'Designeering Interaction': A Missing Link in the Evolution of Human-Computer Interaction

Stéphane Huot

UNIVERSITÉ PARIS-SUD

HABILITATION À DIRIGER DES RECHERCHES

présentée par

Stéphane HUOT

Specialité: Informatique – Interaction Homme-Machine

'Designeering Interaction': A Missing Link in the Evolution of Human-Computer Interaction

7 Mai 2013

Rapporteurs :

M. Saul GREENBERG	Professeur, University of Calgary
M. Robert J.K. JACOB	Professeur, Tufts University
Mme Laurence NIGAY	Professeur, Université Joseph Fourier & Institut Universitaire de France

Examinateurs :

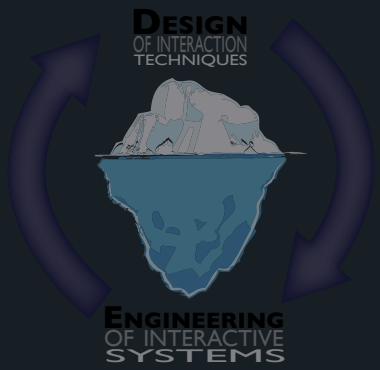
M. Michel BEAUDOUIN-LAFON	Professeur, Université Paris-Sud & Institut Universitaire de France
M. Jan BORCHERS	Professeur, RWTH Aachen University
M. Alain DENISE	Professeur, Université Paris-Sud
Mme Wendy E. MACKAY	Directeur de Recherche, Inria
M. Ted SELKER	Associate Director – CyLab Mobility Research Center, Carneggie Mellon University – Silicon Valley Campus

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

DESIGNEERING INTERACTION

A Missing Link in the Evolution of Human-Computer Interaction

Stéphane Huot
Université Paris-Sud
Laboratoire de Recherche en Informatique
Inria - in|situ| group



Dissertation submitted in fulfillment of the requirements for the `Habilitation à Diriger des Recherches' – May 2013

To R and JJ

ABSTRACT

Human Computer Interaction (HCI) is a fascinating research field because of its multidisciplinary nature, combining such diverse research domains as *design*, *human factors* and *computer science* as well as a variety of methods including *empirical* and *theoretical* research. HCI is also fascinating because it is still young and so much is left to discover, invent and understand. The evolution of computers, and more generally of interactive systems, is not frozen, and so are the ways in which we interact with them. From desktop computers, to mobile devices, to large displays or multi-surface environments, technology extends the possibles, needs initiate technologies, and HCI is thus a constantly moving field. The variety of challenges to address, as well as their underlying combinations of sub-domains (design, computer science, experimental psychology, sociology, etc.), imply that we should also adapt, question and sometimes reinvent our research methods and processes, pushing the limits of HCI research further.

Since I entered the field 12 years ago, my research activities have essentially revolved around two main themes: the design, implementation and evaluation of novel interaction techniques (on desktop computers, mobile devices and multi-surface environments) and the engineering of interactive systems (models and toolkits for advanced input and interaction). Over time, I realized that I had entered a loop between these two concerns, going back and forth between designing and evaluating new interaction techniques, and defining and implementing new software architectures or toolkits. I observed that they strongly influence each other: The design of interaction techniques informs on the capabilities and limitations of the platform and the software being used, and new architectures and software tools open the way to new designs and possibilities.

Through the discussion of several of my research contributions in these fields, this document investigates how interaction design challenges technology, and how technology – or engineering of interactive systems – could support and unleash interaction design. These observations will lead to a first definition of the "*Designeering Interaction*" conceptual framework that encompasses the specificities of these two fields and builds a bridge between them, paving the way to new research perspectives. In particular, I will discuss which types of tools, from the system level to the end user, should be designed, implemented and studied in order to better support interaction design along the evolution of interactive systems. At a more general level, Designeering Interaction is also a contribution that, I hope, will help better "understand how HCI works with technology".

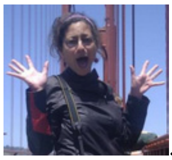
Keywords: Human-Computer Interaction, Interaction Techniques, Engineering of Interactive Systems, Mobile Interaction, Multi-Surface Environments, Toolkits, Designeering Interaction

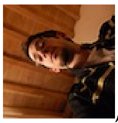
ACKNOWLEDGMENTS

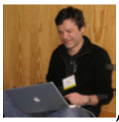
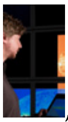
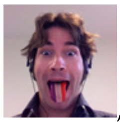
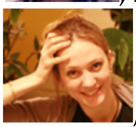
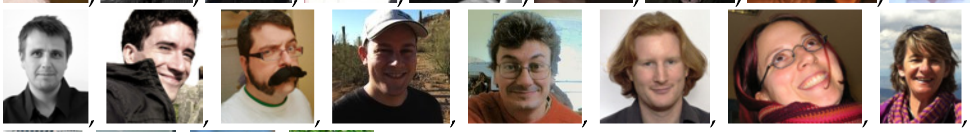
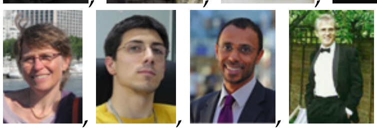
 ,  and  .

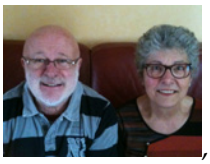
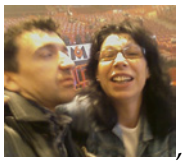
and also  ,  ,  ,  and  .

 and in|situ| .

specials:  ,  ,  ,  and  .

students:  (again),  ,  ,  .

and “en vrac”:  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ...

 ,  ,  .

et

This habilitation concludes two years of leave for research which would not have been possible without the support of the IUT d'Orsay and Inria Saclay–Île-de-france.

Note: this page is best viewed with Adobe Acrobat...

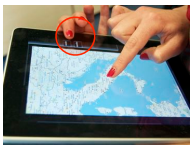
CONTENTS

CONVENTIONS

References to the publications that I have co-authored are displayed in bold: **[WHM12]**.

Hyperlinks to content within this document (sections, pages, acronyms definitions, etc.) are displayed in blue: see the table of contents on page xi.

Hyperlinks to external content are displayed in red: http://insitu.lri.fr/~huot.



