successive micro-movements, application designers can implement any policy for aggregating moves before sending a message to DND^{-1}. Sending `moved` messages would be triggered as soon as an input event of another type occurs, including those related to DND^{-1}'s invocation. For example, several small moves, applied in sequence to an object using arrow keys, can be aggregated into a single `moved` message with the overall amplitude corresponding to the sum of the small successive moves. The message would get sent as soon as users hit any other key or use a mouse button.

DND^{-1} has been designed to be *application independent*. The downside of this choice is that it has no knowledge of the semantics of object movements in the underlying applications. In some cases, a reciprocal drag-and-drop will thus not correspond to the undo of the command that was invoked by the initial drag-and-drop. For example, when an icon is moved from one window to another, and the window arrangement is changed afterwards, moving back this icon using the inverse displacement might not put it back in the window (and folder) from which it was taken. In that regard, DND^{-1} shares one of the limitations of scripting systems that replay sequences based on cursor movements (an issue raised in [Myers 1998]). This is why the interaction techniques that are proposed to navigate DND^{-1} must provide a clear feedforward of what the result of a reciprocal drag-and-drop will be (as Dwell-and-Spring and DnD-List do), so as to help users anticipate and prevent such errors.

DND^{-1} can also be qualified as *selection agnostic*. There are two main reasons for making this choice. First, an application-independent approach requires establishing a common vocabulary across all applications for the communication protocol between the model and the applications. As selection mechanisms can be different from one application to another,[4] establishing a common protocol that would consider an orthogonal notion of *selection* could have been confusing and would have inherently led to more complexity in the communication protocol. Second, objects belonging to a given selection may have different histories. This is an issue with regional undo that is far from trivial to address [Li and Li 2003; Yoon et al. 2013]. As our approach is focused on drag-and-drop, the notion of group is independent from the current selection, and is only implicitly defined when users move several objects together by mean of a drag-and-drop on a multiple selection. While this is a restriction of our model, it has the advantage of containing its complexity.

Even if the DND^{-1} model itself does not conceptually interfere with the notion of selection in applications, the choice of interaction technique used to navigate the history may still have a side effect on the application's selection, if the input events it relies on interfere with those used by the selection mechanism. For example, a reciprocal drag-and-drop invoked using DnD-List does not require any direct interaction with the application's objects and will thus always let the current selection unmodified. On the contrary, invoking Dwell-and-Spring requires performing a long press directly on

[4]Consider, for example, advanced selection mechanisms like that of Adobe Illustrator, where there are two levels of selection—vertex and shape—in comparison with the absence of the notion of selection on graphical controllers such a slider knob.

the object or group that should be reverted. If the application relies on a click (quick press-release sequence) to select an object, Dwell-and-Spring will not interfere and will leave the selection as is. But if the application relies on a simple press to select an object, the object on which Dwell-and-Spring has been invoked will become the new selection. Also, on touch screens that have a limited input vocabulary, long press events may already be mapped to another action, such as entering an *edit* mode (e.g., for moving app icons on home screens on a smartphone) or invoking a contextual menu. The spring handle is offset with respect to the user's finger and should allow him to ignore it if he does not actually want to trigger a reciprocal drag-and-drop. However, additional UI design work may be required to handle some specific cases where Dwell-and-Spring's footprint may still interfere with the application's controls.

Finally, the extended version of Dwell-and-Spring that we introduced in Section 2 may generate some visual clutter, especially if the object on which it is invoked has been moved a large number of times or if it has been involved in numerous multiple selections. How much visual interference this might cause depends on the nature of the graphical scene below, and on how past locations and target locations are distributed over it. To reduce potential clutter, the DND^{-1} model can be implemented with a maximum length of past steps per group of objects kept in history.[5] But a more interesting solution would consist in designing a graphical footprint that progressively becomes less intrusive as users get increasingly familiar with the widget. The feedforward of a reciprocal drag-and-drop could be displayed as a simple straight line, rather than as a series of springs that explicitly represent the full sequence of drag-and-drop that will get reverted. We believe that a richer graphical representation helps explore the history and understand the widget, but that it is not necessary to effectively use it. Our experience with it makes us think that what is the most important, once familiar with how the widget works, is the overlaid feedforward that helps anticipate the result when brushing through the different spring handles. We plan to enhance DND^{-1}'s communication protocol in order to let the application developer specify, at the object level, the best type of feedforward (a series of lines or springs, or a simple line or spring). This could be especially relevant for objects that are always moved along a single axis, such as scrollbar knobs or sliders.

7. CONCLUSION

We introduced DND^{-1}, a model that keeps track of all past locations of objects that can be moved through drag-and-drop interactions in a graphical user interface, including windows and other view management widgets. We have extended the Dwell-and-Spring technique and combined it with DND^{-1} in order to cover a large number of scenarios in which users need to perform a reciprocal drag-and-drop action and that were identified as limitations in Appert et al. [2012]. With this extended version of Dwell-and-Spring, users can now recover any past location of a single object or group of objects. First, they have access to the whole history of locations that an object has visited, and can thus revert a series of drag-and-drop actions in a single step. Second, they have access to the history of drag-and-drop actions on past multiple object selections, and can thus revert a drag-and-drop on a group of objects without breaking the current active selection.

In the experiments that we conducted, participants were able to understand how the technique works and how it makes it easier to perform reciprocal drag-and-drop actions. They were also able to use it to solve graphical layout tasks in which advanced reciprocal drag-and-drop actions are required. We plan to run a field study that will

[5]The number of past steps has an impact on the widget's footprint, which may overflow the screen when invoked close to its left or bottom edge. In such cases, the widget's layout could be mirrored vertically and/or offset in the spirit of how contextual menus behave in desktop interfaces.

focus on the potential distractions that the widget popping up might cause. However, as discussed in this article, we advocate for an implementation where the graphical representation of Dwell-and-Spring gets lighter as the user becomes familiar with it. The analogy with a physical spring is especially useful in the discovery phase of the technique, but it probably becomes less so when users actually know how to use it and want to optimize time.

REFERENCES

Gregory D. Abowd and Alan J. Dix. 1992. Giving undo attention. *Interact. Comput.* 4, 3 (Dec. 1992), 317–342. DOI:http://dx.doi.org/10.1016/0953-5438(92)90021-7

Jason Alexander, Andy Cockburn, Stephen Fitchett, Carl Gutwin, and Saul Greenberg. 2009. Revisiting read wear: Analysis, design, and evaluation of a footprints scrollbar. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems (CHI'09)*. ACM, 1665–1674. DOI:http://dx.doi.org/10.1145/1518701.1518957

Caroline Appert and Michel Beaudouin-Lafon. 2008. Swingstates: Adding state machines to java and the swing toolkit. *Software: Practice and Experience* 38, 11 (Sep. 2008), 1149–1182. DOI:http://dx.doi.org/10.1002/spe.v38:11

Caroline Appert, Olivier Chapuis, and Emmanuel Pietriga. 2012. Dwell-and-spring: Undo for direct manipulation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'12)*. ACM, 1957–1966. DOI:http://dx.doi.org/10.1145/2207676.2208339

Patrick Baudisch, Edward Cutrell, Dan Robbins, Mary Czerwinski, Peter Tandler, Benjamin Bederson, and Alex Zierlinger. 2003. Drag-and-pop and drag-and-pick: Techniques for accessing remote screen content on touch-and pen-operated systems. In *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction (INTERACT'03)*. IOS & IFIP, 57–64. http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.9085.

Patrick Baudisch, Desney Tan, Maxime Collomb, Dan Robbins, Ken Hinckley, Maneesh Agrawala, Shengdong Zhao, and Gonzalo Ramos. 2006. Phosphor: Explaining transitions in the user interface using afterglow effects. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST'06)*. ACM, 169–178. DOI:http://dx.doi.org/10.1145/1166253.1166280

Thomas Berlage. 1994. A selective undo mechanism for graphical user interfaces based on command objects. *ACM Transactions on Computer-Human Interaction* 1, 3 (Sep. 1994), 269–294. DOI:http://dx.doi.org/10.1145/196699.196721

Aaron G. Cass, Chris S. T. Fernandes, and Andrew Polidore. 2006. An empirical evaluation of undo mechanisms. In *Proceedings of the 4th Nordic Conference on Human-Computer Interaction (NordiCHI'06)*. ACM, 19–27. DOI:http://dx.doi.org/10.1145/1182475.1182478

Olivier Chapuis and Nicolas Roussel. 2010. UIMarks: Quick graphical interaction with specific targets. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology (UIST'10)*. ACM, 173–182. DOI:http://dx.doi.org/10.1145/1866029.1866057

Pierre Dragicevic. 2004. Combining crossing-based and paper-based interaction paradigms for dragging and dropping between overlapping windows. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST'04)*. ACM, 193–196. http://doi.acm.org/10.1145/1029632.1029667

W. Keith Edwards, Takeo Igarashi, Anthony LaMarca, and Elizabeth D. Mynatt. 2000. A temporal model for multi-level undo and redo. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST'00)*. ACM, 31–40. DOI:http://dx.doi.org/10.1145/354401.354409

W. Keith Edwards and Elizabeth D. Mynatt. 1997. Timewarp: Techniques for autonomous collaboration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'97)*. ACM, 218–225. DOI:http://dx.doi.org/10.1145/258549.258710

Clifton Forlines, Daniel Vogel, and Ravin Balakrishnan. 2006. Hybridpointing: Fluid switching between absolute and relative pointing with a direct input device. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST'06)*. ACM, 211–220. DOI:http://dx.doi.org/10.1145/1166253.1166286

James Gosling. 1982. *Unix Emacs Reference Manual*. Carnegie-Mellon University, Pittsburgh, PA, USA.

Tovi Grossman, Patrick Baudisch, and Ken Hinckley. 2009. Handle flags: Efficient and flexible selections for inking applications. In *Proceedings of Graphics Interface (GI'09)*. Canadian Information Processing Society, 167–174. http://dl.acm.org/citation.cfm?id=1555880.1555918

Tovi Grossman, Justin Matejka, and George Fitzmaurice. 2010. Chronicle: Capture, exploration, and playback of document workflow histories. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology (UIST'10)*. ACM, 143–152. DOI:http://dx.doi.org/10.1145/1866029.1866054

Jeffrey Heer, Jock Mackinlay, Chris Stolte, and Maneesh Agrawala. 2008. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE Trans. Vis. Comput. Graph.* 14, 6 (Nov. 2008), 1189–1196. DOI: http://dx.doi.org/10.1109/TVCG.2008.137

Ken Hinckley, Patrick Baudisch, Gonzalo Ramos, and Francois Guimbretiere. 2005. Design and analysis of delimiters for selection-action pen gesture phrases in Scriboli. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'05)*. ACM, 451–460. DOI: http://dx.doi.org/10.1145/1054972.1055035

Akrivi Katifori, George Lepouras, Alan Dix, and Azrina Kamaruddin. 2008. Evaluating the significance of the desktop area in everyday computer use. In *First International Conference on Advances in Computer-Human Interaction (ACHI'08)*. IEEE, 31–38. DOI: http://dx.doi.org/10.1109/ACHI.2008.27

Yoshinori Kawasaki and Takeo Igarashi. 2004. Regional undo for spreadsheets. In *UIST'04 Demonstration Abstract*. ACM, 2 pages. http://www-ui.is.s.u-tokyo.ac.jp/~kwsk/undo/kawasaki_uist04_regional.pdf.

Scott R. Klemmer, Michael Thomsen, Ethan Phelps-Goodman, Robert Lee, and James A. Landay. 2002. Where do web sites come from?: Capturing and interacting with design history. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'02)*. ACM, 1–8. DOI: http://dx.doi.org/10.1145/503376.503378

Andrew J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. 2006. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Trans. Softw. Eng.* 32, 12 (Dec. 2006), 971–987. DOI: http://dx.doi.org/10.1109/TSE.2006.116

Masatomo Kobayashi and Takeo Igarashi. 2007. Boomerang: Suspendable drag-and-drop interactions based on a throw-and-catch metaphor. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST'07)*. ACM, 187–190. DOI: http://dx.doi.org/10.1145/1294211.1294243

David Kurlander and Steven Feiner. 1988. Editable graphical histories. In *IEEE Workshop on Visual Languages*. IEEE, 127–134. DOI: http://dx.doi.org/10.1109/WVL.1988.18020

David Kurlander and Steven Feiner. 1992. A history-based macro by example system. In *Proceedings of the 5th Annual ACM symposium on User Interface Software and Technology (UIST'92)*. ACM, 99–106. DOI: http://dx.doi.org/10.1145/142621.142633

Henry A. Landsberger. 1958. *Hawthorne Revisited: Management and the Worker, Its Critics, and Developments in Human Relations in Industry.* Cornell University, Ithaca, New York, NY, USA.

Rui Li and Du Li. 2003. A regional undo mechanism for text editing. In *Proceedings of the 5th International Workshop on Collaborative Editing Systems (IWCES'03)*. Citeseer. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.4754&rep=rep1&type=pdf.

Chii Meng, Motohiro Yasue, Atsumi Imamiya, and Xiaoyang Mao. 1998. Visualizing histories for selective undo and redo. In *Proceedings of the Third Asian Pacific Computer and Human Interaction (APCHI'98)*. IEEE, 459–464. DOI: http://dx.doi.org/10.1109/APCHI.1998.704487

Brad A. Myers. 1998. Scripting graphical applications by demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'98)*. ACM Press/Addison-Wesley Publishing Co., 534–541. DOI: http://dx.doi.org/10.1145/274644.274716

Brad A. Myers and David S. Kosbie. 1996. Reusable hierarchical command objects. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'96)*. ACM, 260–267. DOI: http://dx.doi.org/ 10.1145/238386.238526

Brad A. Myers, Ashley Lai, Tam Minh Le, YoungSeok Yoon, Andrew Faulring, and Joel Brandt. 2015. Selective undo support for painting applications. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI'15)*. ACM, 4227–4236. DOI: http://dx.doi.org/10.1145/2702123.2702543

Toshio Nakamura and Takeo Igarashi. 2008. An application-independent system for visualizing user operation history. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology (UIST'08)*. ACM, 23–32. DOI: http://dx.doi.org/10.1145/1449715.1449721

Mathieu Nancel and Andy Cockburn. 2014. Causality: A conceptual model of interaction history. In *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems (CHI'14)*. ACM, 1777–1786. DOI: http://dx.doi.org/10.1145/2556288.2556990

Tatsuhito Oe, Buntarou Shizuki, and Jiro Tanaka. 2013. Undo/redo by trajectory. In *Proceedings of the 15th International Conference on Human-Computer Interaction: Interaction Modalities and Techniques - Volume Part IV (HCI'13)*. Springer-Verlag, 712–721. DOI: http://dx.doi.org/10.1007/978-3-642-39330-3_77

Atul Prakash and Michael J. Knister. 1994. A framework for undoing actions in collaborative systems. *ACM Transactions on Computer-Human Interaction* 1, 4 (Dec. 1994), 295–330. DOI: http://dx.doi.org/10.1145/198425.198427

Jun Rekimoto. 1999. Time-machine computing: A time-centric approach for the information environment. In *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology (UIST'99)*. ACM, 45–54. DOI: http://dx.doi.org/10.1145/320719.322582

Dario D Salvucci. 2001. An integrated model of eye movements and visual encoding. *Cogn. Syst. Res.* 1, 4 (Feb. 2001), 201–220. DOI:http://dx.doi.org/10.1016/S1389-0417(00)00015-2

Thomas Seifried, Christian Rendl, Michael Haller, and Stacey Scott. 2012. Regional undo/redo techniques for large interactive surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'12)*. ACM, 2855–2864. DOI:http://dx.doi.org/10.1145/2207676.2208690

Katie A. Siek, Yvonne Rogers, and Kay H. Connelly. 2005. Fat finger worries: How older and younger users physically interact with PDAs. In *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction (INTERACT'05)*. Springer-Verlag, 267–280. DOI:http://dx.doi.org/10.1007/11555261_24

Sara L. Su, Sylvain Paris, and Frédo Durand. 2009. QuickSelect: History-based selection expansion. In *Proceedings of Graphics Interface 2009 (GI'09)*. CIPS, 215–221. http://dl.acm.org/citation.cfm?id=1555880.1555929

Michael Terry, Elizabeth D. Mynatt, Kumiyo Nakakoji, and Yasuhiro Yamamoto. 2004. Variation in element and action: Supporting simultaneous development of alternative solutions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'04)*. ACM, 711–718. DOI:http://dx.doi.org/10.1145/985692.985782

Jeffrey S. Vitter. 1984. US&R: A new framework for redoing (extended abstract). *ACM SIGSOFT Software Engineering Notes* 9, 3 (April 1984), 168–176. DOI:http://dx.doi.org/10.1145/390010.808262

Nayuko Watanabe, Motoi Washida, and Takeo Igarashi. 2007. Bubble clusters: An interface for manipulating spatial aggregation of graphical objects. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST'07)*. ACM, 173–182. DOI:http://dx.doi.org/10.1145/1294211.1294241

Lynn D. Wilcox, Bill N. Schilit, and Nitin Sawhney. 1997. Dynomite: A dynamically organized ink and audio notebook. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'97)*. ACM, 186–193. DOI:http://dx.doi.org/10.1145/258549.258700

Yiya Yang. 1992. Anatomy of the design of an undo support facility. *Int. J. Man-Machine Stud.* 36, 1 (Jan. 1992), 81–95. DOI:http://dx.doi.org/10.1016/0020-7373(92)90053-N

YoungSeok Yoon, Brad A. Myers, and Sebon Koo. 2013. Visualization of fine-grained code change history. In *Proceedings of the Symposium on Visual Languages and Human-Centric Computing (VL/HCC'13)*. IEEE, 119–126. DOI:http://dx.doi.org/10.1109/VLHCC.2013.6645254

Chunbo Zhou and Atsumi Imamiya. 1997. Object-based nonlinear undo model. In *Proceedings of the Computer Software and Applications Conference (COMPSAC'97)*. IEEE, 50–55. DOI:http://dx.doi.org/10.1109/CMPSAC.1997.624739

Received May 2015; revised July 2015; accepted August 2015

CHI 2009 ~ Gesture UIs April 9th, 2009 ~ Boston, MA, USA

Using Strokes as Command Shortcuts: Cognitive Benefits and Toolkit Support

Caroline Appert[1,2]
[2]LRI - Université Paris-Sud & CNRS
Orsay, France
appert@lri.fr

Shumin Zhai[1]
[1]IBM Almaden Research Center
San Jose, CA, USA
zhai@us.ibm.com

ABSTRACT
This paper investigates using stroke gestures as shortcuts to menu selection. We first experimentally measured the performance and ease of learning of stroke shortcuts in comparison to keyboard shortcuts when there is no mnemonic link between the shortcut and the command. While both types of shortcuts had the same level of performance with enough practice, stroke shortcuts had substantial cognitive advantages in learning and recall. With the same amount of practice, users could successfully recall more shortcuts and make fewer errors with stroke shortcuts than with keyboard shortcuts. The second half of the paper focuses on UI development support and articulates guidelines for toolkits to implement stroke shortcuts in a wide range of software applications. We illustrate how to apply these guidelines by introducing the *Stroke Shortcuts Toolkit* (SST) which is a library for adding stroke shortcuts to Java Swing applications with just a few lines of code.

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: User Interfaces—*Graphical user interfaces*

Author Keywords
Gesture, Stroke, Shortcuts, Toolkit

INTRODUCTION
Invoking a command in a graphical user interface can usually be done through three different means: finding and clicking its label in a *menu*, finding and clicking its *icon* (e.g. toolbar buttons) or recalling and activating a *shortcut*. The most common type of shortcuts is typing a sequence of keys, known as keyboard shortcuts or hotkeys. Gesturing strokes is an alternative or complementary type of shortcuts that is also used but only in a few products. For example, the Opera[1] and the Firefox[2] web browsers have a set of strokes to open a new window, close a tab, etc. However, while the UI community has a longstanding interest in strokes as an interaction medium [24, 23, 7] and developed several research GUI prototypes, e.g. [17, 25], our understanding in this subject is still rather limited. In particular, there is a lack of basic cognitive and motor performance measurements of stroke gestures as command shortcuts in comparison to the standard keyboard shortcuts.

Traditional keyboard shortcuts have their limits in many situations. First, studies show that users often have difficulty to transition from menu selection to keyboard shortcuts [14]. Second, keyboard shortcuts may not be convenient to use, particularly for a growing number of non-traditional computing and communication devices. For example the iPhone and the pen-based Tablet PCs either have no keyboard at all or require the user to manipulate the screen to make the keyboard accessible. Enabling the users to efficiently trigger a command with a stroke gesture would overcome some of these problems, complement our current interaction vocabulary and enhance user experience.

In this paper we formulate and investigate the following hypothesis: stroke shortcuts may have a cognitive advantage in that they are easier to memorize than keyboard shortcuts. To better support recall, designers should make the shortcuts as analogous or mnemonic to the command name or meaning as possible (See related work on icons [19]). However *arbitrary mappings* are unavoidable since many concepts in the digital world do not offer a direct metaphor to the physical world. Interestingly, because strokes are spatial and iconic, which makes richer and deeper processing possible in human memory [3] even if the mapping is arbitrary, we hypothesize that stroke shortcuts could have cognitive (learning and recall) advantages over keyboard shortcuts.

We test this hypothesis from a user behavior perspective. Complementarily from a system design and development perspective, we articulate a set of guidelines for developing easy to use stroke shortcuts toolkits. As a first step in this direction, we present the *Stroke Shortcuts Toolkit* (SST) that integrates stroke shortcuts in a widely used development environment, the Java programming language and its Swing toolkit. Combining both types of contributions, we hope to encourage broader and faster adoption of stroke shortcuts in real world applications.

[1]http://www.opera.com/products/desktop/mouse/
[2]http://www.mousegestures.org/

CHI 2009, April 4 - 9, 2009, Boston, Massachusetts, USA.

RELATED WORK

Strokes and commands

The best known work on using strokes to activate commands is probably the marking menus designed by Kurtenbach and Buxton [11]. A marking menu is a circular menu displayed after a delay so expert users who have already learned the menu layout can stroke ahead without the visual feedback. Extensions to marking menus include simple mark hierarchical marking menus [27] and the Hotbox [12].