↳*Text and pictures in margins are to help the reader to quickly identify the content of the nearby paragraph(s)*

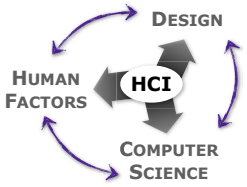
"A vision, Division
Revision, Recognate
An action, A reaction
Distraction, Question the fate."

Dave Grohl – Erase/Replace (2007).

1

INTRODUCTION

Human Computer Interaction (HCI) is a fascinating research field. Fascinating because of its multidisciplinary nature, combining such diverse research domains as *design*, *human factors* and *computer science* as well as a variety of methods including *empirical* (qualitative, quantitative) and *theoretical* research. But HCI is also fascinating because it is still young and so much is left to discover, invent and understand. The field emerged in the early 60's with pioneers such as Douglas Engelbart, whose research group invented the mouse and many other now-standard concepts of user interfaces in NLS/Augment[1] [Eng62; Eng88], and Ivan Sutherland's SketchPad [Sut63], which pioneered many concepts of direct manipulation still present in today's interfaces. HCI gained interest in the early and mid 70's with, for example, the first vision of a notebook computer (Alan Kay's Dynabook) or the first personal computer (the Alto and its Graphical User Interface (GUI) at Xerox PARC), to eventually become a major research field with the creation of the Association for Computing Machinery (ACM) Special Interest Group on Computer-Human Interaction (ACM SIGCHI) together with its flagship ACM SIGCHI Conference on Human Factors in Computing Systems (CHI) in 1982.



↳*HCI is a multidisciplinary field*

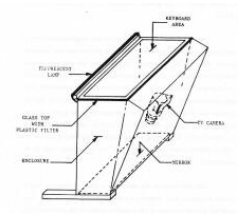
At that time, I was just starting to be a "passive" observer of the rapid evolution of computer interfaces and interaction techniques, as a ten-year old user and programmer of pocket calculators and personal computers. I was fascinated by computer graphics and interaction programming: my addiction to video games was undoubtedly pushing me to look under the hood and to program my own games. I was programming in Basic on my Amstrad computer,

1 The 90 minutes live demonstration of the NLS system in 1968 is known as "the mother of all demos". See http://www.dougengelbart.org/firsts/dougs-1968-demo.html.

developing interaction techniques without even knowing what "Interaction" was. But I quickly realized how much these platforms and language were indeed limiting my creative possibilities. Many years later, this frustration would continue after switching to more structured languages and Application Programming Interfaces (APIs) – e. g. Delphi, Allegro/C++, Java AWT & Swing – as using call-back procedures for programming interaction has its drawbacks in terms of descriptive power and maintainability [ABo8; Mye91]. This probably strongly influenced my interest in looking for new ways of describing and programming interaction, as I will explain in this manuscript.

1.1 MANY THINGS HAVE BEEN DONE

Despite its recency, HCI has had an impact on many other research fields, but also on the technology industry and on society in general. MYERS already observed the impact of academic research in HCI technology, the Computer Science side of the field, 15 years ago [Mye98]: the mouse, GUI and windows-based applications, hypertext, gesture recognition, Augmented Reality (AR) and Virtual Reality (VR), but also Interface Builders or even component architectures, are all inventions coming from academic HCI research labs. And since MYERS' paper, many other "HCI-technology research" related products have emerged and have been widely adopted: multi-touch devices (invented in the 8o's at the University of Toronto, along with the now-standard pinch and flick gestures), motion-based input devices, etc.



↳*1982, invention of multi-touch [Meh82]*

HCI is of course not the only scientific domain whose results have been widely adopted, and conversely, most of these innovations cannot be credited only to progress in HCI. The most visionary concepts were heavily dependent on progress in other technologies, in order to become feasible on a large scale. But it should be noted that a large part of the development and acceptance of these technologies in everyday life is due to HCI: it is the visible part, the usable and used part, the "interface" between computer science and the world, between the machines and their users.

1.2 MANY THINGS REMAIN TO BE DONE

Some early visions have already been realized thanks to technological progress, such as Alan KAY's Dynabook, which is now incarnated into a variety of mobile computers (laptops, notebooks, tablets, smartphones). But some are still only partially achieved, such as Mark WEISER's "Ubiquitous Computing" [Wei91]: while the actual technology now enables the miniaturization, inter-connection and "disappearance" of actual computing devices, "true" ubiquitous computing is still challenging the field in terms of usage and interaction. This is another fascinating aspect of HCI research: the youth of the field and its rapid progresses makes it possible to shape, embrace and, with some luck, make such visionary concepts become true.

HCI, however, faces many challenges, because of the evolution of interactive systems technologies, economic and marketing concerns, specific users needs and more generally changes in our society, all intertwined in a mutual loop. An interesting example is, again, multi-touch systems and interactions that were first prototyped in the 8o's by researchers from the University of Toronto. Bill BUXTON gives a good overview and discussion about the evolution and explosion of multi-touch on his website [Bux07a]. More than simply tracing and commenting on the history of multi-touch, he also highlights the need for the right and complex

context in order to turn a "great invention" into a "real innovation". This is what BUXTON calls "the long nose of innovation" [Bux08], where any invention needs (sometime long) refinements and a final "traction" before being turned into an innovative product.

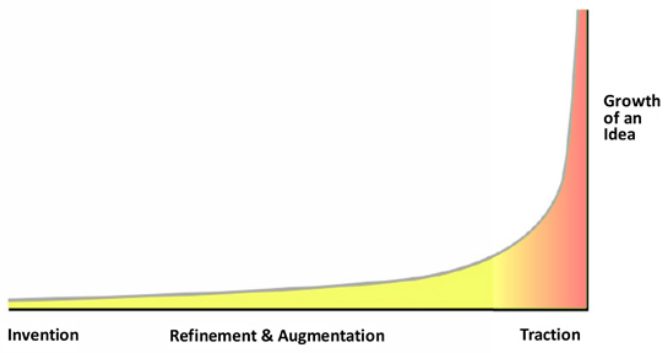


Figure 1.1: The Long Nose of Innovation [Bux08] © Bill Buxton.

Multi-touch technologies, despite being well-known in HCI research, clearly became "en vogue" thanks to the whole new ecosystem that manufacturers created around their multi-touch devices, such as the Apple's iPhone in 2007. The Microsoft Surface (the former multi-touch tabletop, not the 2012 tablet device) appeared at the same period, but with less success than the mobile device. This is an interesting case, demonstrating the impact of the ecosystem over usefulness or usability: mobile devices are far less adapted to multi-touch input than a tabletop surface since they are used most of the time while moving, with only one hand [KBC08]. Then tablets were introduced (again by Apple) and we now have four main kinds of multi-touch devices: screen, table, smartphone and tablet. This commercial success has triggered a regain of interest for HCI research in multi-touch technologies that the field invented... 30 years ago. Indeed, since the release of the iPhone, we observe a significant increase in publications about multi-touch technologies and interaction techniques, whether in mobile conditions (see the proceedings of the ACM MobileHCI conference) or on dedicated hardware (see the recently created ITS conference). Technology and marketing made multi-touch devices now available in the wild and with/for real users, raising new challenges that were not anticipated before, when those devices were still expensive laboratory prototypes with few, if any, real use cases. This, of course, does not mean that we should stop investigating novel inventions, but that it is different research than studying them when they become off-the-shelf appliances.

To me, this is another fascinating aspect of HCI. Even if we, as researchers, do not master all the parameters of what will make a technology available or popular, the matter of fact is that the evolution of computers, and more generally of interactive systems, is not frozen, and so are the ways in which we interact with them. From desktop computers, to mobile devices, to large displays or multi-surface environments, technology extends the possibles (e. g. high-speed networks allow remote collaboration), needs initiate technologies (e. g. big data requires large displays), and HCI is thus a constantly moving field.

1.3 STILL NEED TO MAKE THESE THINGS POSSIBLE

Finally, HCI is also fascinating at an epistemological level, by being both a young and a multidisciplinary research field. The variety of challenges to address, as well as their underlying combinations of sub-domains (design, computer science, experimental psychology, sociology, etc.), imply that we should also adapt, question and sometimes reinvent our research methods and processes, pushing the limits of HCI research further. Again, this is not specific to HCI since reflecting on research methods, criticizing or revising them is a long lasting scientific tradition. However, unlike, e. g., biology or mathematics that address the understanding of "natural phenomenons" in a well established way since centuries now, HCI studies the artifacts that it designs. We have to rely on design methods to create them, on computer science principles to implement them, on experimental psychology or sociology protocols to evaluate them, and we should ensure that all these methods are working well together, in order to operationalize higher-level concepts that are specific to HCI. As observed and formalized by MACKAY and FAYARD, by "*triangulating across the scientific and design disciplines that compose HCI, we can increase the validity of our research results and [...] make them more useful to real-world problems.*" [MF97].

↳$MT = a + b.\log_2(\frac{D}{W} + 1)$
FITTS' *law as a theoretical tool for interaction design*

For instance, theoretical models can provide guidelines for designing and exploring new interaction techniques, but evaluation methods can also help assessing the feasibility and usability of these new designs. A good example is FITTS' law, which models the time needed to acquire a target according to the difficulty of the task (size of the target and its distance from the starting point). After CARD et al. confirmed that pointing with a mouse is modeled by FITTS' law [CEB78], it quickly became an essential tool in HCI [Mac92]. It has been an instrumental source of inspiration for designing new pointing techniques, by suggesting ways of reducing difficulty and increase performance. But it also provides HCI researchers with a tool to evaluate and compare pointing techniques, by grounding and defining an adapted experimental protocol. As we will see later, this is the approach we followed for designing and evaluating the TorusDekstop (section 2.1.1) and TapTap (section 2.2.1) interaction techniques.

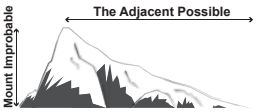
Another example is how CARD et al. generalized this link between computer science and human factors in their seminal book "The Psychology of Human-Computer Interaction" [CMN83]. CARD et al. also proposed to apply a "Morphological Design Space Analysis" to the study of input devices, an approach previously used in engineering and by BERTIN for his study of the semiology of graphics [CMR91]. More than describing a design space, the morphological analysis allows to reflect on the design space itself and to generate further designs. This is the approach that we latter applied in our work on menu systems [NBH09] and to body-centric interaction for multi-surface environments [Wag+13].

At the CHI'12 conference, Stuart CARD gave a very inspiring lecture specifically addressing this problem of "*how to ground the field, accelerate its progress, and make it cumulative by fashioning theories and incorporating them into practice*" and attempting "*to sketch out, in the spirit of the times, what an interaction science for HCI could look like, how it might be incorporated into practice, and how it might be taught*". CARD highlighted the fact that technology develops by combinatoric evolution and that we need to understand how HCI works with technology, and that we need to focus on technological and theoretical progress all together in order to successfully take on the new "golden age" that HCI is entering now. This was for me one of the initial sparks for the framing of my own research work and of my vision of HCI in

general, which led to the concept of "*Designeering Interaction*" that I introduce in this manuscript.

"Designeering Interaction" is a portmanteau word that I first coined in order to describe my two main research interests in HCI: "Design of Interaction Techniques" and "Engineering of Interactive Systems". Over time, I realized that I had entered a loop between these two concerns, going back and forth between designing and evaluating new interaction techniques, and defining and implementing new software architectures or toolkits. I observed that they strongly influence each other: The design of interaction techniques informs on the capabilities and limitations of the platform and the software being used, giving insights into what could be done to improve them. On the other hand, new architectures and software tools open the way to new designs and possibilities, by giving the necessary bricks to build with. This is a slow and gradual process, where things cannot be built until the right technology is available and where the most innovative ideas will push the technology further (examples include KAY's and WEISER's previously mentioned visions).

This situation is quite similar to the evolution of species, which was well described by Richard DAWKINS in his book "Climbing Mount Improbable": "*There can be no sudden leaps upward – no precipitous increases in ordered complexity. Second, there can be no going downhill – species can't get worse as a prelude to getting better. Third, there may be more than one peak – more than one way of solving the same problem, all flourishing in the world*" [Daw97, pp. 1330-1332]. Among others, DAWKINS defends the idea that even the more improbable evolutionary step might occur, but that it is impossible to directly jump several levels of complexity at once (climbing the mountain via its steep cliff, as shown in the side figure). While DAWKINS considers the evolution of "designoids" objects, i. e. "*living bodies and their products*", Steven JOHNSON makes a similar and complementary observation concerning designed artifacts and innovation, drawing on the "Adjacent Possible" theory of Stuart KAUFFMAN: "*The trick to having good ideas is not to sit around in glorious isolation and try to think big thoughts. The trick is to get more parts on the table.*" [Joh10, p. 42]. These parts define the adjacent possible, the set of what could be designed, through a process similar to the evolution of designoids, but by assembling the parts in new ways.