Kurtenbach and colleagues also proposed a technique that can handle a larger vocabulary of gestures in the Tivoli system [13]. In that system, if the user does not know which gestures are available or how to gesture a command, he presses down the pen and waits for a *crib-sheet* to display the commands and their corresponding gesture strokes available in the current context. The Fluid Inking [25] system proposes a similar approach: to discover the available strokes, users invoke a marking menu in which an item is composed of a command name and the corresponding stroke description (in words), such as "Select (Lasso)". Neither crib-sheets in Tivoli nor the augmented marking menus in Fluid Inking have been experimentally evaluated.

Command strokes (CSs) proposed by Kristensson and Zhai [10] took another approach. CSs are based on the ShapeWriter text input system [26, 9]. With ShapeWriter, instead of tapping a sequence of soft keys the user draws a stroke that approximately links the letters of the intended word on a soft keyboard. To invoke a command, the user shape writes the name, or a part of the name, of a command prefixed by the special *Command* key. With CSs, users were able to invoke a command 1.6 times faster than selecting an item in a pull-down menu. Obviously, CSs require the presence of a soft keyboard which takes some valuable screen space.

Shortcuts and memorization

Most studies on command input focus on the execution phase and bypass the command recall phase by using experimental tasks where the stimuli and the responses are congruent and direct. For example, participants select an item in a marking menu in response to a given direction (e.g., N, W, E, S) in [11]. The same is true in [5, 8] where participants selected a color swatch (in a toolglass, flow menu or a palette) in response to a colored dot. In a study that did involve indirect mapping between the stimulus and the response, Odell and colleagues [18] compared toolglasses, marking menus and keyboard shortcuts to invoke a set of three commands (oval, rectangle, line). In particular, they compared two sets of keyboard shortcuts. One used the first letter of each command ('O', 'R' and 'L') while the second used three abstract numeric keys ('1', '2' and '3'). The latter assignment was the most efficient on average in their study.

Grossman et al. [4] recently conducted what is possibly the most comprehensive study to date on learning arbitrary associations between commands and keyboard shortcuts. In their task, the stimulus was a graphical icon of a familiar object and the action was a keyboard shortcut composed of one modifier key and an alphabetic key which was not a letter contained in the object name depicted by the icon. They explored a number of display methods to accelerate user learning of keyboard shortcuts but found them ineffective except for two rather forceful ones: one augments a menu command selection with the speech audio of the corresponding hot keys; the other simply disables the menu selection ability (rendering the menu a crib sheet of shortcuts) forcing the user to rehearse the keyboard shortcuts.

Despite these and other related works, using strokes as shortcuts to commands still requires investigations. First, researchers have never measured users' ability to learn associations between a stroke and a command therefore the understanding of strokes as commands is rather limited. Second, practitioners do not have the right tools to easily implement stroke-based commands and integrate them into mainstream products. The following sections address these understanding and practical aspects respectively.

STROKE SHORTCUTS VS KEYBOARD SHORTCUTS

In this section we evaluate the performance of stroke shortcuts relative to that of keyboard shortcuts. This comparison to keyboard shortcuts is not meant to be a competition, but rather to use keyboard shortcuts as a baseline control condition. Since the use of shortcuts largely depend on their ease of learning, we focus our study on learning aspects involved in both types of shortcuts. We also limit our study to the general case of arbitrary mappings between the commands and the shortcuts, namely mappings without direct mnemonic association in either condition. This decision was based on several considerations. First, a learning experiment takes time to do well even when it is focused. Second, the special cases of mnemonic mapping, which should be maximized in actual design, is rather limited in number. For example the usual way of making a keyboard shortcut mnemonic is to use the first letter of the command name. However this rule makes interface developers quickly run into conflicts: in fact the small set of five common commands {Cut, Copy, Paste, Save and Print} already exhibits two conflicts. Also, for non-English speakers, the same command may have different names in different languages yet it has the same keyboard shortcut (which is probably a reasonable design choice for consistency). Third, stroke shortcuts can always be made as mnemonics as keyboard shortcuts by choosing letter-shaped strokes. Learning required in that case is probably limited.

Participants

Fourteen adults, two females and twelve males, 26 to 44 years old (mean = 31.8, SD = 4.7), participated in our experiment. They were rewarded with a lunch coupon.

Apparatus

The apparatus was a 1.5GHz Pentium M ThinkPad Tablet PC with a 13-inch tablet digitizer screen at 1024×768 resolution. The experiment window was set in full-screen mode. Participants used the stylus to stroke gestures and could hold the tablet at any time if it felt more comfortable. The set of strokes was designed by the experimenter and the stroke recognizer was based on Rubine's algorithm [20] trained with a set of 15 examples per stroke input by the experimenter.

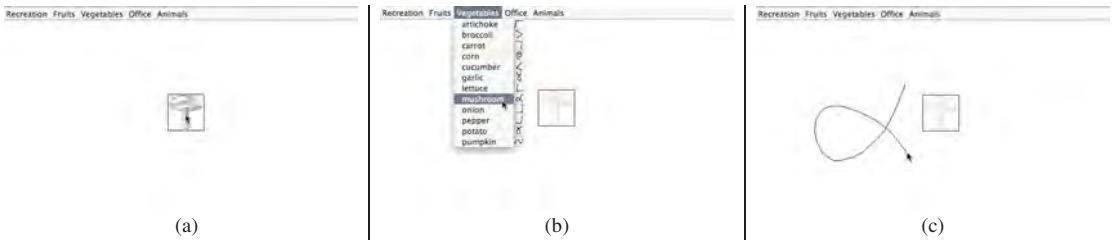


Figure 1. The task used in the experiment: (a) a command stimulus appears as an icon, the participant clicks on it (this makes the icon become semi-transparent) (b) the participant invokes the command through a menu, (c) or through a shortcut (a stroke shortcut in this case)

ICON	Keys	Stroke	ICON	Keys	Stroke
	Shift+W			Ctrl+W	
	Shift+D			Ctrl+D	
...	...	...	...	...	...

Figure 2. An excerpt of the mappings used in the experiment.

m_1	(Karate,12) ; (Pumpkin,12) ; (Hockey,6) ; (Mushroom,6) ; ... ; (Keyboard,2) ; (Garlic,2) ; (Dinosaur,1) ; (Pineapple,1)
m_2	(Karate,6) ; (Pumpkin,6) ; (Hockey,4) ; (Mushroom,4) ; ... ; (Keyboard,1) ; (Garlic,1) ; (Dinosaur,12) ; (Pineapple,12)
...	

Figure 3. Examples of frequency assignments used in the experiment.

Task

We modeled our experimental task after Grossman et al. [4] which was the most recent and most complete study to date on learning keyboard shortcuts. The task required the participants to activate a set of commands that were accessible through both menus and shortcuts. Once a command stimulus (i.e. a graphical icon, as in [4]) was displayed in the center of the screen, the participant was asked to first click on the icon (Figure 1-(a)) and then execute a corresponding command as quickly as possible through either menu selection (Figure 1-(b)) or a shortcut activation (by drawing a stroke or pressing hot keys, depending on the experimental condition) (Figure 1-(c)). The click on the icon at the beginning of each trial prevented the participant from keeping the mouse cursor in the menu area to only interact with menu items. Both types of shortcuts were displayed *on-line* beside the corresponding menu items. The participant was explicitly told to learn as many shortcuts as possible. In case he did not know or remember a shortcut, he can use the menu to directly select the command or look at the shortcut.

The keyboard shortcuts were assigned in accordance to the rule used in [4]: they were composed of a sequence of a modifier key followed by an alphabetic key that was not the first or last letter of the name of the object. To reflect a necessary difficulty in practical keyboard assignments, the same alphabetic key preceded by two different modifier keys (Ctrl or Shift) constituted two different commands. To reduce a potential bias, we reproduced this potential pair confusion in stroke shortcuts as well: the same shaped stroke with two different orientations activated two different commands. Table 2 shows a sample of the icons and the two types of shortcut we tested. To minimize the influence of the participants' personal experience, commands tested were not those in common software applications but rather objects and activities of everyday life organized into five menus (categories): Animals, Fruits, Office, Recreation and Vegetables. Each menu contained 12 menu items resulting in a total of 60 items. In order to have enough trial repetitions, the participants had to activate a subset of 14 commands during the experiment. Note that the rest of the 60 items were also assigned shortcuts and served as distracters both to the participants and to the stroke recognizer.

To reflect the fact that some commands are invoked more frequently than others in real applications, we assigned different frequencies to different commands for each participant, as in [4]. The fourteen frequencies, defined as the number of occurrence per block of trials, were (12, 12, 6, 6, 4, 4, 3, 3, 2, 2, 2, 1, 1). We used 7 frequency assignments (m_1, ..., m_7), balanced across the 14 commands (Figure 3), and assigned each mapping to a group of two participants. The 7 different mappings we used ensured that we collected the same total number of measures per command in the overall experiment.

Design

Participants had to complete 12 blocks of 60 trials organized into two sessions on two consecutive days. Presentation order for commands within a block was randomized while respecting the assigned frequencies. Participants had to perform 10 blocks on the first day and two blocks on the second day. In the first two blocks on the first day (*warm-up*), the only way of invoking a command was through menu selection so participants could become familiar with the menu layout and the experimental task ($Shortcut$ = $None$). In the 8 other blocks (*test*) on the first day commands could be invoked through either menu selection or shortcuts. These blocks were divided into two sets: in 4 blocks the shortcuts were keyboard-based ($Shortcut$ = $Keyboard$) and in the other 4 they were stroke-based ($Shortcut$ = $Stroke$). Within a group of two participants assigned to the same frequency mapping m_i, one experienced the test blocks in the order $Keyboard$ - $Stroke$ while the other the order $Stroke$ - $Keyboard$. For the 11th and 12th blocks on the second

day (*re-test*) both types of shortcuts were available and the participants were told to use what was most convenient for each trial ($Shortcut$ = $Both$).

Before starting the first session, the experimenter distributed instructions explaining the task and asking the participants to learn as many shortcuts as they could in order to complete the study as quickly as possible. Participants were not told what would be in the second session so that they would not consciously rehearse shortcuts during the break between the two days. On the second day, they were told to complete the last two blocks as quickly as possible by using the method of their choice for each trial (*re-test* blocks). Throughout the experiment the participants could rest not only between blocks but after every 20 trials within a block. At the end of the experiment, they were given a questionnaire about their background (if and how much they used keyboard shortcuts and if they had already used a gesture-based interface) and their preference between the two types of shortcuts based on their experience in the study. The final part of the questionnaire was a table organized into three columns "Icon/command", "Keyboard shortcut" and "Stroke shortcut", similar to Figure 2, but with only cells of the first column filled. The participants had to write down the two types of shortcuts as they recalled them for every icon they saw during the experiment. They also had to indicate a confidence level between 0 (don't remember at all) and 1 (totally confident) to each shortcut.

Hypothesis

As mentioned in the Introduction, we hypothesize that an arbitrary association between a command and a shortcut is more learnable when this shortcut is a stroke rather than a combination of keys. This hypothesis is based on two arguments, one in favor of strokes and one against key combinations:

- It has been previously postulated in the literature that strokes (also known as gestures or marks) have various possible advantages including being iconic [17]. The fact that contemporary software applications widely use icons indicates that many users are able to build arbitrary mappings between commands and icons. For example, a compass icon is used for launching the Safari browser, a curved arrow is for reversing the last action (undo) and a floppy disk, now an obsolete concept, is used as an icon for saving the current file on the hard drive. More theoretically, human memory research suggests that deeper or more levels of encoding and processing help memory [3]. The spatial and iconic information in a stroke may better enable users to imagine (encode) an association between the stroke and its corresponding commands. For example, when an upward straight stroke was arbitrarily assigned to the object "bat", the user may make up the association of a bat flying upwards.
- Letters are special symbols which are strongly linked to words in which they appear so it can be very difficult to link a letter to a command name that does not start with this letter (such as Ctrl+V for paste).

Results

We used three measures in our analyses:

- $Time$, the total time interval (in ms) from the command icon being presented to the completion of the correct command. Note that this was the total duration including both recall and execution time.
- $Errors$, the number of times the participant entered a wrong shortcut before typing or stroking the correct one.
- $Recall$, a binary measure which is equal to 1 when the participant was able to activate the right command with a shortcut without opening the menu and without any error, 0 otherwise.

The main results lie in the measures collected for the *test* blocks in which $Shortcut$=$Stroke$ and $Shortcut$=$Keyboard$ were balanced and compared. Variance analysis on the $Time$, $Error$ and $Recall$ data showed that the interaction effect of $Presentation\ Order \times Shortcut$ was not significant, confirming that the counterbalancing strategy for minimizing presentation order effect was successful. We also verified that the participants followed the instructions and indeed used the shortcuts instead of relying solely on menu selection. Across the 8 blocks they used shortcuts in 96% of the trials for $Shortcut = Stroke$ and in 88.5% of the trials for $Shortcut = Keyboard$, indicating that the participants switched from menu selection to stroke shortcuts more often or earlier than to keyboard shortcuts. This measure already suggests that stroke shortcuts were easier to learn.

Our hypothesis was also supported by the three main measures from the 8 *test* blocks. First, on average the trials in the $Stroke$ condition were completed faster than the trials in the $Keyboard$ condition ($F_{1,13} = 36$, $p < .0001$). Second, the participants had significantly better recall scores with stroke shortcuts than with keyboard shortcuts ($F_{1,13} = 32$, $p < .0001$). Third, the participants made significantly fewer errors with stroke shortcuts than with keyboard shortcuts ($F_{1,13} = 23$, $p < .0003$)[3]. Figure 4 summarizes these results.

To compare the learning speed for each type of shortcut, we plotted the mean $Time$ and $Recall$ performances as a function of the number of times an item was tested from the beginning of the experiment (Figure 5). The results also supported our hypothesis: $Time$ decreased faster with stroke shortcuts than with keyboard shortcuts; $Recall$ accuracy increased faster with stroke shortcuts than with keyboard shortcuts. Note that the performance difference between the two types of shortcuts is primarily cognitive (learning and recalling the shortcuts). With enough practice, when the user performance is more likely to be limited by motor execution (around the 25th exposure in this experiment), the difference in both time and recall between the two types of shortcuts became indistinguishable.

Data collected in the *re-test* blocks on the second day allowed us to evaluate users' memory retention of the short-

[3]Note that this result is actually even more favorable to stroke shortcuts since some participants reported that some of their errors were due to a lack of accuracy in the stroke recognizer.

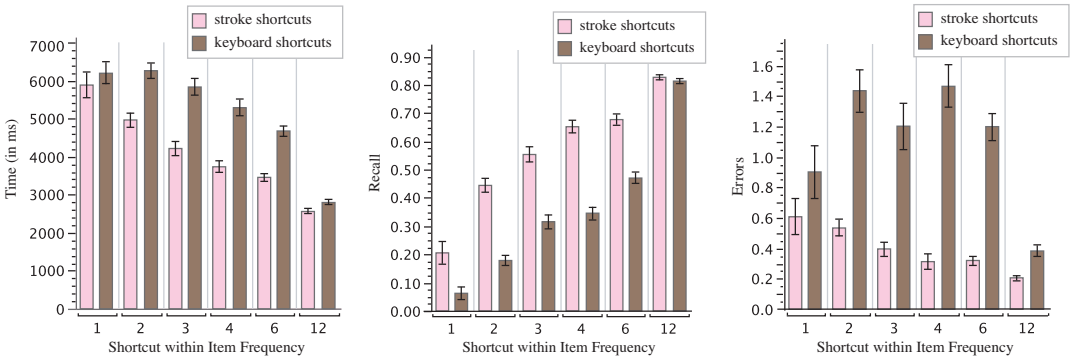


Figure 4. $Time$, $Recall$ **and** $Error$ **by** $Shortcut \times Frequency$

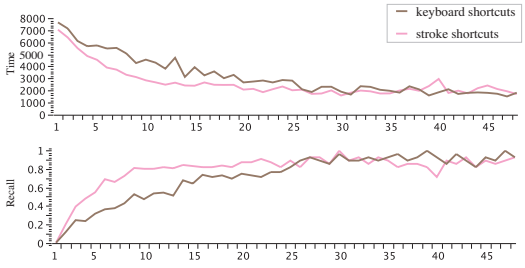


Figure 5. Time and Recall performance according to the number of times a command has been seen

Figure 6. Percentage of use of each technique in *retest* on the second day (by participant)

cuts learned and to see which type of shortcuts they preferred. Figure 6 shows each individual participants percentage of use for each technique ($Keyboard$, $Stroke$ and $Menu$). Although varied by individual, on average significantly more stroke shortcuts than keyboard shortcuts were used ($F_{1,13} = 43$, $p < .0001$). The overall mean percentages of use for the three techniques were: 77.7 % $Stroke$, 20.3 % $Keyboard$, 2 % $Menu$.

Finally, answers to the post hoc questionnaire showed that all of the participants had intensive previous experience with keyboard shortcuts in their everyday activity (about 15-20 different shortcuts) and that none of them had ever used strokes. Despite this experience bias in favor of keyboard shortcuts, the answers to the final question where they had to fill the table revealed that they had learned stroke shortcuts better than keyboard shortcuts in this study. On average 11.6 stroke shortcuts and 4 keyboard shortcuts were correctly answered. The participants' confidence level was also higher with stroke shortcuts (11.7/14 on average; 14 means complete confidence on all commands tested) than with keyboard shortcuts (4.2/14 on average).

The participants' open remarks confirmed some of the analyses that led to our hypothesis. Strokes gave them richer clues to make up an association (more levels of processing) between a command and its arbitrarily assigned stroke: "I thought of this stroke as fish because the shape's stroke makes me think about a basin" or "I associated this stroke with a jump and I see karate as a sport where people jump". Interestingly, no two people mentioned the same trick to associate a stroke with a command.

In summary, although the purpose of stroke shortcuts is not to replace or compete against either menu selection or keyboard shortcuts, the experiment clearly shows that stroke shortcuts can be as efficient as or more advantageous than keyboard shortcuts. After enough practice, the total trial completion times including both recall and execution were indistinguishable between the two types of shortcuts. However with the same fixed amount of practice, the participants successfully recalled more shortcuts and made fewer errors in the $Stroke$ condition than in the $Keyboard$ condition. On the second day the participants chose to use stroke shortcuts significantly more often than keyboard shortcuts, and correctly recalled about 3 times as many stroke shortcuts as keyboard shortcuts.

STROKE SHORTCUTS AND UI DEVELOPMENT

The study we conducted suggests that stroke gestures can be used as command shortcuts that are as effective as, or even more effective than, keyboard shortcuts. However, implementing stroke shortcuts in real applications is more challenging than implementing keyboard shortcuts because com-

monly used graphical toolkits do not support stroke input. In order to encourage the adoption of stroke shortcuts in a wide range of applications, we articulate a set of guidelines for stroke shortcuts development based on an analysis of previous literature and our own experience. We then introduce *Stroke Shortcuts Toolkit* (SST), an extension to Java Swing that we have developed to support stroke shortcuts.

Guidelines to make stroke shortcuts easy to implement

(1) template-based recognition algorithm

Several tools for implementing stroke recognition already exist. For example, Satin [7] is a Java toolkit that uses a special component, a *Sheet*, on which strokes can be drawn and sent to a recognizer. Satin's recognizer is built on Rubine's training-based recognition algorithm [20]. To accurately train the different features representing a stroke in the algorithm (e.g. size, orientation, speed), enough examples (about 15) must be given for each stroke and these examples must reflect the variance along these feature dimensions. Either the interface designer or the end user has to enter these examples. On the one hand, it is difficult for the designer to foresee the stroke variations that can occur among all users [4]. On the other hand, if the training task is left to the end user, another set of difficulties arises: when and how should the interface ask the user to enter these examples? Users tend to be reluctant to invest time and effort upfront to train or adjust software before using it. A third approach is to train the recognizer with examples from a large standardized stroke corpus. However, without a firmly established user community and stroke standard, such a corpus is difficult to collect.

While training-based recognition handles different styles and habits in natural handwriting fairly well, it may not be necessary with novel stroke gestures that can be explicitly defined with unique templates. In fact the work of ShapeWriter has shown that template-based recognition can handle thousands of stroke gestures if multiple channels of information are appropriately integrated ([9]). More recently, Wobbrock et al [22] formally evaluated template matching methods (with and without elasticity [21]) in comparison to Rubine's algorithm for recognizing strokes similar to those used in this paper. In their favored method, the $1 recognizer, each template is represented by a set of equally spaced points, scaled to a given bounding box and rotated to an *indicative* angle (i.e. the angle formed between the first point and the centroid of the template). When a stroke is entered, it is resampled, scaled and rotated to its indicative angle so its distance to each template can be computed by summing the distances between pairs of corresponding points. Their experiment results show that such a simple template matching approach in fact has better performance than Rubine's algorithm. By eliminating training issues while still being accurate, ***a template-based algorithm is the best choice to implement stroke shortcuts***.

(2) Simplify the task of designing a set of strokes

[4]This was a challenge that we faced during the experiment presented in the previous section in which the Rubine's recognizer was trained by the experimenter.

In [15], Long et al. studied the task of designing a set of strokes for Rubine's recognition algorithm. Participants were asked to obtain the best recognition accuracy they could. Results showed that it is a very difficult task and no one participant was able to go beyond the 95.4% recognition rate. A typical problem they observed is that participants tend to add strokes that are too similar to those already defined. This shows that ***designers' imagination must be stimulated by providing them with a design space*** for defining a set of strokes for the commands of the application they want to enhance.

Most of the other problems Long et al. identify in the task of defining a set of strokes are specifically dependent on Rubine's algorithm [15]. They concluded that participants (including computer science students) were not able to get a high recognition rate because they do not understand the principles of the algorithm. It is very difficult to get a mental model of how Rubine's algorithm works: it represents a gesture as a set of features and not as a series of points and uses a covariance matrix that evolves each time an example or a new stroke class is added with the potential unpredictable consequence of degrading the recognition accuracy between the old stroke classes. The study in [15] suggests that ***the underlying mechanisms in the recognition engine must be transparent to the interface designers***.The simple shape matching algorithm used in the $1 recognizer is probably better from that perspective. However, the rotation independence property can be hard to anticipate since the notion of indicative angle is not straightforward. This rotation step can also be a limitation: for example, the rotationa-independent recognizer cannot distinguish among lines in different directions which are convenient for invoking reciprocal commands (e.g. "previous" and "next" in a web browser). Furthermore, Long et al. [16] showed that stroke initial angle and angle formed by first and last stroke points are important to perceive two strokes as different while the rotation independence limits variations that can be expressed along these two dimensions. Thus the most comprehensive and permissive recognition algorithm is probably the one used in the $1 recognizer without the rotation independence which in fact was also the algorithm used in the shape channel of ShapeWriter ([9]).

(3) Make stroke shortcuts visible to end users

A well-known and important drawback of using strokes to activate commands is that these strokes are not self-revealing [13, 6, 1]. In other words, as opposed to buttons and menus, the user cannot guess which stroke-based commands are available and which stroke triggers which command. Often novel features of an interface are unused not because they are difficult to use, but because the users are not aware of them. Therefore ***interfaces should offer visual clues to available strokes*** to make end users able to discover and learn their effect.

(4) Integrate stroke shortcuts in graphical toolkits

Because interface developers are not willing to change their development environment or rewrite their existing applications, ***interface toolkits should support the implementation***

Figure 7. A simple Java Swing interface for a music player.

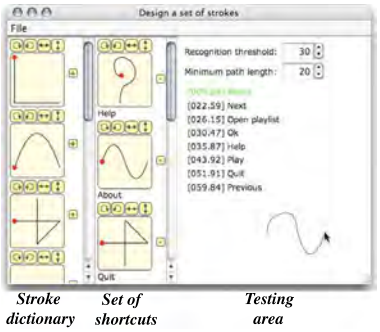


Figure 8. The *Design Shortcuts* application

of stroke shortcuts. Of course, the implementation capabilities should be high-level enough to minimize developer programming effort. As a baseline, developers typically need to only add one line of code per command to implement a keyboard shortcut. Implementing a stroke shortcut should not involve much more programming effort.

SST: stroke shortcuts in Java Swing

In this section, we present SST[5], a Java Swing extension to simplify the addition of stroke shortcuts to any Swing application. To illustrate, lets consider that we want to add stroke shortcuts to the music player window shown on Figure 7 and built with the instruction:

```
SimplePlayer player = new SimplePlayer();
```

To define the mappings between the commands and their shortcut strokes, the developer can invoke the *Design Shortcuts* graphical design environment shown on Figure 8. Launching this environment on the application windows for which he wants to map commands with stroke shortcuts requires the single line:

// Launch Design Shortcuts environment on the main player window and its ***About*** *dialog shown on Figure 10*

```
1  new DesignShortcuts(player, player.about);
```

[5]SST is an open source project containing about 3000 lines of code and is available online: http://code.google.com/p/strokesshortcuts/.

The *Design Shortcuts* interface (Figure 8) is divided into three areas: the *stroke dictionary* (left panel), the *set of shortcuts* (middle panel) and the *testing area* (right panel). To define a new shortcut, the developer clicks on the '+' button displayed on the right of a stroke in the *dictionary*. This pops up the list of commands found in the attached windows. He can either (i) pick one of these commands in the list or (ii) type a *new command* name. Callbacks for these new commands are handled through the use of Java listeners as explained later in this section. At any time, the developer can test the recognition accuracy by drawing in the testing area. As soon as a stroke ends, the application displays the list of the distances between the input stroke and each template, the recognized stroke being the template with the shortest distance.

The *stroke dictionary* contains an initial set of 9 predefined strokes for the developer to choose from. With these predefined strokes, the developer can already define a large set of shortcuts by combining several of these strokes and/or applying geometrical transformations to them. One can use the *transformation buttons* displayed on top of each stroke to rotate or mirror (horizontally or vertically) a stroke before adding it to the set of shortcuts (using the '+' button displayed to the right of the stroke). In the example shown in Figure 8, the developer has used the same shape for **Ok** and **Play**: the orientation of the **Ok** shortcut suggests a check mark while the **Play** shortcut suggests the symbol usually dedicated to the play command in many music players. Selecting several strokes before pressing one of the '+' buttons will build a new stroke that is the concatenation of the selected strokes. For example, the stroke for the **About** command has been defined by concatenating the predefined "arch" stroke with a mirrored copy of it[6]. Once added to the set of shortcuts, the transformation buttons remain displayed for further modifications. If needed, the '-' button displayed to the right of each stroke allows the shortcut to be removed.

Compared with starting with a "blank page", providing a set of primitive strokes and a set of operations on these strokes opens a structured design space that can be systematically explored. However, there is no reason to constrain the developer to this set of primitives. Developers can expand the stroke dictionary with additional custom strokes: a "Free stroke" button at the bottom of the list of primitives opens a separate frame for drawing a stroke to be added to the dictionary. This is how the developer has defined a question mark stroke for the **Help** command in our example (Fig. 8).

Once designed, the set of shortcuts can be saved as a file ("player.strokes" here) and enabled on a given Swing interface through a few lines of code. In our music player example, only 10 lines of code (Figure 9) are needed to accomplish this. SST connects the shortcuts to a Swing GUI using a central object, the *stroke shortcuts manager* (`m`, line 1). This object is in charge of integrating stroke shortcuts to the Java Swing toolkit and is used to register:

- the mappings (stroke to command name) (line 2),

[6]A pop up menu allows to duplicate any stroke in the dictionary.

```
1  StrokeShortcuts m = new StrokeShortcuts(
              player, player.dialogAbout);
2  m.addShortcuts("player.strokes",
              MENU_PREVIEW, TOOLTIP_PREVIEW);

3  m.setCriterion(player.playlist, new Criterion(){
4     public boolean startStroke(MouseEvent event) {
5        return event.getButton() == MouseEvent.BUTTON3;
6     }
7  });
8  m.disableStrokes(player.sliderSong);
9  m.disableStrokes(player.sliderVolume);

10 m.enableStrokesSheet();
```

Figure 9. Complete code to add keyboard shortcuts to the music player interface.

Figure 10. Strokes in different windows.

- the windows that contain commands that can be invoked through these shortcuts (lines 3 and 4) and
- the "strokable" components, i.e. the Swing widgets on which strokes can be drawn (lines 5 to 13, detailed below).

In SST, a stroke is defined as a series of points sent by an analog input device (a mouse or a digital pen) that starts with a press event and ends with a release event. Each stroke occurring on a "strokable" component is entered into the recognizer to get the name of the command that is then invoked through the Java accessibility interface. In our example, line 1 registers both the main frame and the **About** dialog as "strokable" components so the user can draw on any of the two windows as illustrated in Figure 10. By default, all children components of a "strokable" component are also "strokable". Since press, drag and release are events that may already be used by standard widgets, SST allows the developer (i) to associate a criterion on the mouse press event that specifies when the stroke recognition must be enabled or (ii) to disable stroke recognition on specific components. For example, no criterion is required when the user wants to stroke on the interface background while one is required for a list box on which a drag is already an action dedicated to selecting items in the list. In this latter case, the developer can decide to accept only strokes drawn when the right mouse button is pressed (lines 3 to 7). Finally, he disables stroke recognition on the sliders for adjusting the playing point in a song and the volume (lines 8 and 9).

By default, in SST, a stroke leaves a visible ink trail (by means of the transparent overlay available in the window containing the component). At the end of a stroke, its ink trail is either smoothly morphed into the template it matches (in case of recognition, as ShapeWriter does [9]) or flashes red (if it is not recognized). Note that the ink is morphed into a template scaled to the same size as the stroke to minimize visual change. Also, the morphing animation stops as soon as the user starts a new stroke so expert users can enter strokes in rapid succession. The morphing animation (or beautification) not only provides a feedback of recognition result but also helps novice users learn the correct stroke shape and discourages expert users from departing too much from the ideal shape. If the transparent overlay is already used for another purpose, stroke ink can be disabled and a different feedback mechanism can be implemented. One or several stroke listeners can be attached to the shortcut manager which will be notified each time the user begins a stroke, adds a point to a stroke or ends a stroke. When ending a stroke, the event can be of one of the three types: *recognized shortcut*, *recognized stroke* or *non recognized stroke*. In all cases, the current input stroke can be retrieved from the received event so that it can easily be used for other interactions. For example, a non recognized stroke could be used for drawing in a graphical editor.

To address the *visibility* problem (i.e. users do not have a way to discover the available strokes and their meaning), SST offers three types of visual clues to make the user discover and learn the mappings: *Tooltip*, *Menu preview* and a *Strokes Sheet*. The first two types of visual clues are turned on or off by the parameters of `addShortcuts` (line 2 on Figure 9). If *Tooltip* is turned on, any graphical component that provides a shortcut will display it in a tooltip that pops up when the mouse cursor dwells over this component. If this component already has a tooltip associated with it, the existing tooltip is augmented with the stroke illustration while preserving its original text. Similarly, if the *Menu preview* is turned on, any menu item that is invokable by a stroke displays a preview of this stroke beside its label as is usually done with keyboard shortcuts. Finally, a *Strokes Sheet* can be enabled in the stroke shortcuts manager (line 10, Figure 9). A strokes sheet is an independent window that displays the list of shortcuts and the name of their associated commands found in the current opened windows. The behavior of the strokes sheet has been inspired by the Tivoli system in [13]: this sheet pops up each time the user pauses during a stroke (at the beginning or at any moment while stroking) and remains visible until the user enters a shortcut that is successfully recognized or closes the sheet.

In this section, we showed how SST allows a developer to add stroke shortcuts to a Java Swing interface in only 10 lines of code without having to modify the basic code for the music player. The 3000 lines of code in SST offload the developer not only from having to implement or train any recognizer but also from developing visual displays (morphing feedback, tool tips or crib sheet). We have also shown how the *Design Shortcuts* environment helps developers to map a set of strokes by offering a structured design space.

Recognition accuracy in SST

Since the recognizer implemented in SST skips the rotation step of the $1 recognizer evaluated in [22], one may wonder to what extent it affects recognition accuracy. Although we

have not observed a noticeable degradation during our informal tests, we decided to conduct a controlled experiment that measures the recognition accuracy of our simple matching algorithm on the set of strokes shown in Figure 11. We chose to use a set of 16 strokes since the answers to the post hoc questionnaire of Experiment 1 revealed that our participants use roughly 15 different keyboard shortcuts in their everyday use of computers.

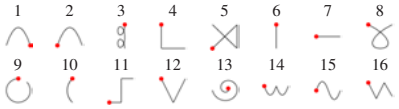


Figure 11. The 16 strokes used in our experiment.

This experiment involved 6 of the participants that had already served in our first experiment and the same apparatus. The task was very simple: one stroke was displayed on the screen and the participant was told to reproduce it as fast as possible and as accurately as possible. As soon as the participant started to draw, the sample stroke disappeared to avoid turning the reproduction task into a copy task. At any time (except during stroking), the participant could have the stroke displayed again by pressing the space bar and start the task again. We implemented this possibility to avoid the situation where the recognition failed and the participant had forgotten what stroke to produce. A trial ended only if the right stroke was recognized.

The experiment had two $Input\ device$ conditions: Pen and $Mouse$. We included a regular mouse for two reasons. First, since the mouse is less dexterous than a pen in articulating shapes (drawing one's signature with a mouse vs. a pen shows the difference), the mouse condition would add more stress to the recognizer. Second, it is also practically useful to know if stroke shortcuts can be used with a mouse. In the experiment the participants had to perform 11 blocks in each condition that were grouped to avoid successive changes of input device. Each block consisted of 16 trials, one per $Stroke$, presented in a random order. The presentation order of input devices was counterbalanced between participants so 3 participants started in the $Mouse$ condition while the 3 others started in the Pen condition. In each condition, the first block was a practice block.

We measured the $Stroking\ time$, i.e. the time between the press and the release event when drawing the right stroke, and the number of $Recognition\ errors$. Analysis of variance revealed a significant effect of both $Input\ device$ ($F_{1,5} = 34$, $p < 0.002$) and $Stroke$ ($F_{15,75} = 25$, $p < 0.0001$) on $Stroking\ time$. Users were faster with a pen (394 ms on average) than with a mouse (704 ms on average). Also, the $Stroking\ Time$ increased with the complexity of the stroke, supporting the results reported in [2]. More surprisingly, we observed a significant interaction effect of $Stroke \times Input\ device$ on $Stroking\ time$ ($F_{15,75} = 5$, $p < 0.0001$): differences between input devices seem to increase with the complexity of the stroke (particular more curves). All these results are illustrated on Figure 12.

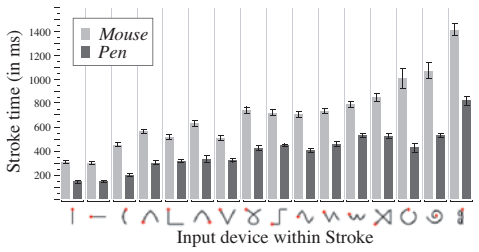


Figure 12. Stroke time by $Stroke \times Input\ Device$

Analysis of variance also revealed a significant effect of $Input\ device$ ($F_{1,5} = 9$, $p < 0.03$) on $Recognition\ errors$. In the $Mouse$ condition, the participants made 7.4% errors on the first attempt at each stroke sample presented. Among these, they succeeded 73% of the time with the second attempt, 10% with the third attempt, and 17% with the subsequent attempts. In the Pen condition, only 3% of the trials failed with the first attempt, of which 76% were corrected with the second attempt, 7% with the third attempt, and 17% with the subsequent attempts. There was also a significant main effect of $Stroke$ ($F_{15,75} = 3$, $p < 0.001$) on $Recognition\ errors$: the error rates drastically changed when removing the 3 more complex strokes from our data: less than 0.001% of the trials in the $Mouse$ condition and 0% of the trials in the Pen condition on the first attempt. Finally, for $Recognition\ errors$, the interaction effect $Stroke \times Input\ device$ was not significant. Overall this study shows that the recognizer used in the *StrokeShortcuts* library is accurate. Although users' stroke articulation speed was considerably slower with a mouse than with a pen, the shape-matching based recognition algorithm could accurately recognize mouse strokes as well.

CONCLUSION

Menu selection has been, and will likely continue to be, the basic and dominant way of activating commands in human-computer interaction. Ubiquitous in modern software applications, keyboard shortcuts provide a faster alternative to frequently used commands. The investigation presented in this paper encourages the use of stroke gesture as shortcuts for touch screen-based devices without a physical keyboard. The conceptual and empirical study in the first part of the article shows that stroke shortcuts can be as effective as keyboard shortcuts in eventual performance, but have cognitive advantages in learning and recall. With the same amount of practice, about three times as many stroke shortcuts were learned as keyboard shortcuts. Following a set of development guidelines articulated in the second half of the paper, we have shown a simple way to implement stroke shortcuts in Java Swing by providing developers with SST. Requiring no training by the developer or the end user, the built-in shape matching-based recognizer in SST can yield high accuracy for simple strokes, even if the strokes are articulated with a mouse. SST offers a structured yet open design environment and simplifies the implementation of strokes' visibility in applications.

Integrating stroke shortcuts in the Java / Swing platform-independent framework is a first step, we now plan to develop extensions to other frameworks like Objective C / Cocoa or C# to cover most of the applications developed for touch screen devices ranging from Apple's iPhone to HP's TouchSmart desktop PCs.

ACKNOWLEDGEMENTS

We thank Michel Beaudouin-Lafon and Alison Sue for helping to improve this article, our participants for having served in our studies and Pierre Dragicevic for sharing with us the set of icons used in [4].

REFERENCES

1. O. Bau and W. E. Mackay. Octopocus: a dynamic guide for learning gesture-based command sets. In *UIST '08: Proceedings of the 21st annual ACM symposium on User interface software and technology*, 37–46, New York, NY, USA, 2008. ACM.
2. X. Cao and S. Zhai. Modeling human performance of pen stroke gestures. In *CHI '07: Proc. ACM Conference on Human factors in computing systems*, 1495–1504, 2007.
3. F. Craik and R. Lockhart. Levels of processing: A framework for memory research. *Journal of Verbal Learning and Verbal Behavior*, 11(6):671–684, 1972.
4. T. Grossman, P. Dragicevic, and R. Balakrishnan. Strategies for accelerating on-line learning of hotkeys. In *CHI '07: Proc. ACM Conference on Human factors in computing systems*, 1591–1600, 2007.
5. F. Guimbretière, A. Martin, and T. Winograd. Benefits of merging command selection and direct manipulation. *ACM Trans. Comput.-Hum. Interact.*, 12(3):460–476, 2005.
6. K. Hinckley, S. Zhao, R. Sarin, P. Baudisch, E. Cutrell, M. Shilman, and D. Tan. Inkseine: In situ search for active note taking. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, 251–260, New York, NY, USA, 2007. ACM.
7. J. Hong and J. Landay. Satin: a toolkit for informal ink-based applications. In *UIST '00: Proc. ACM Symposium on User interface software and technology*, 63–72, 2000.
8. P. Kabbash, W. Buxton, and A. Sellen. Two-handed input in a compound task. In *CHI '94: Proc. ACM Conference on Human factors in computing systems*, 417–423, 1994.
9. P.-O. Kristensson and S. Zhai. Shark2: a large vocabulary shorthand writing system for pen-based computers. In *UIST '04: Proc. ACM symposium on User interface software and technology*, 43–52, 2004.
10. P.-O. Kristensson and S. Zhai. Command strokes with and without preview: using pen gestures on keyboard for command selection. In *CHI '07: Proc. ACM Conference on Human factors in computing systems*, 1137–1146, 2007.
11. G. Kurtenbach and W. Buxton. User learning and performance with marking menus. In *CHI '94: Proc. ACM Conference on Human factors in computing systems*, 258–264, 1994.
12. G. Kurtenbach, G. Fitzmaurice, R. Owen, and T. Baudel. The hotbox: efficient access to a large number of menu-items. In *CHI '99: Proc. ACM Conference on Human factors in computing systems*, 231–237, 1999.
13. G. Kurtenbach and T. Moran. Contextual animation of gestural commands. *Eurographics Computer Graphics Forum*, 13(5):305–314, 1994.
14. D. Lane, H. Napier, S. Peres, and A. Sandor. Hidden Costs of Graphical User Interfaces: Failure to Make the Transition from Menus and Icon Toolbars to Keyboard Shortcuts. *International Journal of Human-Computer Interaction*, 18(1):133–144, 2005.
15. A. C. J. Long, J. A. Landay, and L. A. Rowe. Implications for a gesture design tool. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, 40–47, New York, NY, USA, 1999. ACM.
16. A. C. J. Long, J. A. Landay, L. A. Rowe, and J. Michiels. Visual similarity of pen gestures. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, 360–367, New York, NY, USA, 2000. ACM.
17. P. Morrel-Samuels. Clarifying the distinction between lexical and gestural commands. *Int. J. Man-Mach. Stud.*, 32(5):581–590, 1990.
18. D. Odell, R. Davis, A. Smith, and P. Wright. Toolglasses, marking menus, and hotkeys: a comparison of one and two-handed command selection techniques. In *GI '04: Proc. of Graphics Interface*, 17–24, 2004.
19. Y. Rogers. Evaluating the meaningfulness of icon sets to represent command operations. In *Proc. Conference of the British Computer Society*, 586–603, 1986.
20. D. Rubine. Specifying gestures by example. *SIGGRAPH Comput. Graph.*, 25(4):329–337, 1991.
21. C. Tappert. Cursive script recognition by elastic matching. *IBM Journal of Research Development*, 26(6):765–771, 1982.
22. J. Wobbrock, A. Wilson, and Y. Li. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *UIST '07: Proc. ACM Symposium on User interface software and technology*, 159–168, 2007.
23. C. Wolf. Can people use gesture commands? *ACM SIGCHI Bulletin*, 18(2):73–74, 1986.
24. C. Wolf, J. Rhyne, and H. Ellozy. The paper-like interface. In *Proc. International conference on human-computer interaction on Designing and using human-computer interfaces and knowledge based systems*, 494–501, 1989.
25. R. Zeleznik and T. Miller. Fluid inking: augmenting the medium of free-form inking with gestures. In *GI '06: Proc. of Graphics Interface*, 155–162, 2006.
26. S. Zhai and P.-O. Kristensson. Shorthand writing on stylus keyboard. In *CHI '03: Proc. ACM Conference on Human factors in computing systems*, 97–104, 2003.
27. S. Zhao and R. Balakrishnan. Simple vs. compound mark hierarchical marking menus. In *UIST '04: Proc. ACM Symposium on User interface software and technology*, 33–42, 2004.

Prospective Motor Control on Tabletops: Planning Grasp for Multitouch Interaction

Halla Olafsdottir[1,2,3]
halla@lri.fr
[1]Univ Paris-Sud (LRI)
F-91405 Orsay, France

Theophanis Tsandilas[3,1,2]
Theophanis.Tsandilas@inria.fr
[2]CNRS (LRI)
F-91405 Orsay, France

Caroline Appert[1,2,3]
appert@lri.fr
[3]Inria
F-91405 Orsay, France

ABSTRACT
Substantial amount of research in Psychology has studied how people manipulate objects in the physical world. This work has unveiled that people show strong signs of prospective motor planning, i.e., they choose initial grasps that avoid uncomfortable end postures and facilitate object manipulation. Interactive tabletops allow their users great flexibility in the manipulation of virtual objects but to our knowledge previous work has never examined whether prospective motor control takes place in this context. To test this, we ran three experiments. We systematically studied how users adapt their grasp when asked to translate and rotate virtual objects on a multitouch tabletop. Our results demonstrate that target position and orientation significantly affect the orientation of finger placement on the object. We analyze our results in the light of the most recent model of planning for manipulating physical objects and identify their implications for the design of tabletop interfaces.

Author Keywords
Movement planning; acquisition and manipulation; range of motion; end-state comfort effect; multitouch; tabletops

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

INTRODUCTION

The manipulation of virtual objects has a central role in interaction with tabletops. For example, users move and rotate documents and pictures around the surface to share them with other users. Graphical designers manipulate information and graphical objects to create new content. Multiple users work collaboratively to create schedules, make decisions, or solve complex problems. In all these scenarios, users interact with their hands and their fingers; they grasp, translate, and rotate virtual documents as they would do with physical objects.

Literature in experimental Psychology contains a large body of work that studies the manipulation of physical objects. In particular, several experiments have shown that the initial grasp when acquiring an object is influenced by the subsequent planned actions so as to optimize *end-state comfort* [20]. Research in Human-Computer Interaction has never validated or tested these results, which suggest that we could possibly anticipate people's intentions as soon as they grab an object and before its actual manipulation starts.

Given that multitouch interaction techniques [5, 12, 28] usually simulate object manipulation in the physical world, we hypothesize that movement planning also takes place when users directly manipulate virtual objects with their hands. If this hypothesis is supported, we could possibly infer information about users' prospective movement to improve user experience during the manipulation phase. Interface designers could, for example, develop techniques that adapt their graphical layout to improve visual feedback, avoid potential occlusion issues [4, 25] or reduce interference [10] when multiple users interact in close proximity in collaborative settings. We could also derive directions about how to design grips and visual guides to facilitate both the acquisition and the manipulation of virtual objects.