Similarly, in HCI, technology – hardware and software – and knowledge – experience and theories – are the spare parts of the adjacent possible – the new techniques or systems that could be designed. For example, the *Nintendo Wii Remote* is a "first-order" combination of infrared LEDs, accelerometers, buttons and bluetooth wireless technology, which was obviously in the adjacent possible when it was designed. Nevertheless, this cheap motion sensing device extended the adjacent possible, leading to a radical change in the gaming industry [NR12], and was in turn used as a part for new combinations [Lee08]. But an idea or an invention that lies outside of the adjacent possible cannot be designed by simply "climbing the mountain via the steep cliff". The necessary technological evolutions that will make it possible should be addressed first, as for example the Dynabook which led to many side-projects and innovations that it was depending on, but was never totally realized. This results in a slow and uncertain evolution process, which helps to explore and fill a number of gaps in our research field, but can also lead to wrong ways and deadlocks. A better analysis and framing of this process, i. e. examining the adjacent possible of HCI technology and methods as well as the ways to extend it, would certainly enrich our field and uncover new possibilities for interaction design.

Inspired by these evolutionary notions, I will introduce the concept of "Designeering Interaction", as a framework to discuss how the design of novel interaction techniques and the engineering of interactive systems influence each other. Through the discussion of several of my research projects and research contributions in these fields, I will investigate how *interaction design challenges technology*, and how *technology – or engineering of interactive systems – could support and unleash interaction design*. These observations will lead to the Designeering Interaction conceptual framework which, by accounting for the multidisciplinary nature of the young HCI, encompasses the specificities of these two fields and builds a bridge between them, paving the way to new research perspectives. In particular, I will discuss which kind of tools, from the system level to the end user, should be designed, implemented and studied in order to better support interaction design along the evolution of interactive systems. At a more general level, Designeering Interaction is also a contribution that, I hope, will help better "understand how HCI works with technology".

Note: I use the terms "*technology*" and "*UI technology*" loosely, to refer to the hardware and software tools used for studying and designing interaction techniques/interactive systems (I will most of the time refer to software and toolkits).

This manuscript is divided into three parts.

The **first part** is the main document, providing an overview of my research since I defended my Ph.D. in 2005. **Chapter 1** is this introduction. **Chapters 2** and **3** summarize some of my previous work in the fields of *Interaction Techniques* and *Engineering of Interactive Systems*. These are discussed as an introduction to the concept of "Designeering Interaction", which is itself developed in **chapter 4**. This last chapter concludes with future directions for research.

The **second** and **third parts** are appendices. **Appendix A** contains the full list of my publications (section A.1), as well as a selection of nine research papers that I have co-authored between 2005 and 2013 (section A.2). Finally, **appendix B** includes my curriculum vitae.

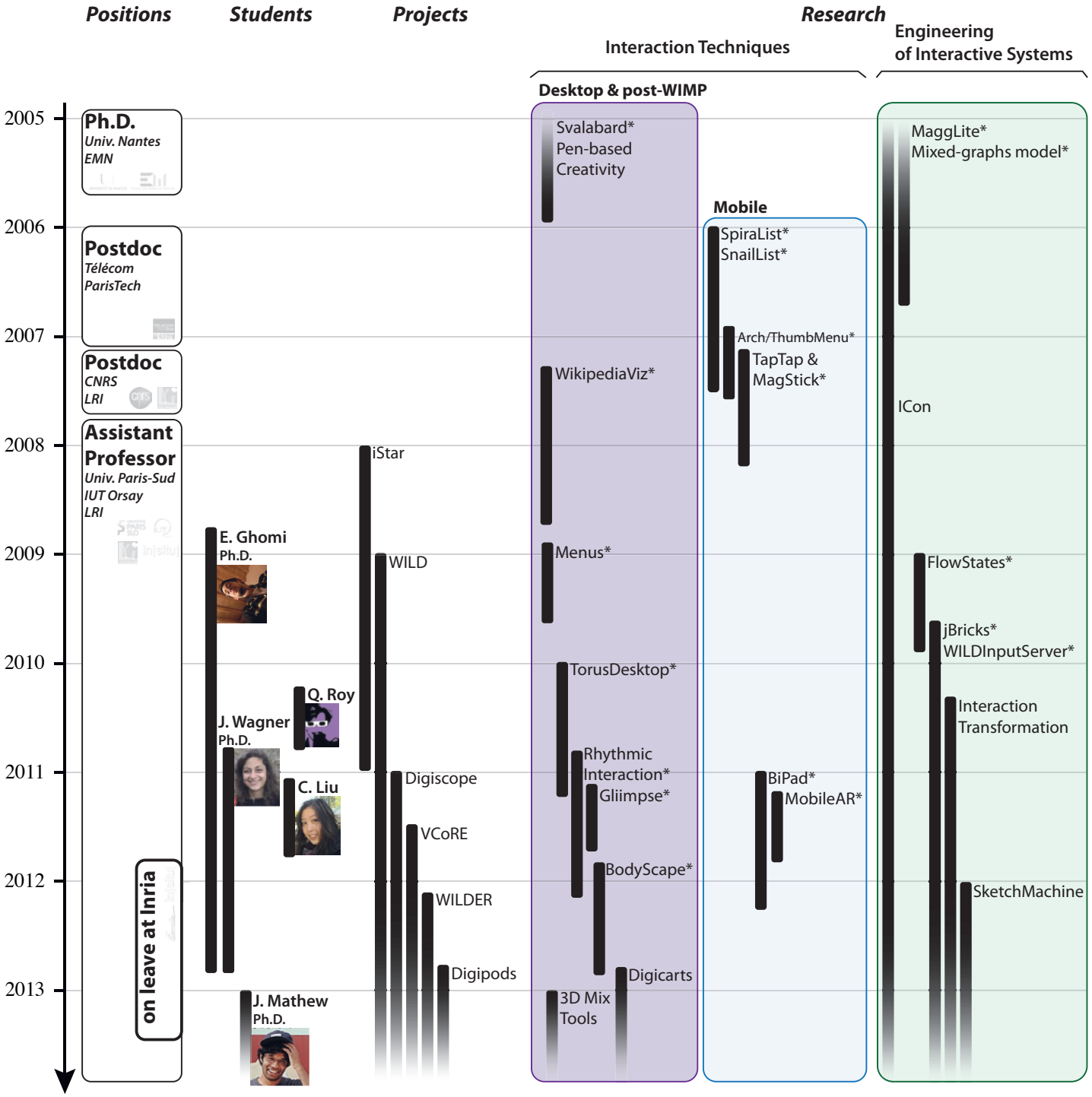


Figure 1.2: Story of my (professional) life.