We test this planning hypothesis by observing how people grasp objects prior to moving them to specific positions and orientations on a horizontal screen. We present three experimental studies that examine a simple two-dimensional docking task on the surface of a multitouch tabletop. The first experiment tests translation-only tasks. The second experiment tests rotation-only tasks. Finally, the third experiment examines tasks that combine both translational and rotational movements. The results of all the three experiments confirm the planning hypothesis. They show that the placement of the fingers at acquisition time is influenced by both the initial and the final state (position and orientation) of the virtual object. They also provide valuable information about how users grasp objects at different positions of a multitouch tabletop.

We analyze our results in the light of the *Weighted Integration of Multiple Biases* model [6], a very recent model in Psychology research. The model helps us to explain how the orientation of a user's initial grasp is influenced by a combination of several factors or *biases*, where each bias pulls the grasp orientation towards a certain orientation. We examine how our experimental results conform to this model. Finally, we discuss the design implications of our findings and identify several future directions. Our work focuses on multitouch tabletops but could serve as a framework for studying object manipulation in a larger range of user interfaces, including multitouch mobile devices and tangible interfaces.

CHI 2014, April 26 - May 01 2014, Toronto, ON, Canada.
Copyright 2014 ACM 978-1-4503-2473-1/14/04...$15.00.
http://dx.doi.org/10.1145/2556288.2557029.

RELATED WORK

Manipulating Objects on Multitouch Surfaces

Previous work has studied a range of multitouch gestures for manipulating objects on interactive surfaces. Wu and Balakrishnan [28] defined a set of gestures that make use of both hands and multiple fingers. Among others, they demonstrated how to perform freeform rotations using the thumb and index finger. Moscovich and Hughes [17] proposed multi-finger interactions that allow users to control a larger number of degrees of freedom to translate, rotate, and deform an object in a single manipulation. Kruger et al. [12], on the other hand, proposed single-touch rotation and translation mechanisms relying on physics-based metaphors for manipulating objects. Hancock et al. have discussed advantages and disadvantages of different rotation and translation techniques [5]. Studies reported in [16, 27] have proposed sets of gestures defined by end-user elicitation methods and concluded that people prefer conceptually and physically simpler gestures than the ones created by HCI researchers. Finally, Hinrichs and Carpendale [8] examined how adults and children naturally interact with tabletops and observed significant variations among gestures of different users.

Discussing the properties of graspable user interfaces, Fitzmaurice and Buxton [3] identify two main phases of interaction: acquisition and manipulation. Although these two phases can be studied separately [3, 24], previous results [9] indicate that manipulation performance may depend on proper acquisition. Multitouch gestures are subject to the physical constraints imposed by the user's arm, wrist and finger joints. As a result, they can result in joint stress and discomfort. Hoggan et al. [9] studied the extent and comfort of $90°$ rotational movements at different locations on a horizontal surface starting from different angular postures. Lozano et al. [13] measured muscle activation using electromyography and observed that gestures involving two fingers can result in high levels of muscle activation. They concluded that "multitouch interaction has impact on the entire hand shoulder system and in some cases the impact can be at risk level".

Planning when Manipulating Physical Objects

How people plan their acquisition and grasp to facilitate movement and optimize comfort has been the focus of a large body of work within the fields of Psychology and Motor Control. This work can be expressed using the notion of *orders of planning* [18]. Within that system, the last task in a sequence that influences the behavior defines the planning order. First order planning occurs when a grasp is influenced by the immediate task, for example the objects shape. Second order, when the grasp is influenced by the subsequent task, e.g., grasping an object to rotate or translate it to a given position, and so on. Research studying first order planning of grasp have revealed that the kinematics of the hand depend, for example, on the size, orientation, and shape [11, 22] of the object of interest.

Several studies have considered second or higher order planning. Marteniuk et al. [14] showed that the kinematics, i.e., the shape of a grasp, is influenced by the intended use of the object. Rosenbaum and colleagues have extensively studied how people orient their hand when grasping an object. They revealed that individuals favor initial hand placements that result in end positions that are either comfortable, i.e, optimize *end-state comfort* [19, 21], or yield the most control [20]. Short and Cauraught have corroborated these results [23]. This type of planning behavior is termed *prospective* movement control [1].

The above studies have mainly focused on discrete tasks where participants had to choose one of two grasps (e.g., grabbing a cylinder with the thumb up or down). Choosing one grasp yields an uncomfortable end position while the other one a comfortable, hence optimizes *end-state comfort*. However manipulating physical or user interface objects usually involves more continuous tasks.

Other studies, reviewed by Herbort [6], have examined continuous tasks, in particular rotations of physical knobs for a range of angles. Their results suggest that *end-state comfort* planning alone cannot sufficiently explain the observed grasp selection in such tasks. Herbort [6] argues that it is unclear how precisely someone can anticipate a final posture of a movement and its associated costs, and therefore, optimal planning may not always be feasible. To account for the various *biases* that determine a grasp selection, he proposes the *Weighted Integration of Multiple Biases* (WIMB) model [7]. In its simplest form, the model can be expressed as follows:

$$p_{initial} = \frac{w_{anti} \cdot p_{anti} + w_{default} \cdot p_{default}}{w_{anti} + w_{default}} \quad (1)$$

According to the model, two different biases contribute to the initial grasp orientation $p_{initial}$. An anticipatory bias pulls the initial grasp toward a pronated or supinated angular position p_{anti}, depending on the intended direction of rotation. A second bias pulls the initial grasp toward a preferred task-independent orientation $p_{default}$. The contributing weights w_{anti} and $w_{default}$ of the two biases can vary, for example, depending on the difficulty of the task or the required end precision. The above model can be extended with additional bias terms, such as one that accounts for the effect of previous movements in a sequence of tasks that involve different rotation directions and angles [6].

To the best of our knowledge, HCI research has never validated or tested the above results. The most relevant contribution in this direction belongs to Möllers et al. [15]. They tested the hypotheses of a predecessor and a successor (i.e., a planning) effect on the offset and angle of a touch point in a sequence of pointing tasks. They observed that finger posture is influenced by the previous pointing action but not by the next pointing action. This suggests that prospective control does not occur in this specific pointing case.

GOALS AND APPROACH

Our hypothesis is that movement planning plays a determinant role in tabletop interaction as movements extend to a large space and object manipulation involves the coordination of multiple limbs, often in constrained positions and postures. Our goal was to test this hypothesis but also understand and

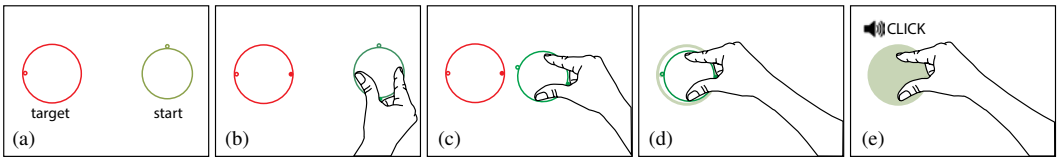


Figure 1. Experimental task scenario: The task requires the user to (b) grab the green object with the thumb and the index finger, (c) move it towards the red target and then align it with it, and (d) hold the object in the target for 600 ms to (e) complete the task.

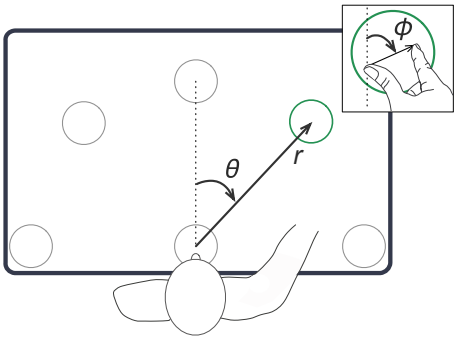


Figure 2. The position of interactive objects expressed in polar coordinates (r, θ), where r is the radial distance and θ is the clockwise angle with respect to the vertical axis of the screen. The grip orientation is expressed by the clockwise angle ϕ defined by the thumb and the index.

describe how planning affects how users grasp virtual objects to facilitate their manipulation.

We conducted three experiments. The experiments tested unconstrained translation and rotation tasks on different locations of a multitouch surface. As opposed to Hoggan et al. [9] who express screen location in x and y coordinates, we use a polar coordinate system and express the location of an object in terms of its distance r and angle θ with respect to the front-center of the screen, close to where the user stands (see Figure 2). This design configuration was driven from the observation that the orientation of a neutral hand posture changes in circular manner around the user. Although the polar coordinate system presented in Figure 2 is not an accurate representation of the user's biomechanical coordinate system, it offers a reasonable approximation and simplifies data analysis. As we see later in this paper, our approach allows for better experimental control and a simpler interpretation of the observed grasp orientations. As Hoggan et al. [9], we focus on one-hand two-finger interaction, where objects are grasped and rotated with the thumb and the index finger (Figures 1-2).

Our studies are mostly inspired by the continuous-tasks approach [6, 7] rather than discrete-tasks approach of Rosenbaum et al. [18, 19, 20]. The former is more generic and can describe situations with uncertainty about the final grasp orientation of a movement and the costs associated with a certain object acquisition strategy. In such cases, optimal planning is difficult or even impossible. We analyze our data in the light of the WIMB model [6], which predicts the initial grasp orientation given the anticipatory target-orientation bias and the default task-independent orientation bias (see Equation 1). The WIMB model was based on results from pure rotation tasks with tangible objects. Here, we examine translation in addition to rotation.

EXPERIMENTAL METHOD

The task of all the three experiments consists of grasping and moving an object. Each experiment, however, focuses on a different movement component. In Experiment 1, we test a translation task where participants have to change the position of the object while keeping its initial orientation. Experiment 2 involves rotations, requiring participants to change the objects' orientation but not their position. In Experiment 3, we combine translations and rotations so participants need to both change the position and orientation of the object.

Apparatus

The experiments were performed on a 3M Multi-Touch Display C3266P6 with 698.4×392.85 mm display area, a refresh rate of 120 Hz, and a native resolution of 1920×1080. The display was placed flat on a table in landscape orientation, resulting in the multitouch surface to be at a height of 95 cm. A digital video camera on a tripod above the display monitored the participant's hand and arm movements.

The experimental software was developed in Java 2D (JDK 6) and ran on a Macbook Pro 2.66 GHz Intel Core i7 with 4GB memory, running Mac OS X 10.6.8. Touch noise was reduced with a complementary filter.

Common Task Features

Figure 1 illustrates a typical scenario for our experimental tasks. In all three experiments the touch display shows a circular *start* object, which can be moved and rotated, and a static circular *target*. The *start* object is green and has a diameter of 60 mm. The *target* object is red and has a diameter of 70 mm. To start a trial the user presses a touch button at the bottom half of the display. The user has then to grab the *start* object with the thumb and the index of the right hand and manipulate it to make its position and orientation match the *target*. The user can freely translate and rotate but not resize the object. Translations follow displacements of the center of the segment connecting the touch points of the two fingers. Rotations follow changes in the angular position of this segment. The orientation of both object and target are indicated by a handle (small open circles). Grasping the *start* object triggers the appearance of a secondary handle (small

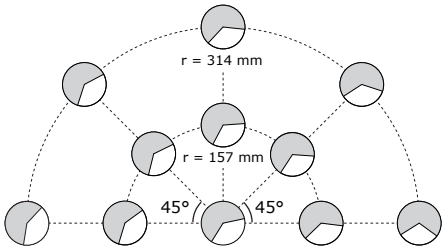


Figure 3. Extreme range of grip angles for various tabletop positions. Average ranges for the right hand of 10 right-handed participants.

closed circles) located $180°$ from the primary (Figure 1-b). To complete a task the *start* object has to be held in the *target* for 600 ms. The precision tolerance for placing the object into the target is $\pm 5°$ in angular direction and 5 mm in diameter. The angular positions (θ_{start} and θ_{target}) of objects, their radial distances (r_{start} and r_{target}) and their rotation angles β are specific to each experiment and will be detailed later.

The user interface provides visual feedback to indicate that the object was correctly placed into the target. It also provides visual and audio feedback to inform the user about the completion of the task and errors, which occur when the user lifts a finger before task completion.

Procedure

Prior to each experiment, participants had to wash their hands and dry carefully in order to minimize screen friction and facilitate object sliding. Participants were positioned standing at the center of the long side of the display and were not allowed to walk. The operator asked them to only use the thumb and the index finger of the right hand to interact with the object, while keeping their left hand down by their side. Participants were not explicitly encouraged to plan their grasps and were not aware of the experimental goals. They were instructed not to rush and avoid errors.

Measures

We recorded detailed information about the position of the fingers on the multitouch screen and their movements. Our two main dependent variables are:

1. The initial grasp orientation $\phi_{init} \in [-180°, 180°]$, measured as the clockwise angle between the vertical axis and the vector from the thumb to the index finger (see Figure 2). Our 3M multitouch display could not differentiate between fingers. We derived the correct grasp orientation from the range of attainable grasp orientations, measured at each screen position in a pre-study with 10 participants (see Figure 3). We also used detailed logs and recorded video to ensure that grasp orientation was derived correctly.

2. The default task-independent grasp orientation $\phi_{default} \in [-180°, 180°]$ for each position of the display. To measure it, we only consider trials where *start* and *target* configurations are the same.

We also measure $ErrorRate$ and the reaction time RT participants need to plan their grasp before touching the screen.

Hypotheses

We hypothesize that ϕ_{init} is determined by both the *start* and *target* object configurations. We expect that planning will occur for both rotational and translational movements. Since the orientation of ergonomic hand gestures changes along different locations of the tabletop [9], we predict that users will plan appropriately in order to reduce the occurrence of uncomfortable end-postures. Influenced by the WIMB model [6], we hypothesize that ϕ_{init} will be affected by three factors of postural bias: $\phi_{default}$ of the *start* position, $\phi_{default}$ of the *target* position, and the object's angle of rotation β.

EXPERIMENT 1: TRANSLATIONS

We first tested translation tasks, where participants had to grab and move an object, keeping its initial orientation.

Participants

Twelve volunteers (four women and eight men), 23 to 32 years old, participated in the experiment. All were right-handed and had normal or corrected-to-normal vision.

Task and Conditions

We tested six screen positions for both the *start* and the *target* objects. One was located close to the user, centered on the vertical axis of the display, 35 mm from the front edge. We refer to it as the *User* position. The other five positions were located around the *User* position with an angular position θ_{start} of $-90°$, $-45°$, $0°$, $45°$, and $90°$, and a radial distance of $r = 314$ mm. The *start* and the *target* objects could appear at the same position. In this case, the user should hold the *start* object and keep it inside the *target*.

To test whether and to what extent planning occurs for translation tasks, there were two main conditions:

Known Target. The *target* appears with the *start* object at the beginning of the task. Users are aware of the end position of their movements, and therefore, they can plan the orientation of their grasps.

Hidden Target. This is a control condition. The *target* is initially hidden. It appears after the user acquires the *start* object. Thus, users cannot plan the orientation of their grasp.

Design

We followed a within-participants full-factorial design, which can be summarized as follows:

12 participants
$\times$ 2 Conditions (*Known*, *Hidden Target*)
$\times$ 2 Blocks
$\times$ 6 θ_{start} (*User*, $-90°$, $-45°$, $0°$, $45°$, $90°$)
$\times$ 6 θ_{target} (*User*, $-90°$, $-45°$, $0°$, $45°$, $90°$)
$\times$ 3 Replications

$= 5184$ tasks in total

In addition, participants completed 15 practice tasks for each condition. The order of presentation of the two conditions

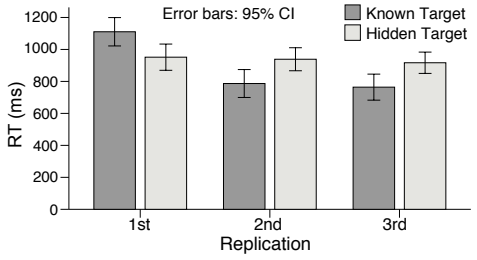


Figure 4. Mean reaction time for each of the three task replications

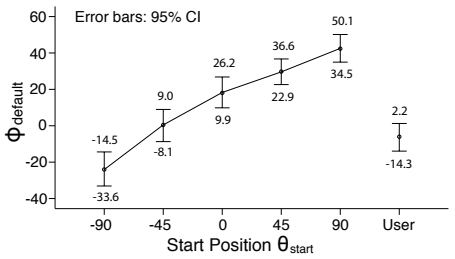


Figure 5. The default grasp orientation $\phi_{default}$ at each position θ_{start}

was fully balanced among participants. Start and target positions were randomized within each block. Tasks were grouped by three but in a different way for each condition. In the *Known Target* condition, groups contained the three replications of the same task, allowing participants to re-plan and possibly revise their grasp orientation. In case of an error, the participant had to restart the task. In the *Hidden Target* condition, groups contained a random selection of tasks. When an error occurred, the task was not repeated immediately. It was moved to the end of the block and was replaced by the following in the list. This design eliminates planning effects for this condition. Experimental sessions lasted 50 to 60 minutes.

Results

For error comparisons, we used the Wilcoxon signed-rank test. For *RT*, we conducted a 5-way Repeated Measures (RM) ANOVA with the complete set of factors. For $\phi_{default}$, we conducted a 3-way ANOVA, where we included only *Known Target* tasks for which θ_{target} was identical to θ_{start}. Finally, for ϕ_{init}, we split our data into three sets:

1. PERIPHERY: The *start* and *target* objects are at the periphery of the display.
2. OUTWARD: The *start* object is close to the user.
3. INWARD: The *target* object is close to the user.

We conducted a 5-way RM ANOVA for the first set and 4-way RM ANOVAs for the second and third set, as the factors θ_{start} and θ_{target}, respectively, were not relevant for these sets. We only report on main effects and two-factor interactions that are meaningful and relevant to our hypotheses. When possible, we use a 95% confidence interval (CI) [2] to report on the estimated difference between two means.

Errors

$ErrorRate$ was 3.6% ($SD = 1.9\%$) and 5.3% ($SD = 3.5\%$) for *Known* and *Hidden* targets, respectively. Yet, this difference was not statistically significant ($Z = -1.37$, $p = .17$). Interestingly, leftward movements, starting from the right half (45°, 90°) and ending to the left half (−45°, −90°) of the display resulted in more errors than rightward movements ($Z = -3.06$, $p = .002$). Their error rate was 15.3% ($SD = 7.8\%$) compared to a 3.0% ($SD = 4.0\%$) of the exact opposite movements. We believe that there are two causes of this difference. First, fingers of the right hand may produce more friction when moved leftwards. Second, the right arm is more constrained by the user's body when moving leftwards. Similarly, we found that outward movements starting close to the user produced more errors than the reverse inward movements ($Z = -2.32$, $p = .021$), where $ErrorRate$ was 6.4% ($SD = 3.8\%$) and 3.5% ($SD = 3.1\%$), respectively. We believe that increased finger friction and movement constraints due to the anatomy can also explain this difference.

Reaction Time

RT was not significantly different between *Known* and *Hidden* targets (CI: [−112 ms, 15 ms], $p = .12$). However, we found a significant interaction *Condition* × *Replication* ($F_{2,22} = 76.78$, $p < .001$). Figure 4 presents the estimated mean values. The results suggest that planning only occurred for the first instance of each series of replicated tasks.

Grasp Orientation

θ_{start} had a significant effect on the default grasp orientation $\phi_{default}$ ($F_{2.2,23.9} = 84.79$, $p < .001$)[1]. Figure 5 presents how $\phi_{default}$ varied along different angular positions.

PERIPHERY. ϕ_{init} was not significantly different between *Known* and *Hidden* targets (CI: [−9.9°, 2.5°], $p = .22$). However, the interaction *Condition* × θ_{target} was significant ($F_{4,44} = 13.25$, $p < .001$), which indicates a planning effect. We found a significant main effect of both θ_{start} ($F_{1.2,12.8} = 71.24$, $p < .001$) and θ_{target} ($F_{1.9,21.0} = 12.83$, $p < .001$). Surprisingly, *Replication* did not significantly affect the grasp ($F_{2,22} = 1.51$, $p = .242$). This seconds our results on response time for *Known Target*: participants planned their grasp for the first task in the group but did not refine it after. As shown in Figure 6, ϕ_{init} was mainly determined by the *start* position. The *target* position contributed less, mainly for target positions at the left half of the display.

INWARD. Again, ϕ_{init} was not significantly different between *Known* and *Hidden* targets (CI: [−5.2°, 2.1°], $p = .36$). The effect of θ_{start} was significant ($F_{1.6,17.2} = 37.90$, $p < .001$). However, *Condition* × θ_{start} was only marginally significant ($F_{12.2,24.5} = 3.15$, $p = .056$). As shown in Figure 6, planning only occurred as a slight bias towards lower grasp angles for *start* positions at the right half of the display.

[1]When sphericity is violated, the degrees of freedom have been corrected by using Greenhouse-Geisser correction.

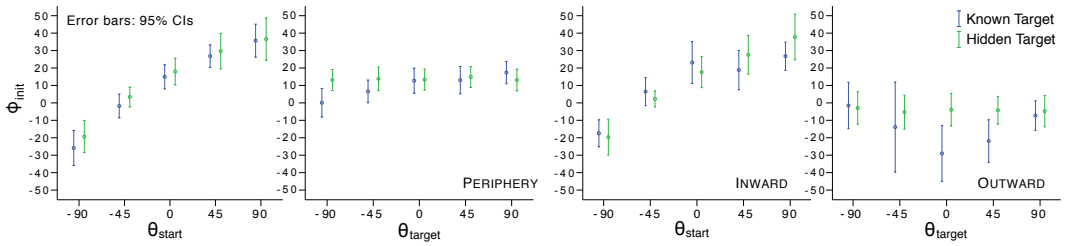


Figure 6. Experiment 1: Effect of start θ_{start} and target θ_{target} angular positions on ϕ_{init} for peripheral, inward and outward translations

OUTWARD. ϕ_{init} was significantly lower for *Known Target* (CI: $[-18.1°, -2.9°]$, $p = .011$). The effect of θ_{target} was significant ($F_{2.0,22.0} = 4.66$, $p = .02$), as was the interaction *Condition* $\times$ θ_{target} ($F_{1.7,18.7} = 4.34$, $p = .033$). Figure 6 shows that planning occurred, but not as expected. The orientation bias added by the *target* positions θ_{target}=0° and θ_{target}=45° has a direction opposite to the one suggested by their default orientations (see Figure 5). This means that participants chose a grasp away from both the start and end-state comfort position. For the $-45°$ target position, results are more unclear because different participants chose different strategies. Our interpretation is that comfort is not always determined by the start and end state of the movement. As the arm and hand have multiple segments and joints that need to coordinate in order to accomplish a movement, transitions between intermediate states can play an important role. In this particular case, we observed that participants adapted their grasp to optimize the flow of their movement.

EXPERIMENT 2: ROTATIONS

The second experiment tested pure rotation tasks that are closer to the physical rotation tasks reviewed by Herbort [6].

Participants

Twelve volunteers (four women and eight men), 22 to 46 years old, participated in the experiments. Three had also participated in Experiment 1. All were right-handed and had normal or corrected-to-normal vision.

Task

Participants performed rotations in two directions $\beta_{dir} \in \{clockwise, counterclockwise\}$. Rotations β had three levels: 40°, 80°, 120°. As the task did not involve translations, the *start* and *target* positions overlapped. We tested the same angular positions θ as in Experiment 1 but added a closer radial distance $r = 157$ mm. We discarded the $User$ position, as rotational movements are uncomfortable when the hand is too close to the body.

Contrary to Experiment 1, the *target* object was always displayed. Our pilot tests showed that completing the most difficult tasks ($\beta \geq 80°$) with no previous knowledge of the *target* was hard or impossible.

Design

We followed a within-participants full-factorial design, which can be summarized as follows:

12 participants
$\times$ 3 Blocks
$\times$ 5 θ ($-90°$, $-45°$, 0°, 45°, 90°)
$\times$ 2 r (157 mm, 314 mm)
$\times$ 2 β_{dir} ($clockwise$, $counterclockwise$)
$\times$ 3 β (40°, 80°, 120°)

$=$ 2160 tasks in total

Prior to the experiment, participants completed 15 practice tasks. The order of tasks within each block was randomized. The experiment took approximately 20 minutes to complete.

Results

For error comparisons, we used the Wilcoxon signed-rank (2 related samples) or the Friedman test (k related samples). For *RT* and ϕ_{init}, we conducted full 5-way RM ANOVAs.

Errors

The angle of rotation β had a significant effect on errors ($\chi^2(2) = 11.35$, $p = .003$). *ErrorRate* was 3.5% ($SD = 3.7\%$), 2.6% ($SD = 2.4\%$), and 6.4% ($SD = 3.3\%$) for 40°, 80°, and 120°, respectively. Differences were significant between 40° and 120° ($p = .024$) and between 80° and 120° ($p = .013$). *ErrorRate* was 4.6% ($SD = 3.0\%$) for clockwise and 3.6% ($SD = 3.2\%$) for counterclockwise rotations, but this difference was not significant ($Z = -1.03$, $p = .31$).

Reaction Time

The mean *RT* was 1057 ms. It was significantly longer (CI: [6 ms, 157 ms], $p = .038$) for clockwise than for counterclockwise rotations. Larger angles also took longer to plan ($F_{2,22} = 20.80$, $p < .001$). More specifically, 120° rotations took 208 ms (CI: [120 ms, 269 ms]) more than 40° rotations ($p = .001$) and 151 ms (CI: [75 ms, 226 ms]) more than 80° rotations ($p = .003$). *Block* did not have any significant effect on *RT* ($F_{2,22} = .51$, $p = .61$), i.e., no learning occurred.

Grasp Orientation

ϕ_{init} was significantly higher for counterclockwise rotations (CI: $[21.1°, 27.5°]$, $p < .001$). The effects of β ($F_{1.4,15.2} = 20.76$, $p < .001$) and the interaction $\beta_{dir} \times \beta$ ($F_{1.1,12.5} = 50.76$, $p < .001$) were also significant. As shown in Figure 7, the effect of clockwise rotations was more pronounced. This result is not surprising. It can be explained by the fact that the right range of grasp orientations, which is used for the planning of counterclockwise rotations, is more constrained compared to the the left range of orientations (see Figure 3).

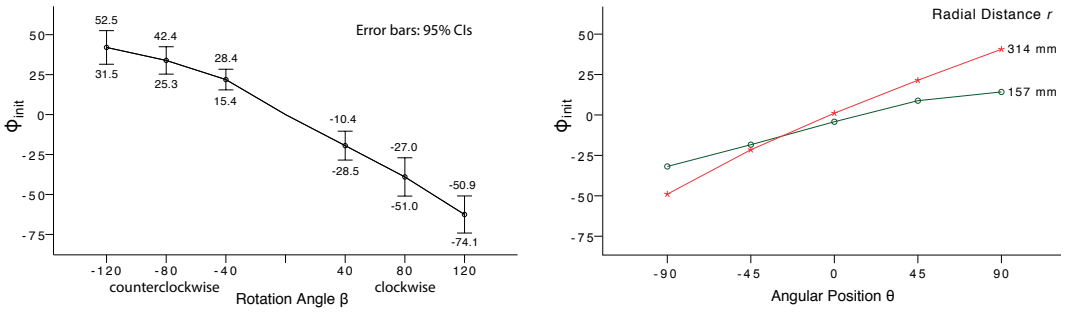


Figure 7. Experiment 2: Effects of the rotation angle β and the angular position θ on the grip orientation ϕ_{init}

Figure 7 illustrates the effects of θ ($F_{1.6,17.4} = 257.84$, $p < .001$), r ($F_{1,11} = 23.13$, $p < .001$), and $\theta \times r$ ($F_{2.3,25.5} = 42.26$, $p < .001$). The effect of θ decreases as r becomes shorter, and we can expect that it converges to zero as interaction approaches the user's position. Finally, we found no learning effects. The main effect of *Block* was not significant ($F_{2,22} = 1.68$, $p = .21$) and neither was its interaction with other factors ($p > .7$).

EXPERIMENT 3: TRANSLATIONS AND ROTATIONS

Experiments 1 and 2 showed planning effects for both translations and rotations. Experiment 3 tests how users plan their grasp orientation in preparation to more complex tasks where rotation and translation occur in parallel.

Participants and Task

This study involved the same participants as Experiment 2. We tested six *start* and *target* positions, where θ_{start}, $\theta_{target} \in \{-60°, 0°, 60°\}$ and r_{start}, $r_{target} \in \{157 \text{ mm}, 314 \text{ mm}\}$. In addition to these positions that define the translational movement component, we tested three angles of rotation $\beta \in \{-90°, 0°, 90°\}$.

Design

We followed a within-participants full-factorial design:

12 participants
× 3 Blocks
× 3 θ_{start} ($-60°$, $0°$, $60°$) × 2 r_{start} (157 mm, 314 mm)
× 3 θ_{target} ($-60°$, $0°$, $60°$) × 2 r_{target} (157 mm, 314 mm)
× 3 β ($-90°$, $0°$, $90°$)

= 3888 tasks in total

Participants performed 15 practice tasks prior to the experiment. The order of tasks within each block was randomized and the experiment took 30-35 minutes to complete.

Results

For errors, we used the Wilcoxon signed-rank and the Friedman tests. For *RT* and ϕ_{init}, we conducted full 6-way RM ANOVAs. For $\phi_{default}$, we conducted a 3-way RM ANOVA, where $\theta_{target} = \theta_{start}$, $r_{target} = r_{start}$, and $\beta = 0$.

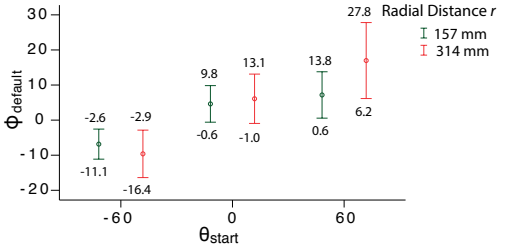


Figure 8. Experiment 3: $\phi_{default}$ in different screen locations

Errors

ErrorRate was 5.5% ($SD = 3.5\%$). As in Experiment 1, leftward movements produced more errors ($Z = -2.10$, $p = .036$). More specifically, *ErrorRate* for movements starting from $60°$ and ending at $-60°$ was 12.0% ($SD = 11.2\%$) compared to a 4.6% ($SD = 4.6\%$) for the reverse movements. *ErrorRate* for outward and inward movements was 8.7% ($SD = 7.1\%$) and 4.0% ($SD = 4.1\%$), respectively, but this difference was not significant ($Z = -1.91$, $p = .056$). Similarly, the effect of the rotation angle β was only marginally significant ($\chi^2 = 5.91$, $p = .052$).

Reaction Time

The effect of β was significant ($F_{2,22} = 11.57$, $p < .001$). Clockwise rotations were again 60 ms longer (CI: [12 ms, 107 ms], $p = .019$) to plan than counterclockwise rotations, increasing *RT* from 1068 to 1128 ms. The effect of *Block* was significant ($F_{1.2,13.3} = 8.25$, $p = .01$) for this experiment. The increased task difficulty could explain this result.

Grasp Orientation

Figure 8 presents our results for the default grasp orientation $\phi_{default}$. We found a significant effect of both θ_{start} ($F_{1.3,14.0} = 18.24$, $p < .001$) and its interaction $\theta_{start} \times r_{start}$ ($F_{2,22} = 7.04$, $p = .004$). As in Experiment 1, $\phi_{default}$ increases with θ_{start}. The effect is stronger for distant ($r = 314$ mm) than for close objects ($r = 157$ mm).

We then analyzed the initial grasp orientation ϕ_{init}. We found significant effects for θ_{start} ($F_{1.3,14.7} = 81.9$, $p < .001$), r_{start}

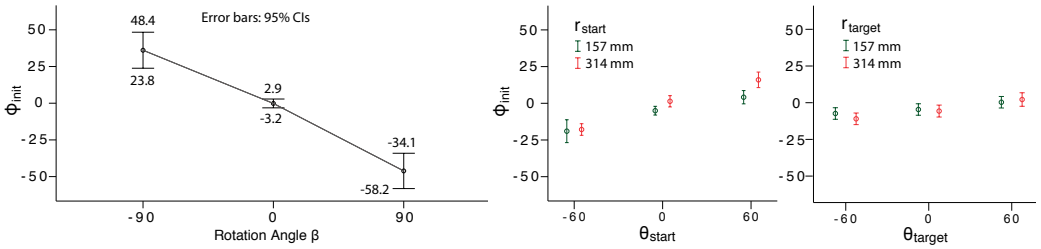


Figure 9. Experiment 3: Effects of the rotation angle β, the start position θ_{start} and the target position θ_{target} on the grip orientation ϕ_{init}

($F_{1,11} = 26.73$, $p < .001$), and θ_{target} ($F_{1.4,15.2} = 27.19$, $p < .001$). The effect of r_{target} was not significant ($F_{1,11} = 3.25$, $p = .099$). However, its interaction $r_{target} \times \theta_{target}$ was significant ($F_{2,22} = 8.44$, $p = .002$), as was the interaction $r_{start} \times \theta_{start}$ ($F_{2.5,27.4} = 5.96$, $p = .004$). Overall, grip adaptation was more pronounced for distant positions ($r = 314$ mm). Figure 9 illustrates these effects. Results are consistent with the findings of Experiment 1. Participants adapted their grasp orientation based on both the *start* and the *target* position of their movement. Again, the bias of the *start* position was stronger than the bias of the *target* position.

Finally, the effect of the rotation angle β was significant ($F_{1.1,11.7} = 61.43$, $p < .001$). As shown in Figure 9, results follow closely results of Experiment 2. Participants anticipated how to adapt their initial grasp despite to the translation movement that occurred in parallel with the rotation task. As in Experiment 2, we did not observe any learning effect.

SYNTHESIS OF FINDINGS

Our results support our hypothesis, being in accordance with the general principles of Herbort's WIMB model for physical objects [6]. Users plan their grasp orientation in preparation for the manipulation of virtual objects. Planning takes place under the influence of several biases that include at least a *task-independent preferred bias* and an *anticipatory bias*. When planning is not possible, as in the *Hidden target* condition of experiment 1, participants adopt the strategy of using a "standard" initial grip for all target positions (see Figure 6).

In all the three experiments, we found that the initial grasp orientation ϕ_{init} is influenced by both the *start* and *target* configurations. Experiment 1 showed that users adapt their ϕ_{init} to account for the difference between the *start* and *target* value of $\phi_{default}$, which varies across distant angular positions (see Figures 5 and 8). Experiment 2 showed that users adapt their ϕ_{init} in preparation for rotations so that they do not end up in uncomfortable positions. Experiment 3 examined both translations and rotations and showed that both of the above effects occur in parallel, with planning for rotations having a stronger effect. Finally, we observed that in special cases the *start* and *target* configurations are not the only factors to affect grasp orientation. In Experiment 1, *Outward* tasks, participants used noticeably different planning strategies for the $-45°$ target position, demonstrated by the large confidence interval of ϕ_{init} (see Figure 6). Some participants chose to "push" the object with a positive ϕ_{init} while others preferred to "pull" it using a negative ϕ_{init}. This suggests that in some situations, different planning strategies can be appropriate for the same task. We plan to further investigate this observation in future work.

As the studies reviewed by Herbort [6] considered only rotational tasks, we can check if our results of rotations fit the same formal model. Figure 10 presents the results of Experiment 2 through WIMB's mathematical formulation (see Equation 1) for $r = 317$ mm. We have normalized the initial and default grasp orientations by setting $p_{init} = \phi_{init} - \phi_{default}$ and $p_{default} = 0$, where the default orientations $\phi_{default}$ are the values measured by Experiment 1. Following Herbort's [6] approach, we examine clockwise and counter-clockwise rotations separately. Our results are consistent with previous results on the manipulation of physical objects, summarized in his survey. As WIMB predicts, we observe that users tend to compensate small angles proportionally more than large ones. We also observe that the effect of the anticipatory bias is stronger for clockwise rotations. We hypothesize that this is due to the fact that the range of motion is smaller in clockwise than counter-clockwise direction at most screen positions (see Figure 3). When a task involves a clockwise rotation, participants are required to do a larger (than if the task was a counter-clockwise rotation) preparatory rotation in the opposite direction to avoid uncomfortable or even impossible hand and arm positions. This asymmetry in movement direction may also explain why we observe a longer planning time (i.e., reaction time) for clockwise rotations in Experiments 2 and 3.

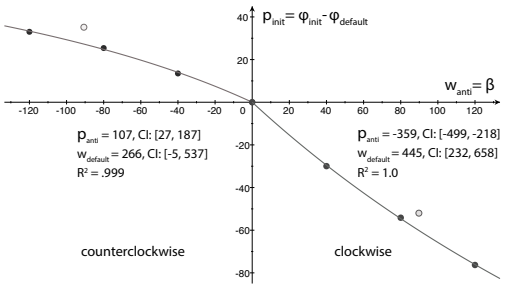


Figure 10. The WIMB model for Experiment 2, when $r = 314$ mm. We assume that $p_{default} = 0$. The empty circles show the results of Experiment 3 for simple only rotations.

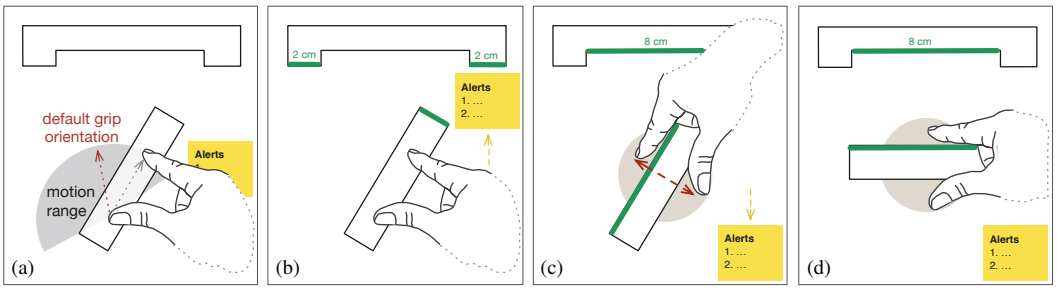


Figure 11. Using movement planning to improve feedback and prevent occlusion in an object-matching scenario. (a) The user acquires an object along its main graspable axis. (b) The system anticipates an anticlockwise rotation and indicates exact matches between its short edge and edges of the top object. The "alerts" box moves upwards, avoiding hand occlusion. (c) A pinch gesture over the object creates a circular grip that allows the user to grasp and rotate it clockwise. The system anticipates the planned rotation and indicates matches of it long edge. It also moves the "alerts" box downwards to minimize occlusion during the manipulation of the object. (d) The user has rotated the object in order to move it to the hole of the object at the top.

Finally, we found that clockwise rotations were more error prone than counterclockwise rotations. These results are in agreement with the results of Hoggan et al. [9] who concluded that performance is significantly inferior for clockwise rotations. The planning effect we observe in our experiment seems to be at odds with those of Möllers et al. [15], which did not observe prospective planning in a sequential pointing task on a multi-touch screen. However, looking closer at their task, we can see that comfort plays a minor role while *start* and *target* finger orientations are not constrained by each other. We suspect that movement planning in this case adds cognitive overhead without necessarily aiding the task.

IMPLICATIONS AND FUTURE DIRECTIONS

Our results open a new space for innovation with design implications for several application scenarios. First, they can inform the design of the form and affordances of virtual objects around a tabletop. Different surface positions are associated with different ranges of motion and different default grasps. Designers can make use of this information to appropriately position objects on the surface or design grips and interaction techniques that facilitate grasping (Figure 11-c).

Getting knowledge about the planned movement early enough when the user acquires an objet can be also valuable for improving user experience during its manipulation. We are particularly interested in exploring the design of new occlusion-aware techniques [4, 25]. Enhancing existing hand-occlusion models for multitouch [25] with a movement-planning model could possibly provide more reliable estimation about the occluded areas at acquisition time or during manipulation. Such information could be useful for optimizing the display of feedback and visual content at visible locations of the screen. It could be also useful for improving motor control, e.g., by avoiding object snapping around positions that are away from predicted targets. We do not encourage designs that make blind use of such predictions, as this could be the source of user frustration in case of false predictions. Figure 11 illustrates a simple scenario where movement planning is used to optimize visual feedback and reduce hand occlusion.

Our results could be also useful in collaborative scenarios where spatial interference and conflicts between the actions of collaborators are frequent [10]. We can foresee conflict-resolution techniques that make use of information about prospective movement. In addition, when users organize pieces of information collaboratively, the system could detect potential relationships between objects located in different personal workspaces and assist users with appropriate visual feedback. For example, it could display handles around an object that suggest a grasp and thus a specific movement that would bring this object close to other related ones.

Finally, we are interested in studying the role of movement planning for other multitouch devices, such as tablets, especially in connection with how users grasp and hold them [26]. Future work also needs to explore its implications for tangible user interfaces, where grasping and acquisition are determinant factors of user performance [3, 24].

CONCLUSION

Translational and rotational tasks are manipulations commonly performed on multitouch tabletops. We have investigated whether prospective planning is present when people perform such manipulations. We have shown that users choose a grip orientation that is influenced by three factors: (1) a preferred orientation defined by the start object position, (2) a preferred orientation defined by the target object position, and (3) the anticipated object rotation. We have examined these results in the view of the WIMB model, which has been recently introduced by Herbort [6] to explain planning for the manipulation of physical objects. We have shown that our results are consistent with the WIMB model.

We have also shown that relative to the geometry of the tabletop, upwards, leftwards movements and clockwise rotations are more difficult for users to perform. While the effects of planning on interaction with multitouch interfaces are not yet fully understood, our results provide a first look at a phenomenon that should be taken into account when designing tabletop applications.

Acknowledgments. This work was partly funded by ANR-11-JS02-004-01.

REFERENCES

1. Cohen, R. G., and Rosenbaum, D. A. Prospective and retrospective effects in hulman motor control: planning grasps for object rotation and translation. *Psychological Research 75*, 4 (2011), 341–349.
2. Cumming, G., and Finch, S. Inference by eye: confidence intervals and how to read pictures of data. *American Psychologist 60*, 2 (2005), 170–180.
3. Fitzmaurice, G. W., and Buxton, W. An empirical evaluation of graspable user interfaces: towards specialized, space-multiplexed input. In *Proc. CHI '97*, ACM (1997), 43–50.
4. Hancock, M. S., and Booth, K. S. Improving menu placement strategies for pen input. In *Proc. GI '04*, Canadian Human-Computer Communications Society (2004), 221–230.
5. Hancock, M. S., Carpendale, S., Vernier, F. D., Wigdor, D., and Shen, C. Rotation and translation mechanisms for tabletop interaction. In *Proc. TABLETOP '06*, IEEE Computer Society (2006), 79–88.
6. Herbort, O. Optimal versus heuristic planning of object manipulations: A review and a computational model of the continuous end-state comfort effect. *New Ideas in Psychology* (2013), 1–11.
7. Herbort, O., and Butz, M. The continuous end-state comfort effect: weighted integration of multiple biases. *Psychological Research 76*, 3 (2012), 345–363.
8. Hinrichs, U., and Carpendale, S. Gestures in the wild: studying multi-touch gesture sequences on interactive tabletop exhibits. In *Proc. CHI '11*, ACM (2011), 3023–3032.
9. Hoggan, E., Williamson, J., Oulasvirta, A., Nacenta, M., Kristensson, P. O., and Lehtio, A. Multi-touch rotation gestures: Performance and ergonomics. *Proceedings of CHI 2013* (2013), 3047–3050.
10. Hornecker, E., Marshall, P., Dalton, N. S., and Rogers, Y. Collaboration and interference: awareness with mice or touch input. In *Proc. CSCW '08*, ACM (2008), 167–176.
11. Jeannerod, M. The timing of natural prehension movements. *Journal of motor behavior 16*, 3 (1984), 235–254.
12. Kruger, R., Carpendale, S., Scott, S. D., and Tang, A. Fluid integration of rotation and translation. In *Proc. CHI '05*, ACM (2005), 601–610.
13. Lozano, C., Jindrich, D., and Kahol, K. The impact on musculoskeletal system during multitouch tablet interactions. In *Proc. CHI '11*, ACM (2011), 825–828.
14. Marteniuk, R. G., MacKenzie, C., Jeannerod, M., Athenes, S., and Dugas, C. Constraints on human arm movement trajectories. *Canadian Journal of Psychology 41*, 3 (1987), 365–378.
15. Möllers, M., Dumont, N., Ladwig, S., and Borchers, J. Improving touch accuracy on large tabletops using predecessor and successor. In *Proc. CHI '13*, ACM (2013), 755–758.
16. Morris, M. R., Wobbrock, J. O., and Wilson, A. D. Understanding users' preferences for surface gestures. In *Proc. GI '10*, Canadian Information Processing Society (2010), 261–268.
17. Moscovich, T., and Hughes, J. F. Multi-finger cursor techniques. In *Proc. GI '06*, Canadian Information Processing Society (2006), 1–7.
18. Rosenbaum, D. A., Chapman, K. M., Weigelt, M., Weiss, D. J., and van der Wel, R. Cognition, action, and object manipulation. *Psychological Bulletin 138*, 5 (2012), 924–946.
19. Rosenbaum, D. A., Marchak, F., Barnes, H. J., Vaughan, J., Slotta, J., and Jorgensen, M. Constraints for action selection: overhand versus underhand grips. In *Attention and Performance XIII*, M. Jannerod, Ed. Erlbaum, Hillsdale, NJ, 1990, 321–342.
20. Rosenbaum, D. A., van Heugten, C. M., and Caldwell, G. E. From cognition to biomechanics and back: The end-state comfort effect and the middle-is-faster effect. *Acta Psychologica 94*, 1 (1996), 59–85.
21. Rosenbaum, D. A., Vaughan, J., Barnes, H. J., and Jorgensen, M. J. Time course of movement planning: Selection of handgrips for object manipulation. *Journal of Experimental Psychology: Learning, Memory, and Cognition 18*, 5 (1992), 1058–1073.
22. Santello, M. Kinematic synergies for the control of hand shape. *Archives Iteliennes de Biologie 140* (2002), 221–228.
23. Short, M. W., and Cauraugh, J. H. Precision hypothesis and the end-state comfort effect. *Acta Psychologica 100*, 3 (1999), 243–252.
24. Tuddenham, P., Kirk, D., and Izadi, S. Graspables revisited: multi-touch vs. tangible input for tabletop displays in acquisition and manipulation tasks. In *Proc. CHI '10*, ACM (2010), 2223–2232.
25. Vogel, D., and Casiez, G. Hand occlusion on a multi-touch tabletop. In *Proc. CHI '12*, ACM (2012), 2307–2316.
26. Wagner, J., Huot, S., and Mackay, W. Bitouch and bipad: designing bimanual interaction for hand-held tablets. In *Proc. CHI '12*, ACM (2012), 2317–2326.
27. Wobbrock, J. O., Morris, M. R., and Wilson, A. D. User-defined gestures for surface computing. In *Proc. CHI '09*, ACM (2009), 1083–1092.
28. Wu, M., and Balakrishnan, R. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. In *Proc. UIST '03*, ACM (2003), 193–202.