"We now have interfaces that allow the use of computers for such highly interactive tasks as making engineering drawings and taking airline reservations. But despite considerable advancements, the systems we have are often ragged and in places are sufficiently poor to cripple whole ranges of use."

Stuart K. Card, Thomas P. Moran and Allen Newell – The Psychology of Human-Computer Interaction (1983) [CMN83].

2

FROM DESIGNING INTERACTION TO ENGINEERING INTERACTIVE SYSTEMS

Designing and studying novel interaction techniques – for specific needs, contexts or users – is probably the primary concern of most people working in HCI, whether being a computer scientist, designer, psychologist or practitioner. Designing interaction techniques requires drawing on multidisciplinary skills – e. g. design, prototyping, implementation –, and building upon multiple methodologies – e. g. design methods, human factors, empirical studies. In this chapter, I focus on the relationship between *Interaction Design* and *Engineering*: I demonstrate, through examples, that one needs to consider engineering problems when designing interaction techniques. The first sections report on my research work and contributions to the field of novel interaction techniques for both standard desktop and mobile devices. The last section discusses how Interaction Design challenges technology.

2.1 INTERACTION ON THE DESKTOP

Despite the new interface paradigms and forms of interaction that were introduced during the last decade, such as touch interfaces, gestural interaction or physics metaphors [AB06], the Windows Icons Menus Pointer (WIMP) computer desktop controlled with a mouse and a keyboard is a die-hard metaphor. WIMP interfaces have seen only minor changes since their early days, mostly cosmetic improvements that address only a few usability issues. They are still the most common way, if not the only one, to interact with a personal computer, and they are, in fact, well-adapted to many professional and entertainment tasks. As a result, they remain a central focus of HCI research, with a large body of work on window management and direct manipulation.

2.1.1 *Pointing*

Target acquisition is one of the most frequent tasks in interactive systems and it that has been studied extensively. However, very few, if any, of the recent improvements in pointing techniques are used in our everyday interactive systems. This could be attributed to Buxton's "Long Nose of Innovation" that I already mentioned in the introduction, but there are also strong technical reasons: most of these techniques need to be implemented at a system level (device drivers) or need access to informations, such as target locations, that is not made available by the application or the system.

As observed by Wobbrock et al., pointing facilitation techniques can be "target-aware" or "target-agnostic", depending on whether they need the system to respectively know the potential targets in advance or not [Wob+09]. According to Fitts' law, one could reduce pointing time to a given target by artificially increasing its size and/or, less frequently, reducing the distance required to reach it. Most of the recently introduced pointing techniques are built on this approach, achieving very high performance rates [Bal04; CLP09; ZDB12]. But the counterpart of this gain in performance is that most of these techniques are target-aware, making them very sensitive to distractors [Bal04; BO11] (other objects in the surrounding area of the targeted one), but also difficult or even impossible to implement in real systems. I therefore addressed this challenge of designing a target-agnostic pointing facilitation technique that could perform as well or better than target-aware techniques.

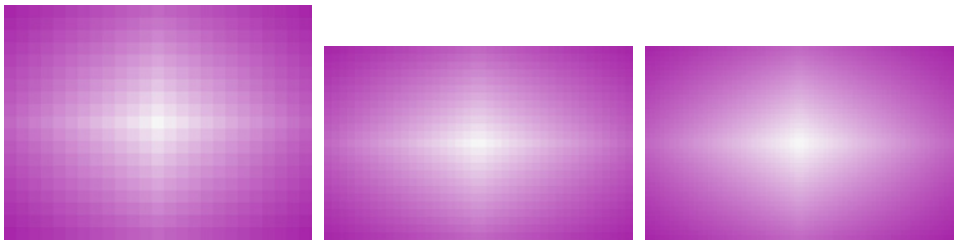
↳*pointing through screen edges could be beneficial*

For the technique to be target-agnostic, affecting target sizes was obviously not an option. We[1] needed to find a way to reduce pointing distance, without any knowledge of the desired target. The solution that we adopted was to build the technique upon the principle of "cursor-wrapping": allowing the system pointer to jump from one side of the screen to the opposite one when crossing an edge. Cursor-wrapping had already been introduced years ago in most common graphical environments: more than 15 years ago, the FVWM X Window Manager could be configured to enable it, and today, several Windows or Mac OS X system-level tools provide the same feature[2]. In fact, this can be implemented easily as long as an API allows to manipulate the system pointer, which is the case in all current systems. In terms of performance, this approach effectively reduces the pointing distance in many cases, and thus might reduce pointing time in theory: we performed some naive Fitts' law simulations showing clear benefits of wrapping in many situations. Some of these results are presented in Figure 2.1 at several screen resolution. Data is computed by dividing the screen in potential 40 pixel-wide square targets and estimating movement time for all the possible pointing tasks (by applying Fitts' law in both direct pointing and cursor-wrapping condition). For each starting target, the technique with the shorter movement time is chosen. In each case, the darker the color, the more beneficial is cursor-wrapping. While we had no evidence that cursor-wrapping obeys Fitts' law, which does not capture the potential additional costs of cursor-wrapping, these simulations were useful to assess the potential of the approach.

Despite these potential improvements, previous cursor-wrapping implementations had never been formally evaluated nor designed to effectively improve

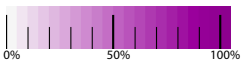
1 My collaborators on this project were Olivier Chapuis (CNRS-in|situ|) and Pierre Dragicevic (Inria-AVIZ).