TouchTokens: Guiding Touch Patterns with Passive Tokens

Rafael Morales Gonzalez[2,1,4] **Caroline Appert**[1,2,4] **Gilles Bailly**[3,4] **Emmanuel Pietriga**[2,1,4]
[1]Univ Paris-Sud & CNRS ; [2]INRIA [3]Telecom-ParisTech & CNRS [4]Univ Paris-Saclay
Orsay, France Paris, France France
morales@lri.fr appert@lri.fr gilles.bailly@telecom-paristech.fr emmanuel.pietriga@inria.fr

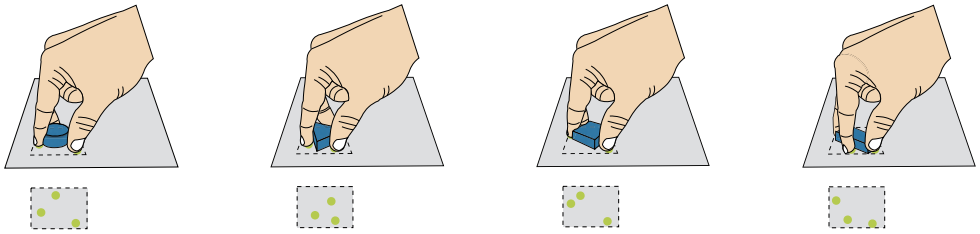


Figure 1. TOUCHTOKENS **are passive tokens that guide users' fingers to specific spatial configurations, resulting in distinguishable touch patterns.**

ABSTRACT

TOUCHTOKENS make it possible to easily build interfaces that combine tangible and gestural input using passive tokens and a regular multi-touch surface. The tokens constrain users' grasp, and thus, the relative spatial configuration of fingers on the surface, theoretically making it possible to design algorithms that can recognize the resulting touch patterns. We performed a formative user study to collect and analyze touch patterns with tokens of varying shape and size. The analysis of this pattern collection showed that individual users have a consistent grasp for each token, but that this grasp is user-dependent and that different grasp strategies can lead to confounding patterns. We thus designed a second set of tokens featuring notches that constrain users' grasp. Our recognition algorithm can classify the resulting patterns with a high level of accuracy (>95%) without any training, enabling application designers to associate rich touch input vocabularies with command triggers and parameter controls.

Author Keywords

Tangible interaction; Multi-Touch input

ACM Classification Keywords

H.5.2 : User Interfaces - Graphical user interfaces.

INTRODUCTION

The main characteristics of multi-touch gestures performed on the capacitive screens that typically equip tablets, smartphones, touchpads, as well as some tabletops, are the number of fingers involved and the individual trajectories of those fingers. Examples include 2- or 3-finger slide, and 2-finger pinch. But to the exception of a few research projects that consider touch points as chords [19, 21], interactive systems ignore the relative spatial configuration of contact points; what we call a *touch pattern*.

Our goal is to enable users to perform gestures based on a set of distinct touch patterns, thereby increasing the richness of input vocabularies for tactile surfaces. Our approach relies on physical guidance, as it would be unrealistic to expect touch patterns to be executed consistently across users, or even over time by the same user. As the literature suggests that users adopt grasp strategies that depend on the object to manipulate [39, 47], we investigate the potential of tangible tokens held on the surface to act as physical guides constraining the relative position of users' fingers.

We present TOUCHTOKENS, a novel interaction technique based on a set of easy-to-make passive tokens and a fast and simple recognition algorithm that can discriminate the unique touch pattern associated with each token in the set.[1] The approach features several advantages. First, physical tokens can provide space-multiplexed input by associating different controllers with different functions [18]. Second, tokens can alleviate issues related to discovery, exploration and learning inherent to gesture-based interaction [56]. Finally, tokens provide haptic feedback that promotes eyes-free interaction [28].

TOUCHTOKENS make it easy to implement applications that combine multi-touch and tangible input at low cost. Such a combination has the potential to foster collaboration, support distributed cognition, and enhance the user experience [1, 29, 46]. As opposed to other tangible systems that require electronic instrumentation (e.g., [9, 34]) or specific conductive material (e.g., [17, 33]), our system relies on an algorithm

CHI'16, May 07–12, 2016, San Jose, CA, USA.
 ISBN 978-1-4503-3362-7/16/05...$15.00.
http://dx.doi.org/10.1145/2858036.2858041

[1]Implementations of the algorithm and vector descriptions of the tokens ready for 3D-printing or laser-cutting are available at https://www.lri.fr/~appert/touchtokens/.

that relies on standard multi-touch APIs and on passive tokens thats can be made of any non-conductive material, including wood or transparent acrylic.

We performed a formative user study to collect touch patterns, in which participants had to grasp and manipulate a set of twelve tokens of varying shape and size on a tabletop surface. The analysis of this pattern collection showed that people grasp the same token consistently across trials, but that it is quite difficult to identify a set of tokens and to design a robust recognition algorithm that works for all users. The two main sources of confusion are that different users may adopt different grasp strategies for the same token, and that one user may adopt the same strategy for distinct tokens. Based on these observations, we designed a second set of six tokens featuring notches that constrain users' grasp. These notches are designed to ensure a comfortable grasp while serving two purposes: 1) minimizing, for a given token, the variability of the contact points' relative position; and 2) maximizing the distinctiveness of touch patterns. We performed a summative study in which participants had to grasp and manipulate this set of tokens on both a tabletop and a tablet. Results show that our algorithm recognizes these touch patterns with an accuracy higher than 95%, and does so without any training or calibration. Application designers can map the gestures performed with these tokens to any command or parameter control, as illustrated in the examples introduced before the concluding discussion about limitations and future work.

RELATED WORK

TOUCHTOKENS makes use of physical tokens to augment the power of expression of multi-touch input, building upon tangible interaction and touch input research. Our review of related work is structured accordingly, giving an overview of projects that considered tangible tokens above interactive surfaces, or leveraged the power of expression of touch input.

Tangible tokens for tactile surfaces

Some tactile surfaces rely on *diffuse illumination*, which makes it possible to recognize both objects and hands in contact with the surface, using computer-vision algorithms to analyze the frames captured by IR cameras. Such techniques have been used, e.g., to track mice and keyboards [24] or to design physical widgets [50]. The Conté tool [49] is an artistic crayon that consists of an acrylic block that emits and reflects IR light. When tethered, its location and orientation can be tracked on diffuse illumination surfaces. Several projects rely on fiducial markers to ease the image-based analysis, such as the ReacTable [30, 31], which offers a tangible environment for music composition. Tokens can also be stacked on top of one another, using fiducial markers with transparent areas [3] or optical fiber bundles [5] to track them. Diffuse illumination hardware setups are somewhat bulky, however, and are thus mostly used for large surfaces such as tabletops.

Most touchscreens are *capacitive*: they detect a drop in capacitance when one or more fingers touch them. Various projects have investigated conductive objects. These objects contain a circuit of conductive material that links the areas that are in contact with the user's fingers to the areas that are in contact with the capacitive surface (the object's *"feet"*). As soon as the user touches an object, its feet become grounded and generate a drop in capacitance similar to a multi-touch pattern. Physical widgets [33] rely on this technique, as do physical button pads that can be clipped to the edges of a device [55], or more advanced objects that feature moving parts [17, 28] or that can be stacked [17]. Designing conductive tokens is challenging: the feet must be positioned carefully and the circuit must be stable so that the generated touch pattern can be recognized consistently. As capacitive screens have been designed for human fingers, properties such as the feet's minimal size and the minimal distance between two feet, which depend on the device, must be carefully chosen [55].

Other projects have explored more cost-effective ways of building conductive objects. Wiethoff et al. [51] use cardboard and conductive ink. This works well for low fidelity prototyping, but does not scale with real usages. Blagojevic et al. [11] report on a design experience where they have built a small set of geometric tools (ruler, protractor and set square) for a tabletop drawing application. They tested different construction strategies by combining different low-cost conductive materials (e.g., conductive ink, conductive foam, aluminium tape, copper wires). Their experience shows that making a physical tool conductive is quite difficult, as many factors have to be considered (consistent circuit, stability, friction with the screen, good grasp, etc.). In the end, the best design consisted of drilling holes in the tool and using conductive foam to cover the tool and fill the holes. In the panorama of capacitive tokens, PUCs [48] are an exception: they rely on the principle of mutual capacitance so as to be detected even when users do not touch them. However, most systems have to be augmented with an additional calibration clip to cheat the implemented adaptive filtering that tends to interfere with the PUCs' detection.

Some tangible systems work with *magnetic* tokens that modify the magnetic field incoming to the magnetometer built in mobile devices [8]. However, as a magnetometer reflects the sum of the magnetic fields it senses, supporting multiple tokens requires putting more than a simple magnet inside the tokens. Bianchi and Oakley [9] propose to use a more elaborate electronic system that features a motor to make the mounted magnet spin at a specific frequency. Putting a grid of Hall sensors behind the surface, Liang et al. [34] get a 2D image of the magnetic field that can be analyzed to track the location and orientation of objects above the surface. Each object can also be shielded with a case made of galvanized steel to avoid attraction and repellence effects between several tokens [35].

Multi-touch input and power of expression

Researchers have considered several avenues to increase the power of expression of touch input, including finger identification, finger pressure, or finger impact in order to multiplex input by assigning one command per finger, pressure level, or impacting zone. Finger identification relies on pattern recognition techniques coupled with advanced sensors such as fingerprint scanners [43] or fiber optic plates [27]. Projects such as SimPress [7] or FatThumb [12] capture the size of the finger's contact area and assign two different meanings to a soft

tap and a hard tap. It is even possible to capture both the normal and tangential components of the force applied on the surface using extra pressure sensors [25]. Identification and amount of pressure of the finger in contact can also be assessed by classifying muscle activity in the forearm [6]. Finally, TapSense [22] discriminates which part of the finger (nail, knuckle, pad or tip) hits the surface by using acoustic sensing. With the exceptions of SimPress and FatThumb, that capture two pressure levels based on the size of the contact area, all of the above techniques rely on tactile surfaces that are augmented with additional sensors.

Some systems make use of whole-hand gestures (e.g., horizontal vs. vertical hand, straight vs. curved hand) for manipulating virtual objects [16, 40, 52, 53], or invoking virtual tools by mimicking the hold of their physical counterparts [4, 23]. Most of these projects rely on tactile surfaces that give access to the shape of the whole contact region, and cannot run on regular capacitive surfaces, which have been developed for finger input and consequently deliver standard point-based multi-touch coordinates only. One notable exception is the TouchTools system [23] that uses machine learning to recognize up to seven touch patterns associated with seven hand postures on a capacitive screen. A few systems can also recognize chord gestures. Finger-Count [2] counts the number of contact points on the surface. Arpège [21] supports more chords by relying on the contact points' relative position. The technique requires per-user calibration to record the fingers' natural position when the hand rests in a comfortable posture.

TOUCHTOKENS take a different approach and does not make the assumption that the fingers' relative position is always the same. The technique relies on *different relative finger positions* that users would adopt naturally when grasping a tangible token on a surface. Recognizing typical hand postures when people grasp objects has been investigated in experimental psychology to identify everyday objects and then infer users' activities (such as holding a mug or typing at the keyboard) [39]. Experimental studies show that it is possible to distinguish objects that differ in their size [14], shape (e.g., cylinder, pyramid, etc.) [42] or both [39, 47]. However, these studies assume that the system provides access to the whole hand posture, using advanced motion capture systems that can provide the position and orientation of all hand joints.

TOUCHTOKEN

The primary objective of TOUCHTOKENS is to guide the *registration pose* [19] of multi-touch gestures on an interactive surface. TOUCHTOKENS take advantage of users' ability to grab physical objects in the real word. Our idea is that the geometry (shape and size) of an object impacts how users grab it. Different objects will thus have different touch patterns on the tactile surface, which can be discriminated. Touch patterns are recognized at registration and remain active until all contact points have left the surface. In particular, users can *relax* their grasp in the *execution phase* of their gesture [54], thus reducing finger occlusion and enabling a larger range of motion. In this section, we describe how to build a system based on TOUCHTOKENS. Applications and limitations of the approach are discussed at the end of the paper.

Fabrication

TOUCHTOKENS require neither embedding electronics in the tokens nor augmenting the tactile surface with additional hardware (such as, e.g., a computer vision system), which makes setup easy. Tokens can be built from any non-conductive material such as wood, plastic, metal or glass, since the system only relies on the fingers' relative position, which is already provided by the tactile surface. This flexibility allows designers to easily prototype and test different TOUCHTOKENS variants with a 3D printer or a laser cutter. In particular, designers have a lot of control on the tokens' appearance. For tokens that have permanent roles associated with them, interface designers can engrave an icon or a label on them, or use a specific color. For temporary associations, end-users could adopt more volatile solutions, such as adding stickers or writing with an erasable pen if the chosen material affords it (e.g., pencil on a wooden token). Tokens can also be made of transparent material such as glass or acrylic, to avoid occluding the content displayed on the tactile surface.

Recognition

When grabbing a token with more than two fingers in contact with the surface, TOUCHTOKENS can infer its identity, and thus the corresponding registration pose, from the relative spatial configuration of the touch points. The recognition engine is initialized with one or more typical touch patterns per token and, when a touch pattern of at least three points occurs, the algorithm computes the distance between this input pattern and the set of template patterns. The recognized token is the one associated with the template that minimizes this distance metric.

Computing the distance between two touch patterns (input I:$\{I_1, ..., I_n\}$ and template T:$\{T_1, ..., T_n\}$) is not straightforward, however. First, most tactile surfaces do not provide finger identification. Second, tokens have an arbitrary orientation on the surface. Figure 2 illustrates how our algorithm processes touch patterns in order to identify the *best alignment* between the reference template and actual input patterns, from which the distance is computed.

The key steps for identifying the best alignment are as follows: (1) compute the centroid C_I of the three (or more) touch points; (2) generate all sequences of touchpoint labels (permutations) so that their IDs always appear in counterclockwise order; (3) rotate all these touch patterns so as to align vector $\overrightarrow{C_I I_1}$ with the x-axis. (4) The algorithm then translates touch patterns to align the input (C_I) and template (C_T) centroids. (5) It finally pairs the points in the permutation with the template's points in order to compute the distance, simply by summing all distances between paired points. The distance between reference template and actual input is given by the *best input alignment*, which is the permutation that minimizes this distance metric.

A typical implementation of the recognition engine amounts to about a hundred lines of code, and will work on any capacitive surface. The engine relies on simple geometrical features, which makes it easier to understand recognition errors compared to less transparent techniques such as those based

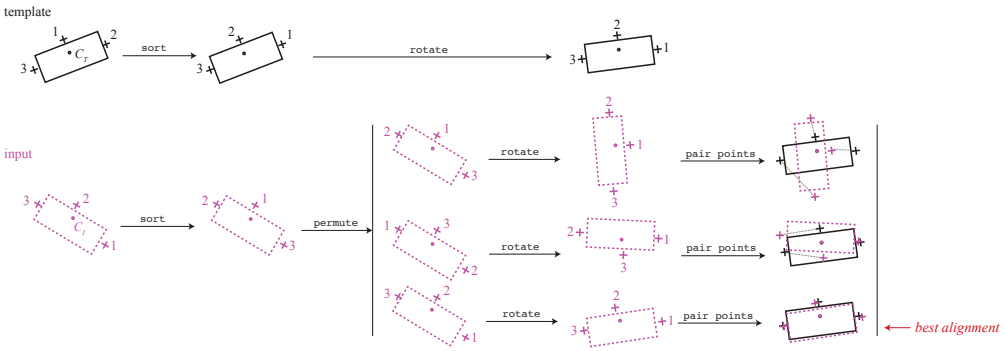


Figure 2. Template and input touch pattern alignment process.

on machine learning, that work as black boxes. The algorithm is very fast: recognition time scales linearly with the number of candidate templates. A Java implementation will be made available publicly, featuring both TUIO and Android APIs[2].

REGULAR TOKENS

TOUCHTOKENS rely on the hypothesis that the geometry of tokens impacts how users grasp them, resulting in distinguishable touch patterns. In order to test this hypothesis and identify a set of tokens that can actually be discriminated, we first ran a formative study in which participants had to grasp a set of twelve tokens that vary in shape and size.

Experiment design

Token Set

We selected a set of $4 \times 3 = 12$ tokens (Figure 3) that vary in their shape (square, circle, rectangle, and triangle) and size (3cm, 4cm and 5cm). The choice of size was informed by informal tests, taking into account both human and technological constraints. The tokens should remain comfortable to grasp with at least three fingers, which entails bio-mechanical constraints on the minimum and maximum token size. Capacitive surfaces also impose a minimal distance between finger tips, which will be seen as a single point if too close to one another. Our tokens are made of wood and are 6mm thick. We had initially considered tokens 3mm thick, but those were too difficult to grab. The tokens' corners are also slightly rounded so as to avoid sharp wedges that could have hurt participants.

Types of interaction

Participants are seated in front of the tabletop (at the center of the long edge) and perform a series of trials with the different tokens (Figure 4). As illustrated in Figure 5, the graphical display always features a progress bar in the top-left corner and a picture of the token to use in the current trial in the top-right corner. The action to be done with the token depends on the type of interaction (INTERACTION). The *Global* condition operationalizes the case where users invoke a global command with the token (e.g., launching an app); the *Local* condition corresponds to the case where users apply a command at a specific location on screen (e.g., copying a graphical object); and the *Path* condition captures the case where users invoke a command and set its parameter value with a gesture (e.g., adjusting the opacity of a layer in a visualization). The progress bar indicates for how long participants have *dwelled*. It starts filling-in as soon as a stable touch pattern is detected on the surface. The dwell's duration depends on the type of interaction. If the number of fingers in contact changes, or if the touch pattern's centroid drifts away from its initial position by more than 30 pixels, the progress bar is reset and participants have to perform the trial again.

[2]It is part of the earlier-mentioned supplemental material made available to reviewers.

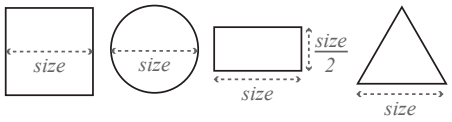


Figure 3. Set of tokens used in the first study ($size \in 3cm, 4cm, 5cm$).

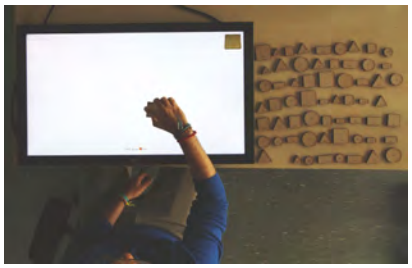


Figure 4. Experimental setup.

The experiment was divided into three phases, one per INTERACTION condition, always presented in the same order:

1. In the first phase (INTERACTION = *Global*), participants have to select the right token, put it anywhere on the tabletop, hold it with at least three fingers, and hold still for at least 1 second.

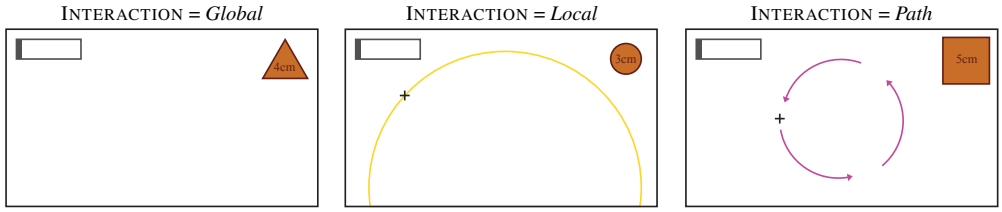


Figure 5. Types of interaction (INTERACTION).

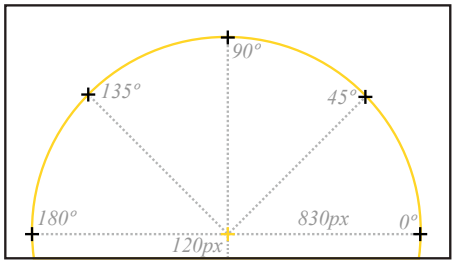


Figure 6. In the INTERACTION = ***Local*** **condition, participants have to put the token at a specific location** (LOCATION ∈ 0°, 45°, 90°, 135°, 180°).