2 e.g. www.networkactiv.com/SoundyMouse.html, www.digicowsoftware.com/detail?_app=Wraparound



(a) 1024x768 (b) 1600x1000 (c) 2560x1600

Figure 2.1: Cursor-wrapping Fitts' law simulation. Each graphic presents the percentage of cases when cursor-wrapping is beneficial over direct pointing for different screen resolutions.



↳*color scale for Fig. 2.1 (% of cases when TorusDesktop is beneficial over direct pointing)*

performance. We identified two main reasons why a naive cursor-wrapping technique could fail to effectively improve performance, or could have a limited area of positive impact:

A. The lack of control over cursor jumps;

B. The lack of feedback on cursor position after it jumps.

We then designed the *TorusDesktop* technique to address these issues. TorusDesktop implements an advanced cursor-wrapping technique that adds a dead-zone between opposite edges of the screen (see Figure 2.2). The dead-zone allows the user to control and trigger cursor wrapping on-demand, prevents accidental wrapping and preserves the property of using the border of the screen to stop the cursor while pointing targets on the edge (known as "edge pointing" [ACB08]). TorusDesktop also provides continuous visual feedback on both edges of the screen when the cursor is inside the dead-zone. This helps to control the cursor when traveling off-screen, and also when reacquiring it after wrapping around the edges. As shown in Figure 2.2, we explored three visual feedback methods, two inspired by related work (a and b in Figure 2.2, inspired respectively by the *Halo* [BR03] and *Wedge* [Gus+08] techniques) and an original one (c in Figure 2.2).

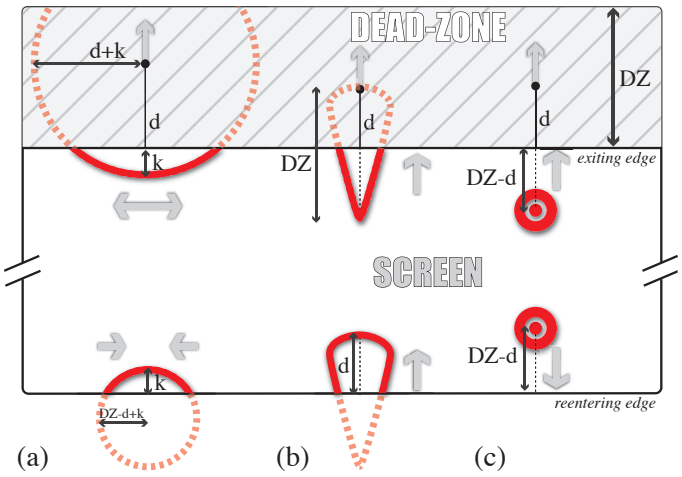


Figure 2.2: Torus Desktop dead-zone and feedback principle.

We conducted a series of controlled experiments in order to assess which feedback method was the most appropriate, which dead-zone size was the most efficient for preserving edge pointing without penalizing wrapping too much, and

finally we compared TorusDesktop to standard direct pointing. Details about the experiments and the results can be found in our CHI paper **[HCD11]** (see full article p. 143). Here I summarize our most important findings.

In our experimental setup, a 30" 2560x1600 display with a standard mouse, a 125-pixel dead-zone and our "ghost" feedback has proven to be the most efficient and preferred design for cursor-wrapping. As shown in figure 2.3, the comparison between direct pointing only, cursor-wrapping only and TorusDesktop (the user has to chose between direct pointing and cursor-wrapping for each task) gave us some interesting insights into the performance and the behavior of our participants, according to the direct distance to the target. First, we observe that starting from a direct distance of 2010px, both cursor-wrapping and TorusDesktop perform better than direct pointing, TorusDesktop having an average penalty of ~50ms over cursor-wrapping that we can explain with the cost of the choice between the two possible pointing trajectories. Below 1810px, we observe that TorusDesktop performs similarly to cursor-wrapping, suggesting that participants overused cursor-wrapping when direct pointing would have been beneficial. Finally, when the direct distance is between 1810px and 2010px, we observe that both techniques have similar performances, suggesting that this is an area of uncertainty, where choosing between direct pointing and cursor-wrapping does not affect performance.

While positive, these results are less advantageous than expected in our initial Fitts' law simulations. We computed a Fitts' law regression with our experimental data and obtained an acceptable overall fitting (r^2 = 0.848). However, we observed very high variability across dead-zone and target sizes, showing that Fitts' ID only very roughly captures the difficulty of TorusDesktop tasks, explaining the difference between our experimental results and our initial theoretical simulations (see Figure 2.1).

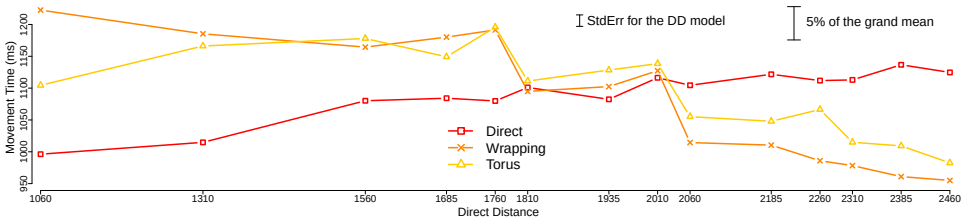


Figure 2.3: Average movement time of Direct pointing, Cursor wrapping and TorusDesktop as a function of direct pointing distance. Since the y-axis has its origin at 950, scale bars of error and grand mean are shown.

Overall, cursor-wrapping as we implemented in TorusDesktop is a viable solution for a target-agnostic pointing facilitation technique, that remains simple to implement and use in existing systems[3]. It adds a global ~200ms penalty to pointing tasks, but is still beneficial over direct pointing when the pointing distance is greater than 2010 pixels. However, as it was also demonstrated by Quinn et al., choosing between several possible pointing trajectories has a performance cost for the user [QCD12], and, as seen in our experiment, can lead to suboptimal choices. A solution to overcome this problem would be to help the user to learn which technique to use in each case. While we have not yet thoroughly explored this possibility, we already have some clues on how to

3 TorusDesktop is distributed as a free Mac OSX application (see http://insitu.lri.fr/TorusDesktop).

achieve it. We could build upon a theoretical model of cursor-wrapping with a dead-zone, similar to FITTS' law, in order to predict when the technique could be beneficial. We could then design training and feedback methods to help the user acquire sufficient expertise to make the right choice, e. g. overlaying a graphical representation similar to the one we used for our initial simulations onto the user's desktop.