2. In the second phase (INTERACTION = *Local*), participants have to select the right token, put it on the cross (Figure 6), holding it still with at least three fingers for at least 1 second. The cross can be in five different LOCATION. These locations are chosen on a semi-circle roughly centered on the participant as in [38] (see Figure 6), as the token's location on the surface (relative to the participant) may influence the neutral hand posture and thus how the token is grasped. The distance between the touch pattern's centroid and the center of the cross must be at most 50px. If this distance is greater, the progress bar turns red and participants must perform the trial again.

3. In the third phase (INTERACTION = *Path*), participants must hold the token still with at least three fingers for a short period of 100ms. The background turns from gray to white. Participants then have to slide the token along the path indicated by purple arrows. In this condition, participants can plan a manipulation with the token, which may influence their initial grasp [38]. When sliding the token, they can lift some fingers but must keep at least one finger in contact with the surface. If they lift all fingers before having performed the whole gesture, the background turns back to gray and they have to start again. Figure 7 shows the six types of paths that participants had to follow with each token. We chose these tasks based on the taxonomy of multi-touch gestures from [37]. For external circular gestures (*Ext-CCW* and *Ext-CW*), participants have to slide the token along a clockwise or counterclockwise circular path. As soon as the touch pattern's centroid has completed one full circle, the background turns green and participants can proceed to the next trial. For internal circular gestures

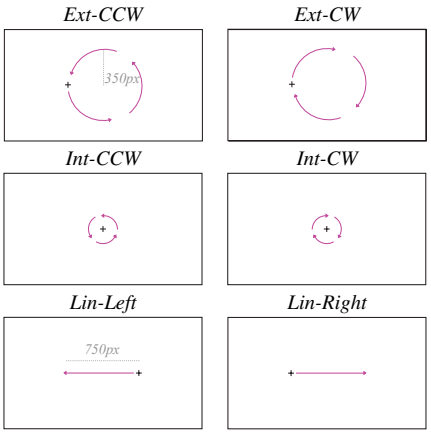


Figure 7. In the INTERACTION = ***Path*** **condition, the participant has to put the token on the surface and slide it along a specific path** (PATH ∈ {***Ext-CCW***, ***Ext-CW***, ***Int-CCW***, ***Int-CW***, ***Lin-Left***, ***Lin-Right***}).

(*Int-CCW* and *Int-CW*), participants have to rotate the token around its center, as they would do with a physical circular knob. As soon as the touch pattern has been rotated by at least 45° around its centroid, the background turns green to indicate that the trial has been successfully completed. Finally, for linear gestures (*Lin-Left* and *Lin-Right*), participants simply have to slide the token to match the amplitude and direction indicated by the arrow.

Participants and Apparatus

Twelve volunteers (3 female), aged 23 to 33 year-old (average 26.5, median 25.5), participated in the experiment. The experiment software was running in full screen mode on a 3M C3266P6 capacitive screen (display dimensions: 698.4 x 392.85 mm, resolution: 1920 x 1080 pixels) placed horizontally on a desk (Figure 4). A digital video camera on a tripod recorded participants' hand and arm movements. The experimental software was developed in Java 2D (JDK 7) and ran on a Mac Pro 2.8 GHz Intel Quad Core with 16GB memory, running Mac OS X 10.7.5.

Procedure

Participants are seated at the center of the long side of the tabletop. They receive instructions detailing the goal of the experiment and the different experimental tasks they will

have to perform. In particular, the operator initially informs participants that the goal is to design a system that is able to recognize tokens based on users' grasp. He encourages them to be consistent in their grasp across trials with tokens that have the same shape. In order to identify which grasp is comfortable, the operator gives participants four tokens, one per shape with *size* = 4*cm* (*Square*$_4$, *Circle*$_4$, *Rectangle*$_4$ and *Triangle*$_4$), and asks them to manipulate each token a bit on the surface in order to choose a comfortable grasp. The operator then notes this grasp in his logs and the experiment starts.

As mentioned above, the experiment consists of three phases that are always presented in the same order:

- Phase 1 (INTERACTION = *Global*): 12 TOKEN × 5 repetitions = 60 trials. In this phase, the presentation order for the trials is randomized in order to observe if people are actually able to grasp the same token consistently across different trials that are not consecutive. To minimize the visual search time associated with identifying the right token to take, the operator printed 5 copies of each individual token and initially sorted the 60 tokens on the table, on the right side of the screen (Figure 4).
- Phase 2 (INTERACTION = *Local*): 12 TOKEN × 5 LOCATION × 2 repetitions = 120 trials. The order of TOKEN × LOCATION is randomized across participants. The 2 repetitions per TOKEN × LOCATION condition are presented one after another to limit the length of the experiment.
- Phase 3 (INTERACTION = *Path*): 12 TOKEN × 6 GESTURE × 2 repetitions = 144 trials. As in phase 2, the order of TOKEN × GESTURE conditions is randomized across participants, with the 2 repetitions presented one after another.

After completion of these three phases, participants receive a questionnaire where they have to give a comfort score for each of the twelve tokens. The questionnaire features 12 Likert-scale type questions where participants have to give a rating between 0 (not comfortable to grasp at all) to 5 (very comfortable). The overall procedure lasted about an hour.

Results

We first tested if participants' grasps of the different tokens can be distinguished using the recognition strategy described in the previous section. To that end, we train our recognition algorithm using the first three trials of Phase 1 as templates for each token. This training strategy corresponds to what a system relying on a light training phase would require.[3] We then evaluate our algorithm on the remaining trials, i.e., $(2 \times 12 + 2 \times 12 \times 5 + 2 \times 12 \times 6) \times 12$ participant = 3456 trials.

A χ^2 analysis reveals that both INTERACTION ($\chi^2(2, N = 3456) = 12, p = 0.002, \phi = 0.06$) and TOKEN ($\chi^2(11, N = 3456) = 109, p < 0.001, \phi = 0.18$) have a significant effect on RECOGNITION RATE. Figure 8 illustrates the observed differences between conditions. A finer analysis of TOKEN's effect on RECOGNITION RATE per INTERACTION shows that TOKEN has a significant effect on RECOGNITION RATE in all INTERACTION conditions (*Global*: $\chi^2(2, N = 288) = 25, p = 0.009, \phi = 0.3$, *Local*: $\chi^2(2, N = 1440) = 58, p < 0.001, \phi = 0.2$ and *Path*: $\chi^2(2, N = 1728) = 85, p < 0.001, \phi = 0.2$). The effect of secondary factors (LOCATION for *Local* and GESTURE for *Path*) on RECOGNITION RATE is not significant ($p = 0.4$ and $p = 0.2$).

[3] We tested our algorithm with different training strategies to accommodate more variability (e.g., considering templates picked from the three experiment phases) but there was no clear gain compared against the training cost it would entail for end-users.

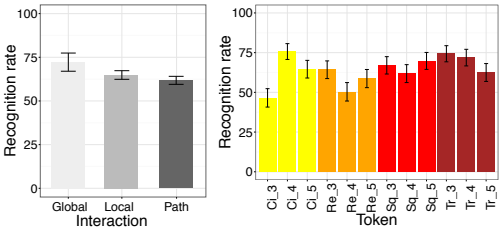


Figure 8. Recognition rate per INTERACTION (left) and per TOKEN (right). Error bars represent the 95% confidence interval.

We then wanted to investigate the impact of the token subset's size on recognition rate. In order to identify the largest number of grasps that can be accurately discriminated for each participant, we computed all possible subsets of tokens among the initial set of 12. The total number of subsets comprising at least two tokens (TOKENCOUNT >= 2) is:

$$\sum_{TokenCount=2}^{12} \binom{12}{TokenCount} = 2^{12} - 12 - 1 = 4083$$

For each subset, we ran our recognition algorithm with the same training strategy (only the first three trials from experiment phase INTERACTION = *Global*) in order to compute, for this subset, the recognition rate per participant. We observe that the per-subset recognition rate across participants exhibits a very high variability. For example, if we consider subsets that have 7 tokens (TOKENCOUNT = 7), the "worst" subset has a recognition rate of 63% on average across participants (worst-performing participant: 31%, best-performing participant: 98%), while the "best" subset has a recognition rate of 81% on average across participants (worst-performing participant: 57%, best-performing participant: 100%).

Recognition rate per participant

In order to test how many distinguishable grasps can be recognized per participant, we report the maximal recognition rate for each value of TOKENCOUNT $\in \{2, ..., 12\}$. If a participant P gets a maximal recognition rate R for TOKENCOUNT=N, this means that there exists at least one set of N tokens that are recognized with R% accuracy on average for participant P. Figure 9 reports these recognition rates for the best-performing and worst-performing participants, as well as the average over all participants. The charts illustrate that our algorithm can accurately discriminate a high number of grasps for some participants (the best-performing participant has a recognition accuracy higher than 90% for up to 10 tokens in all INTERACTION conditions), while it performs quite poorly for others (the worst-performing participant has

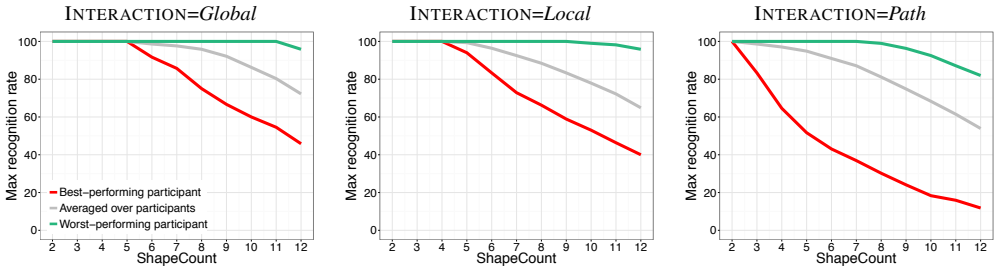


Figure 9. Recognition rate per INTERACTION × TOKEN

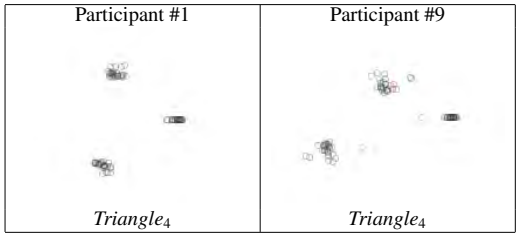


Figure 10. Touch patterns (aligned by our algorithm) for a participant who adopts very consistent grasps for token *Triangle*$_4$ (left) and for a participant who adopts varying grasps (right). Red dots belong to touch patterns that are used as templates.

a recognition accuracy lower than 90% even for sets of only three tokens in condition INTERACTION = *Path*). This variability comes from two sources: *intra-grasp variability* and *inter-grasp similarity*.

Figure 10 displays the 27 touch patterns we have collected for *Triangle*$_4$ for two participants. It illustrates two extreme levels of *intra-grasp variability*. Participant 1 (left) grasps token *Triangle*$_4$ in a very consistent manner, while Participant 9 (right) demonstrates much more variation in how he grasps it, challenging our recognition strategy. The second source of confusion comes from *inter-grasp similarity*: if a user chooses one grasp strategy for a given token that is very similar to the one he uses for another token in terms of similarity of the touch patterns, the two tokens will get confounded. Together, these two phenomena explain why we observe such a large variability across participants regarding the composition of the token sets that are recognized accurately.

Recognition rate between participants

Figure 9 reports the best set of tokens for each participant, and thus does not reflect the fact that the same subset of tokens can be very accurately recognized for one participant while it will be poorly recognized for another participant. We report the biggest sets of tokens that reach consensus among all our participants below (i.e., the sets of tokens that have a recognition accuracy of at least 90% for all participants):

- for INTERACTION=*Global*, we find 6 sets of 5 tokens ;
- for INTERACTION=*Local*, we find 13 sets of 3 tokens ;
- for INTERACTION=*Path*, we find 6 pairs of tokens ;

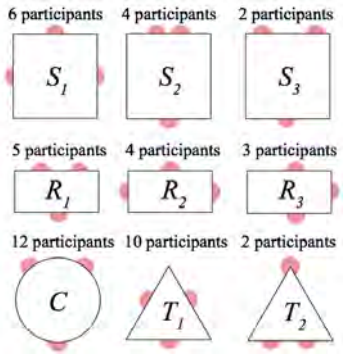


Figure 11. The nine grasp strategies observed in Experiment 1

- for all INTERACTION conditions undifferentiated, we find 8 sets of 3 tokens with an average of at least 90% accuracy for all participants.

Grasp strategies

Figure 11 summarizes the different grasp strategies that participants adopted for the different token shapes (extracted from an analysis of the operator's logs and video sequences recorded during the experiment). We observed that all participants use the same strategy for circles (C). Squares and rectangles receive less consensus, with three different strategies observed for each of them. The main strategy for squares uses three edges (S_1; 6/12). The two other strategies use only two edges, and differ in the distance between the two fingers on the same edge: small (S_2; 4/12) or large (S_3; 2/12). For rectangles, one strategy uses the two long edges only (R_1; 5/12). The two other strategies use three edges: two contact points on the short edges (R_2; 4/12) or on the long edges (R_3; 3/12). One of the grasp strategy for triangles makes use of a corner (T_2; 2/12), which was quite surprising. Two participants adopted it, but actually rated it as very uncomfortable.

To understand what kind of confusions occur in the recognition process, we implemented a visualization that displays all collected touch patterns using the best alignment computed by our recognition algorithm (Figure 10 was built with this tool). We computed the confusion matrix by considering the

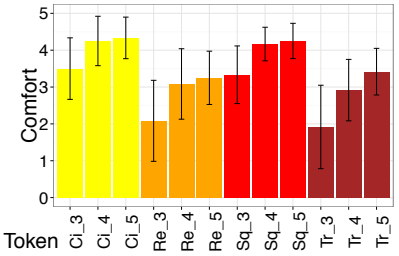


Figure 12. Comfort score per token. Error bars represent the 95% confidence interval.

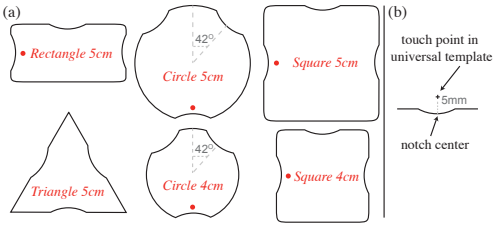


Figure 13. (a) The 6 tokens with notches. (b) The touch point's location in the template is offset by 5mm along the normal to the token's edge.

27 types of touch patterns (3 *size* × 9 grasp strategies). The visualization tool was a good complement to the confusion matrix as (1) some confusions do not appear in the matrix if a template for one token is too close to a template for another token; and (2) the different grasp strategies were not adopted the same number of times, leading to numbers in the confusion matrix that could not be compared in an absolute manner. From this analysis, we draw a few take-away messages. The flat isosceles triangle of grasp strategy R_2 is very representative and well-recognized. T_2 is also representative, but is too uncomfortable to be further considered. In contrast, some postures are difficult to distinguish. For instance, touch patterns R_1 and R_3 often form an equilateral triangle similar to the one of T_1. Finally, S_1 and C can also cause confusions.

TOKENS WITH NOTCHES

Our foundational hypothesis was that physical tokens constrain users' grasp in a consistent manner, which leads to consistent touch patterns that can be recognized with a high level of accuracy. The results of our formative experiment revealed that our hypothesis was only verified for some participants. We also observed significant variations in grasp strategies among users, which means that a set of tokens that works well for one user might not work so well for another user. As we aim at devising a solution that works effectively for all users in a consistent manner, we investigated a solution to decrease the different sources of variability.

We designed a new set of tokens, illustrated in Figure 13, similar to those considered in the formative study, but that feature notches. The purpose of these notches is to afford a particular grasp strategy, i.e., to suggest a specific way of positioning the fingers to grab a given token. The design of these tokens was guided by the following requirements. We wanted the token set to feature a wide range of shapes, as tokens should be easy to identify by visual and tactual perception [36]. Sets that feature different shapes also provide better mnemonic cues, making it easier for users to remember token-command associations. Finally, the tokens should remain comfortable to grasp. Based on these requirements, we picked the most comfortable size for each shape (5cm), and added the circular and square tokens of 4cm, which were also rated as very comfortable (Figure 12). We limited our summative study to this set of six tokens which, together with all token manipulation gestures, already provides a rich input vocabulary.

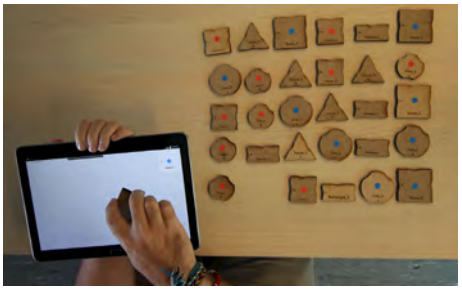


Figure 14. Experimental setup in the *Tablet* condition.

The grasp strategies observed during our formative experiment (Figure 11) informed the positioning of notches on token shapes. The notches' dimensions were refined through trial and error: narrow and deep notches introduce corners under finger tips, which make them uncomfortable; large and shallow notches are more comfortable, but introduce tangential variability in finger position. Our final design tries to strike a balance, and consists of notches 15mm wide and 1.5mm deep. Tokens whose shape afforded variable grasp strategies in the previous experiment feature a dot that indicates where to put the thumb, as illustrated in Figure 13-a.

These new tokens are designed to strongly constrain how users grasp them. We hypothesize that this will result in significantly reduced level of variability, which should enable our approach to work without any training. For each token, we compute a representative touch pattern that will act as a *universal template* for all users. The touch pattern is derived from the notches' position, slightly offset from the token's edge along the normal to that edge, so as to better capture users' grasp (Figure 13-b). The exact value of this offset (5mm) is calculated from the average offset measured in trials performed with circular tokens in the previous experiment, comparing the radius of the circle that passes through the three touch points with the radius of the actual physical token. (The precise, vector-based description of these tokens, ready for laser-cutting or 3D printing, will be distributed publicly and is also part of the earlier-mentioned supplemental material available to reviewers.)

EXPERIMENT

We ran a controlled experiment to test users' ability to manipulate tokens with notches, and to evaluate our algorithm's accuracy when provided with the above-mentioned universal templates in combination with this particular kind of tokens. The experimental design is similar to that of the previous study, but uses the set of 6 tokens of Figure 13. We also include an additional DEVICE condition: participants perform the tasks on both the tabletop and a tablet. Because of the smaller size of the tablet, we exclude the *Local* condition when DEVICE=*Tablet*, as the different locations (Figure 6) are clearly too close to one another to impact users' grasp. Contrary to our formative experiment, participants did not receive any other instructions than to grasp the tokens using the notches. In particular, the operator never asked them to adopt a consistent grasp across trials for a given token.

Experimental design

Procedure

Half of the participants started with the *Tabletop*, while the other half started with the *Tablet*. The strategy for counterbalancing the presentation order of trials is exactly the same as in the first experiment. The only difference lies in the *Tablet* condition, in which participants only performed *Global* and *Path* tasks (in this order), but not the *Local* task.

In the *Tabletop* condition, we collected 12 participants × 6 TOKEN × (5 [*Global*] + 2 × 5 LOCATION [*Local*] + 2 × 6 GESTURE [*Path*]) = 1944 trials. In the *Tabletop* condition, we collected 12 participants * 6 TOKEN × (5 [*Global*] + 2 × 6 GESTURE [*Path*]) = 1224 trials.

Participants & Apparatus

12 volunteers (3 female), aged 23 to 39 years-old (average 26.4, median 24.5), one left-handed, participated in this experiment. Five of them had participated in the previous study. The experimental setup for the *Tabletop* condition was exactly the same as in the previous experiment. In the *Tablet* condition, participants were seated at the same table, but had to hold the tablet during the whole experiment, as illustrated in Figure 14. The tablet (Samsung GT-P5110 Galaxy Tab 2) had a 256.7 x 175.3 mm display area with a resolution of 1280 x 800 pixels.

Results

As illustrated in Figure 15, the recognition rate in both DEVICE conditions is very high: 98.7% on the *Tabletop* and 99.3% on the *Tablet*. A χ^2 analysis reveals that the effect of INTERACTION on RECOGNITION RATE is significant neither in the *Tabletop* condition ($p = 0.8$) nor in the *Tablet* condition ($p = 0.3$). However, TOKEN has a significant effect in both DEVICE conditions (*Tabletop*: $\chi^2(5, N = 1944) = 30, p < 0.001, \phi = 0.12$ and *Tablet*: $\chi^2(5, N = 1224) = 30, p < 0.001, \phi = 0.16$)). Actually, in the *Tabletop* condition, the RECOGNITION RATE is a bit lower for $Circle_4$ (95.6%) than it is for all other tokens (> 98.7%). The same is true for token $Square_4$ (96.5%) in comparison with all other tokens (> 99%) in the *Tablet* condition.

Interestingly, even if we realized a posteriori that the thumb marker (dot) is meant for right-handed users, our left-handed participant did not have any trouble manipulating the tokens.

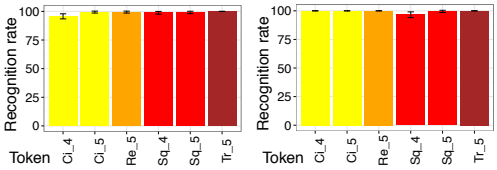


Figure 15. Recognition rate per TOKEN in the *Tabletop* (left) and *Tablet* (right) conditions. Error bars represent the 95% confidence interval.

He simply put his thumb in the notch opposite to the dot, ignoring the latter. Of course, he was able to do so because our tokens feature an axis of symmetry. However, we expect that TOUCHTOKENS's approach can be used for arbitrary-shape tokens, including some that would not feature a symmetric touch pattern. In that case, users can still flip them to accommodate their handedness, provided that the tokens are flat. If a token cannot be flipped easily, a solution would consist in designing two variants: same shape but pattern of notches mirrored. When the pattern cannot be mirrored because of the shape's geometry, it is still possible to design two patterns, one for each handedness.

APPLICATIONS

The above results show that, using tokens with notches, it is possible to build robust applications that will take advantage of both gesture-based and tangible interaction. Application domains that would benefit from such type of input are quite varied and have already been discussed in the literature, including: geographical information systems [32], database querying [29, 45], information management [41, 44] and music composition [30]. We developed a set of proof-of-concept applications[4] to illustrate the different roles that TOUCHTOKENS can play in an interactive system (see Figure 16).