2.1.2 *Advanced Input Methods for Triggering Commands*

Beyond pointing, another common task in standard interactive setups is triggering actions or commands from menus or shortcuts. One well-known problem of standard techniques, i. e. linear menus and keyboard shortcuts, is that they do not scale to the exploding number of commands of current applications [Bea97]. Users need extensive learning and practice in order to become proficient with them [Bai09]. Extensive research has been conducted in order to design menu systems that overcome this exponential growth of functionalities, and most of the time they are based on alternative graphical layouts and items arrangement (a very complete survey can be found in Gilles BAILLY's work "MenuA" [Bai09]). Another approach is to investigate the use of alternative input methods to augment menus or to trigger commands.

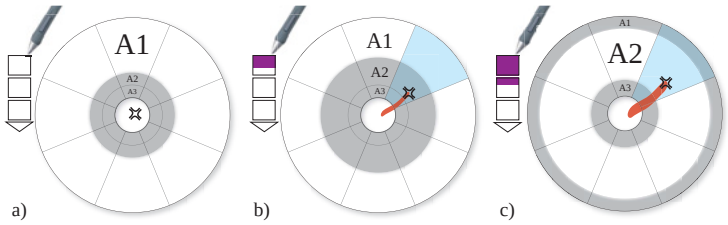


Figure 2.4: A 3-level Push Menu in different states: a) initial state (level 1 is active, colored in white). b) as the pressure increases, the ring of level 2 grows. c) when the applied pressure has raised above level 1, the level-2 ring is activated (in white) and the level-3 ring starts growing.

AUGMENTING MARKING MENUS WITH PRESSURE I extended *Marking Menus* [KB91] to take advantage of pressure-sensitive input **[HNB08; NBH09]**. Marking Menus are circular menus whose items can be selected as in a traditional menu (novice mode), or by performing only the corresponding gesture mark (expert mode) [KB91]. This approach provides a smooth transition from novice to expert, with the expert mode performing much better than other menu techniques. However, the number of items that a Marking Menu can contain is limited (~8) and hierarchical marking menus are more complex to use and memorize [KB93]. As shown in Figure 2.4, the *Push Menu* is a standard circular menu made of several concentric rings, one for each of the pressure level to apply to the input device. A dynamic visual feedback helps users to adjust and learn the amount of pressure required for selecting the desired item in novice mode. In expert mode, the menu behaves in a similar way to a standard Marking Menu, by showing only a gesture mark, but accounting for pressure. This augmentation of Marking Menu with pressure allowed us to double the number of items a user can handle with performance levels similar to a standard Marking Menu. The main limitation of the Push Menu is, of course, the required operation of a pressure sensitive input device. However, such technology is more and more common in many professional setups (pen-based interfaces), and even on standard laptops (pressure-sensitive touchpads).

↳*pressure doubles the capacity of Marking Menus*

RHYTHMIC PATTERNS AS AN INPUT METHOD A complementary approach to the use of advanced input channels is to take advantage of expert practices and user abilities in order to enhance interaction. This was the main topic of Emilien GHOMI's Ph.D. studies, which I co-supervised with Michel BEAUDOUIN-LAFON. In this context, we[4] explored the use of rhythmic patterns as an input method for triggering actions. In fact, rhythm is predominant in everyday life and has been studied in many contexts, including perception and action [Gla01; MM05], knowledge and learning (e. g. language [Lau+04]), artistic applications [Moe02], curing some diseases such as stress or sleep disorders [Sac08]. Despite our extensive understanding of rhythm in other domains, it has been little studied as an input mechanism in HCI.

↳*temporality and rhythm in HCI*

Interactive software take advantage of the temporal dimension, e. g. the distinction between long clicks and short clicks, "dwelling"–freezing the interaction to segment gestural interaction [Hin+05] or to switch mode [FCR09], successively highlight items in *Rhythmic Menus* [MAC99]. Some techniques, while not based on rhythmic input per se, rely on the temporal grouping of input events in a periodic way, e. g. double click, *Motion Pointing* [FEG09] or *Cyclostar* [MLG10]. But only a few techniques involve the reproduction of rhythmic patterns: *Five-key* [SL07] enables text entry with rhythmic sequences, in [CM06], tempo reproduction is used to select a particular song in a music library, and *Tapsongs* [Wob09] provides an alternative to textual passwords where users tap a rhythmic pattern that they have registered with the system for authentication.

↳*advantages of rhythm as an input method*

Beyond these point design, using rhythm to interact with computer systems has several potential advantages. First, as evidenced by research in Cognitive Science, there is a direct correspondence between performing a rhythm (action) and listening to a rhythm (potential audio stimulus and feedback). This could help to better learn and memorize commands associated to rhythmic patterns, similarly to motor learning. Second, rhythms can be performed in a variety of situations and has many advantages over gestural input in mobility conditions for example, since sensing rhythm only requires small sensors and reduced movements, and can be performed eye-free. But rhythmic patterns are not meant to replace more conventional command input methods. Instead, it is a way to enhance existing methods with a richer vocabulary, e. g. give access to a restricted set of commands, or conversely to simplify interaction in some cases, e. g. with a tactile device in the pocket or while driving, to shut down an alarm clock in the dark, or with devices that do not have a display.

Our objective was to operationalize and implement a rhythmic input method for triggering commands in order to generalize previous approaches by giving a better understanding on the use of rhythm in HCI, thus leveraging its benefits. Our study, which is reported in detail in our CHI'12 publication **[Gho+12]** (see full article p. 175), addressed three research questions:

FEASIBILITY: Studies in several domains – e. g. cognitive sciences, physiology, music – attempt to explain how and why we perceive and produce periodicities, but they rarely deal with the reproduction and memorization of rhythmic patterns associated to tasks. Thus, even if perceiving and performing rhythm is quite natural, are users able to reproduce, learn and memorize patterns? Can they use them to trigger commands?