TOUCHTOKENS can act as *controllers* or *filters*, and can be used to manipulate both the content of an application or the presentation of this content. For instance, they can be used to adjust the parameters of a visualization, enabling users to focus more on the result of their actions as the manipulation of physical tokens decreases the demand on visual attention [45]. We have developed a simple scatterplot visualization in D3 [13] to illustrate this idea. The different categories in the data (e.g., countries grouped by continent) are associated with different symbols (which have distinct shapes and colors), as is typically the case when visualizing multivariate datasets. One TOUCHTOKEN, with matching shape and color, is associated with each category and can be used to adjust the visual representation of the corresponding data points in the scatterplot: changing their size by rotating the token, and their opacity by sliding it. TOUCHTOKENS can be transparent, in which case they will typically be used as physical see-through tools [10, 15], altering the content that falls below the token (e.g., filtering) or changing its visual attributes (e.g., rendering). For instance, we have developed a simple mapping application in which tokens are associated with different layers. The tokens act as magic lenses [10] that

[4] All demonstrated in the companion video.

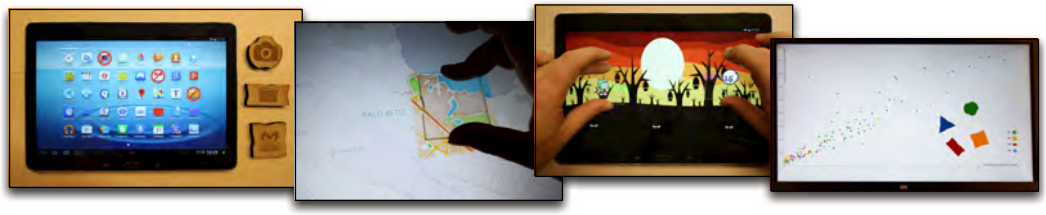


Figure 16. Proof-of-concept applications: access control, tangible magic lenses, character controllers in a game, data visualization.

reveal the corresponding layer while leaving the context untouched. See-through tools can also be used to move content in the workspace, as demonstrated in our simple game, where transparent tokens control the location and orientation of individual characters.

TOUCHTOKENS can also act as a receptacle for, or tangible representative of, digital content. Tokens then give access to the associated content [26]. One of our demo application illustrates how TOUCHTOKENS can be used for access control. Tokens can be used, e.g., to launch applications whose icons are otherwise invisible or disabled on the tablet's homescreen, enabling the device to be shared with family (parental control) and friends with some restrictions. Access to private content can be made even more secure by requiring that the token be put in a specific location, or that a particular gesture be performed with it. Our last application demonstrates the use of TOUCHTOKENS as digital containers. Users can reify photo albums into tokens, and add a picture to an album by holding the corresponding token above it. They can then display an album's content as a grid of thumbnails by rotating the token on the surface, or launch a slideshow by sliding it.

DISCUSSION AND FUTURE WORK

Main findings

We ran a formative experiment to investigate the possibility of recognizing individual tokens by categorizing their associated touch patterns. We were hypothesizing that differences in token shape and size might be sufficient to accurately discriminate those patterns. Our results revealed significant inter-user variability in terms of accuracy: our algorithm can recognize up to ten touch patterns with more than 90% accuracy for some users, while for other users, its accuracy falls down as soon as three or more tokens are in the set. This variability comes from two sources: 1) some users employ different grasp strategies for the same token; 2) some users employ grasp strategies for different tokens that yield very similar touch patterns.

Based on these observations, we then designed a set of six tokens featuring notches aimed at reducing this variability while remaining comfortable to grasp. A summative experiment showed that with this set of tokens, our recognizer has a minimum accuracy over all participants higher than 95% (avg. 98%), and this without any training. Augmented with notches, TOUCHTOKENS offer a low-cost, yet reliable, solution for enabling tangible interaction on multi-touch surfaces. As mentioned earlier, we make this recognizer freely available, along with vector-based templates for the tokens.

Alternative recognition strategies

Our algorithm is fast, robust, and easy to implement. It also features the best recognition rate among all alternatives that we implemented and tested on the data collected during our formative study. Alternative approaches we considered led to significantly poorer performance. In particular, we tested k-Nearest-Neighbour (k=1 and k=3) and SVM algorithms, using both raw data and describing features. The raw data was pre-processed to make it independent from rotation angle and finger identification. The describing features we considered included the touch envelope's area, as well as various descriptive statistics (min, max, mean, median and standard deviation) for measures such as point-centroid distance, distance between successive points, distance between any pair of points, etc. These machine learning approaches yielded recognition rates ranging from 50% to 85% per participant. Compared to this, the analytical approach detailed in this paper, which consists in aligning touch patterns using their centroid as a reference point, works much better. We also considered using as a reference point the center of the best-fit circle (i.e. the circle that passes through three touch points while minimizing the distance to all remaining points) rather than the centroid, but results were slightly worse. The recognition rate was lowered by about 3% on average.

Future work

After this first investigation, we plan to study more systematically the limits of our approach, to see how it can scale to larger sets of tokens and/or to tokens that have varying geometries. We want to better characterize the minimal difference between two touch patterns, in order to be able to *automatically* position notches that meet our requirements: create tokens that are comfortable to grasp and that our algorithm can recognize with a high level of accuracy.

We also plan to conduct a fine-grained analysis of the fingers' traces on the surface at the precise moment they are lifted off. We want to investigate if these traces provide enough data to find out whether the token is still on the surface or not. Indeed, when lifting her fingers off the surface, the user might leave the token on it, or she might remove it. This would allow us to support additional interactions, such as when placing multiple tokens on the tactile surface to express, e.g., layout and alignment constraints [20] or advanced database queries with networks of tokens [29].

REFERENCES

1. Leonardo Angelini, Denis Lalanne, Elise van den Hoven, Ali Mazalek, Omar Abou Khaled, and Elena Mugellini. 2015. Tangible Meets Gestural: Comparing and Blending Post-WIMP Interaction Paradigms. In *Proceedings of the 9th International Conference on Tangible, Embedded, and Embodied Interaction (TEI '15)*. ACM, 473–476. DOI: http://dx.doi.org/10.1145/2677199.2683583
2. Gilles Bailly, Jörg Müller, and Eric Lecolinet. 2012. Design and Evaluation of Finger-count Interaction: Combining Multitouch Gestures and Menus. *Int. J. Hum.-Comput. Stud.* 70, 10 (Oct. 2012), 673–689. DOI: http://dx.doi.org/10.1016/j.ijhcs.2012.05.006
3. Tom Bartindale and Chris Harrison. 2009. Stacks on the Surface: Resolving Physical Order Using Fiducial Markers with Structured Transparency. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces (ITS '09)*. ACM, 57–60. DOI: http://dx.doi.org/10.1145/1731903.1731916
4. Tom Bartindale, Chris Harrison, Patrick Olivier, and Scott E. Hudson. 2011. SurfaceMouse: Supplementing Multi-touch Interaction with a Virtual Mouse. In *Proceedings of the 5th International Conference on Tangible, Embedded, and Embodied Interaction (TEI '11)*. ACM, 293–296. DOI: http://dx.doi.org/10.1145/1935701.1935767
5. Patrick Baudisch, Torsten Becker, and Frederik Rudeck. 2010. Lumino: Tangible Blocks for Tabletop Computers Based on Glass Fiber Bundles. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, 1165–1174. DOI: http://dx.doi.org/10.1145/1753326.1753500
6. Hrvoje Benko, T. Scott Saponas, Dan Morris, and Desney Tan. 2009. Enhancing Input on and Above the Interactive Surface with Muscle Sensing. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces (ITS '09)*. ACM, 93–100. DOI: http://dx.doi.org/10.1145/1731903.1731924
7. Hrvoje Benko, Andrew D Wilson, and Patrick Baudisch. 2006. Precise Selection Techniques for Multi-Touch Screens. In *Proceedings of the SIGCHI conference on Human Factors in computing systems (CHI '06)*. 1263–1272. DOI: http://dx.doi.org/10.1145/1124772.1124963
8. Andrea Bianchi and Ian Oakley. 2013. Designing Tangible Magnetic Appcessories. In *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction (TEI '13)*. ACM, 255–258. DOI: http://dx.doi.org/10.1145/2460625.2460667
9. Andrea Bianchi and Ian Oakley. 2015. MagnID: Tracking Multiple Magnetic Tokens. In *Proceedings of the 9th International Conference on Tangible, Embedded, and Embodied Interaction (TEI '15)*. ACM, 61–68. DOI: http://dx.doi.org/10.1145/2677199.2680582
10. Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. 1993. Toolglass and Magic Lenses: The See-through Interface. In *Proceedings of the 20th Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '93)*. ACM, 73–80. DOI: http://dx.doi.org/10.1145/166117.166126
11. Rachel Blagojevic and Beryl Plimmer. 2013. CapTUI: Geometric Drawing with Tangibles on a Capacitive Multi-touch Display. In *Proceedings of INTERACT '13*. Springer, 511–528.
12. Sebastian Boring, David Ledo, Xiang 'Anthony' Chen, Nicolai Marquardt, Anthony Tang, and Saul Greenberg. 2012. The fat thumb: using the thumb's contact size for single-handed mobile interaction. In *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services companion (MobileHCI '12)*. ACM, 207–208. DOI: http://dx.doi.org/10.1145/2371664.2371711
13. Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (Dec. 2011), 2301–2309. DOI: http://dx.doi.org/10.1109/TVCG.2011.185
14. Bryan Buchholz, Thomas J Armstrong, and Steven A Goldstein. 1992. Anthropometric data for describing the kinematics of the human hand. *Ergonomics* 35, 3 (1992), 261–273.
15. Wolfgang Büschel, Ulrike Kister, Mathias Frisch, and Raimund Dachselt. 2014. T4 - Transparent and Translucent Tangibles on Tabletops. In *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI '14)*. ACM, 81–88. DOI: http://dx.doi.org/10.1145/2598153.2598179
16. Xiang Cao, A.D. Wilson, R. Balakrishnan, K. Hinckley, and S.E. Hudson. 2008. ShapeTouch: Leveraging contact shape on interactive surfaces. In *Proceedings of the 3rd International Workshop on Horizontal Interactive Human Computer Systems (TABLETOP '08)*. IEEE, 129–136. DOI: http://dx.doi.org/10.1109/TABLETOP.2008.4660195
17. Liwei Chan, Stefanie Müller, Anne Roudaut, and Patrick Baudisch. 2012. CapStones and ZebraWidgets: Sensing Stacks of Building Blocks, Dials and Sliders on Capacitive Touch Screens. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, 2189–2192. DOI: http://dx.doi.org/10.1145/2207676.2208371
18. George W. Fitzmaurice, Hiroshi Ishii, and William A. S. Buxton. 1995. Bricks: Laying the Foundations for Graspable User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '95)*. ACM Press/Addison-Wesley Publishing Co., 442–449. DOI: http://dx.doi.org/10.1145/223904.223964

19. Dustin Freeman, Hrvoje Benko, Meredith Ringel Morris, and Daniel Wigdor. 2009. ShadowGuides: Visualizations for In-situ Learning of Multi-touch and Whole-hand Gestures. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces (ITS '09)*. ACM, 165–172. DOI: http://dx.doi.org/10.1145/1731903.1731935

20. Mathias Frisch, Sebastian Kleinau, Ricardo Langner, and Raimund Dachselt. 2011. Grids & Guides: Multi-touch Layout and Alignment Tools. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, 1615–1618. DOI: http://dx.doi.org/10.1145/1978942.1979177

21. Emilien Ghomi, Stéphane Huot, Olivier Bau, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2013. Arpège: Learning Multitouch Chord Gestures Vocabularies. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces (ITS '13)*. ACM, 209–218. DOI: http://dx.doi.org/10.1145/2512349.2512795

22. Chris Harrison, Julia Schwarz, and Scott E. Hudson. 2011. TapSense: Enhancing Finger Interaction on Touch Surfaces. In *Proceedings of the 24th ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, 627–636. DOI: http://dx.doi.org/10.1145/2047196.2047279

23. Chris Harrison, Robert Xiao, Julia Schwarz, and Scott E. Hudson. 2014. TouchTools: Leveraging Familiarity and Skill with Physical Tools to Augment Touch Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, 2913–2916. DOI: http://dx.doi.org/10.1145/2556288.2557012

24. Björn Hartmann, Meredith Ringel Morris, Hrvoje Benko, and Andrew D. Wilson. 2009. Augmenting Interactive Tables with Mice & Keyboards. In *Proceedings of the 22nd ACM Symposium on User Interface Software and Technology (UIST '09)*. ACM, 149–152. DOI: http://dx.doi.org/10.1145/1622176.1622204

25. Seongkook Heo and Geehyuk Lee. 2011. Force Gestures: Augmenting Touch Screen Gestures with Normal and Tangential Forces. In *Proceedings of the 24th ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, 621–626. DOI: http://dx.doi.org/10.1145/2047196.2047278

26. Lars Erik Holmquist, Johan Redström, and Peter Ljungstrand. 1999. Token-Based Access to Digital Information. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC '99)*. Springer-Verlag, 234–245. http://dl.acm.org/citation.cfm?id=647985.743869

27. Christian Holz and Patrick Baudisch. 2013. Fiberio: A Touchscreen That Senses Fingerprints. In *Proceedings of the 26th ACM Symposium on User Interface Software and Technology (UIST '13)*. ACM, 41–50. DOI: http://dx.doi.org/10.1145/2501988.2502021

28. Yvonne Jansen, Pierre Dragicevic, and Jean-Daniel Fekete. 2012. Tangible Remote Controllers for Wall-size Displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, 2865–2874. DOI: http://dx.doi.org/10.1145/2207676.2208691

29. Hans-Christian Jetter, Jens Gerken, Michael Zöllner, Harald Reiterer, and Natasa Milic-Frayling. 2011. Materializing the Query with Facet-streams: A Hybrid Surface for Collaborative Search on Tabletops. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, 3013–3022. DOI: http://dx.doi.org/10.1145/1978942.1979390

30. Sergi Jordà, Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner. 2007. The reacTable: Exploring the Synergy Between Live Music Performance and Tabletop Tangible Interfaces. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction (TEI '07)*. ACM, 139–146. DOI: http://dx.doi.org/10.1145/1226969.1226998

31. Martin Kaltenbrunner and Ross Bencina. 2007. reacTIVision: A Computer-vision Framework for Table-based Tangible Interaction. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction (TEI '07)*. ACM, 69–74. DOI: http://dx.doi.org/10.1145/1226969.1226983

32. Hideki Koike, Wataru Nishikawa, and Kentaro Fukuchi. 2009. Transparent 2-D Markers on an LCD Tabletop System. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, 163–172. DOI: http://dx.doi.org/10.1145/1518701.1518728

33. Sven Kratz, Tilo Westermann, Michael Rohs, and Georg Essl. 2011. CapWidgets: Tangible Widgets Versus Multi-touch Controls on Mobile Devices. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems (CHI EA '11)*. ACM, 1351–1356. DOI: http://dx.doi.org/10.1145/1979742.1979773

34. Rong-Hao Liang, Kai-Yin Cheng, Liwei Chan, Chuan-Xhyuan Peng, Mike Y. Chen, Rung-Huei Liang, De-Nian Yang, and Bing-Yu Chen. 2013. GaussBits: Magnetic Tangible Bits for Portable and Occlusion-free Near-surface Interactions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, 1391–1400. DOI: http://dx.doi.org/10.1145/2470654.2466185

35. Rong-Hao Liang, Han-Chih Kuo, Liwei Chan, De-Nian Yang, and Bing-Yu Chen. 2014. GaussStones: Shielded Magnetic Tangibles for Multi-token Interactions on Portable Displays. In *Proceedings of the 27th ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, 365–372. DOI: http://dx.doi.org/10.1145/2642918.2647384

36. Jack M Loomis and Susan J Lederman. 1986. Tactual perception. In *Handbook of Perception and Human Performance*, K. R. Boff, L. Kaufman, and J. P. Thomas (Eds.). John Wiley & Sons.

37. Halla Olafsdottir and Caroline Appert. 2014. Multi-touch Gestures for Discrete and Continuous Control. In *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI '14)*. ACM, 177–184. DOI: http://dx.doi.org/10.1145/2598153.2598169

38. Halla Olafsdottir, Theophanis Tsandilas, and Caroline Appert. 2014. Prospective Motor Control on Tabletops: Planning Grasp for Multitouch Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, 2893–2902. DOI: http://dx.doi.org/10.1145/2556288.2557029

39. Brandon Paulson, Danielle Cummings, and Tracy Hammond. 2011. Object Interaction Detection Using Hand Posture Cues in an Office Setting. *Int. J. Hum.-Comput. Stud.* 69, 1-2 (Jan. 2011), 19–29. DOI: http://dx.doi.org/10.1016/j.ijhcs.2010.09.003

40. Jun Rekimoto. 2002. SmartSkin: An Infrastructure for Freehand Manipulation on Interactive Surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '02)*. ACM, 113–120. DOI: http://dx.doi.org/10.1145/503376.503397

41. Jun Rekimoto, Brygg Ullmer, and Haruo Oba. 2001. DataTiles: A Modular Platform for Mixed Physical and Graphical Interactions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '01)*. ACM, 269–276. DOI: http://dx.doi.org/10.1145/365024.365115

42. Marco Santello and John F Soechting. 1998. Gradual molding of the hand to object contours. *Journal of neurophysiology* 79, 3 (1998), 1307–1320.

43. Atsushi Sugiura and Yoshiyuki Koseki. 1998. A User Interface Using Fingerprint Recognition: Holding Commands and Data Objects on Fingers. In *Proceedings of the 11th ACM Symposium on User Interface Software and Technology (UIST '98)*. ACM, 71–79. DOI: http://dx.doi.org/10.1145/288392.288575

44. Brygg Ullmer, Hiroshi Ishii, and Dylan Glas. 1998. mediaBlocks: Physical Containers, Transports, and Controls for Online Media. In *Proceedings of the 25th Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. ACM, 379–386. DOI: http://dx.doi.org/10.1145/280814.280940

45. Brygg Ullmer, Hiroshi Ishii, and Robert J. K. Jacob. 2003. Tangible Query Interfaces: Physically Constrained Tokens for Manipulating Database Queries. In *Proceedings of INTERACT '03*. 279–286.

46. Consuelo Valdes, Diana Eastman, Casey Grote, Shantanu Thatte, Orit Shaer, Ali Mazalek, Brygg Ullmer, and Miriam K. Konkel. 2014. Exploring the Design Space of Gestural Interaction with Active Tokens Through User-defined Gestures. In *Proceedings of the SIGCHI conference on Human Factors in computing systems (CHI '14)*. ACM, 4107–4116. DOI: http://dx.doi.org/10.1145/2556288.2557373

47. Radu-Daniel Vatavu and Ionu Alexandru Zaii. 2013. Automatic Recognition of Object Size and Shape via User-dependent Measurements of the Grasping Hand. *Int. J. Hum.-Comput. Stud.* 71, 5 (May 2013), 590–607. DOI: http://dx.doi.org/10.1016/j.ijhcs.2013.01.002

48. Simon Voelker, Kosuke Nakajima, Christian Thoresen, Yuichi Itoh, Kjell Ivar Øvergård, and Jan Borchers. 2013. PUCs: Detecting Transparent, Passive Untouched Capacitive Widgets on Unmodified Multi-touch Displays. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces (ITS '13)*. ACM, 101–104. DOI: http://dx.doi.org/10.1145/2512349.2512791

49. Daniel Vogel and Géry Casiez. 2011. Conté: Multimodal Input Inspired by an Artist's Crayon. In *Proceedings of the 24th ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, 357–366. DOI: http://dx.doi.org/10.1145/2047196.2047242

50. Malte Weiss, Julie Wagner, Yvonne Jansen, Roger Jennings, Ramsin Khoshabeh, James D. Hollan, and Jan Borchers. 2009. SLAP Widgets: Bridging the Gap Between Virtual and Physical Controls on Tabletops. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, 481–490. DOI: http://dx.doi.org/10.1145/1518701.1518779

51. Alexander Wiethoff, Hanna Schneider, Michael Rohs, Andreas Butz, and Saul Greenberg. 2012. Sketch-a-TUI: Low Cost Prototyping of Tangible Interactions Using Cardboard and Conductive Ink. In *Proceedings of the 6th International Conference on Tangible, Embedded and Embodied Interaction (TEI '12)*. ACM, 309–312. DOI: http://dx.doi.org/10.1145/2148131.2148196

52. Daniel Wigdor, Hrvoje Benko, John Pella, Jarrod Lombardo, and Sarah Williams. 2011. Rock & Rails: Extending Multi-touch Interactions with Shape Gestures to Enable Precise Spatial Manipulations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, 1581–1590. DOI: http://dx.doi.org/10.1145/1978942.1979173

53. Mike Wu and Ravin Balakrishnan. 2003. Multi-finger and Whole Hand Gestural Interaction Techniques for Multi-user Tabletop Displays. In *Proceedings of the 16th ACM Symposium on User Interface Software and Technology (UIST '03)*. ACM, 193–202. DOI: http://dx.doi.org/10.1145/964696.964718

54. Mike Wu, Chia Shen, Kathy Ryall, Clifton Forlines, and Ravin Balakrishnan. 2006. Gesture Registration, Relaxation, and Reuse for Multi-Point Direct-Touch Surfaces. In *Proceedings of the 1st International Workshop on Horizontal Interactive Human-Computer Systems (TABLETOP '06)*. IEEE, 185–192. DOI: http://dx.doi.org/10.1109/TABLETOP.2006.19

55. Neng-Hao Yu, Sung-Sheng Tsai, I-Chun Hsiao, Dian-Je Tsai, Meng-Han Lee, Mike Y. Chen, and Yi-Ping Hung. 2011. Clip-on Gadgets: Expanding Multi-touch Interaction Area with Unpowered Tactile Controls. In *Proceedings of the 24th ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, 367–372. DOI: http://dx.doi.org/10.1145/2047196.2047243

56. Shumin Zhai, Per Ola Kristensson, Caroline Appert, Tue Haste Andersen, and Xiang Cao. 2012. Foundational Issues in Touch-Surface Stroke Gesture Design: An Integrative Review. *Found. Trends Hum.-Comput. Interact.* 5, 2 (Feb. 2012), 97–205. DOI: http://dx.doi.org/10.1561/1100000012