4 In collaboration with Guillaume FAURE (former Ph.D. student at in|situ|) and his co-advisor Olivier CHAPUIS (CNRS-in|situ|).

INTERACTION DESIGN: The number of possible rhythmic patterns is virtually infinite and they can be presented in several ways. Which patterns make sense for interaction and how to design a vocabulary? What feedback helps in executing and learning patterns?

TECHNICAL ISSUES & INTEGRATION: Like most continuous high-level input methods, e. g. voice, marks and gestures, Rhythmic Interaction relies on a recognizer to segment and interpret user input. How can we design effective recognizers that do not require training?

We proposed a comprehensive framework to support the design of vocabularies of rhythmic patterns. Considering the number of commands and actions often used when interacting with computers, we introduced a definition of rhythmic pattern inspired by rhythmic motifs in music. A rhythmic pattern is a sequence of taps and breaks whose durations are counted in beats. Taps can be an impulse (a hit on an input device), a short tap (one beat) or a long tap (two beats). They always start at the beginning of a beat, and there cannot be more than one tap per beat. Breaks can be short (one beat) or long (two beats). A pattern cannot begin or end with a break or have two successive breaks.

↳*designing rhythmic patterns*

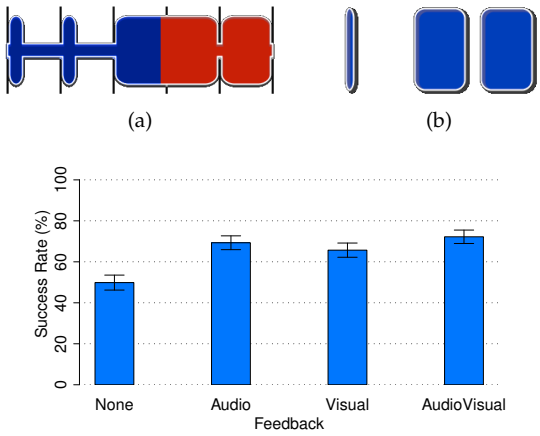


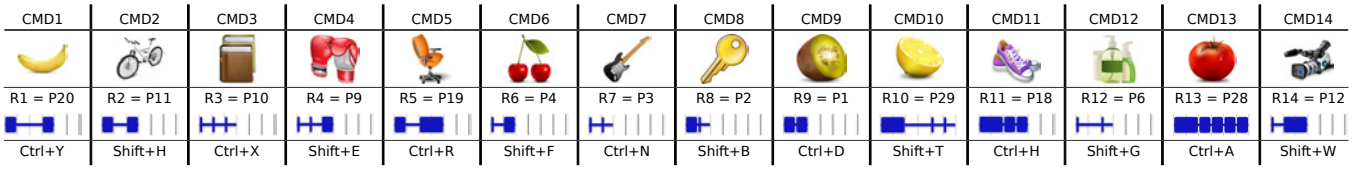
Figure 2.5: Reproduction of rhythmic patterns. (a) Our animated visual representation of rhythmic patterns presents impulses, short and long taps with shapes on a beat scale. Shapes are filled in blue, following the default speed (125 BPM); (b) During reproduction of patterns, a visual feedback follows user input in a similar way to the animated visual representation; (c) Reproducing rhythmic patterns is more accurate with feedback.

We first studied whether "standard" users (without expert skills in rhythmic-based practices like music) were able to reproduce rhythmic patterns following our design rules, varying complexity. We selected 30 representative patterns amongst the 799 possibles between two and six beats long, and asked 12 participants to reproduce them. Patterns were presented with the animated visual stimulus of Figure 2.5a augmented with sound while the shape was filled. Participants were then asked to reproduce the pattern by taping on the touchpad. We also investigated four types of feedback during the reproduction of patterns: visual (similar to the shape-filling stimulus), audio, audiovisual (combination of visual and audio feedback) and no feedback at all. In order to assess reproduction accuracy, we have implemented an intentionally very strict recognition algorithm, able to recognize the 799 patterns in the whole vocabulary and not just the 30

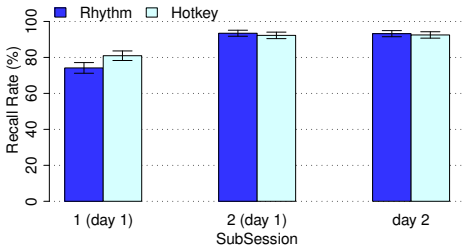
↳64.3% of reproduction accuracy with a very strict recognizer

patterns of the study. The overall success rate was of 64.3%. This may seem low, but the recognizer was deliberately very strict regarding the temporal structure of patterns. The task was thus similar to playing a percussion instrument, which can take years to master. Concerning reproduction feedback, we found that no-feedback was the worst condition (~50% recognition rate), all the other feedback conditions being not significantly different from each other (see Figure 2.5c). Overall, both quantitative and qualitative results suggest that rhythmic patterns could be a viable alternative for an input method, provided that feedback is given during reproduction and that the recognizer is less strict, adapted to the actual subset of patterns (a classifier).

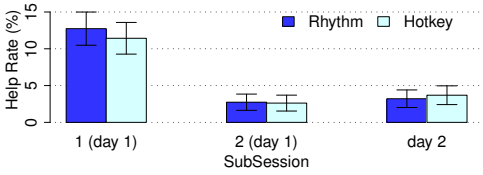
We implemented a pattern classifier which computes the "distance" between the input sequence and the patterns in the vocabulary **[Gho+12]** (see full article p. 175). When applying this classifier to the data of the first experiment, the overall success rate was of 93%. We used this classifier to conduct a controlled experiment, comparing rhythmic patterns with standard hotkeys, following an experimental protocol similar to Appert and Zhai comparison of gesture shortcuts with hotkeys [AZ09]. The comparison was studying the memorization of 14 "commands" symbolized by pictures, associated to rhythmic 14 patterns and 14 hotkeys (see Figure 2.6a). The experiment involved 14 participants and consisted of two phases: *learning*, where both the command and the technique stimulus were presented, and *testing*, where only the command was presented but with the possibility to invoke an help system. After each trial, participants were also asked to indicate the trigger they were trying to perform, in order to check if they remembered the trigger but failed to reproduce it, or to test if the recognizer classified a wrong pattern. The experiment was split into two sessions held on two consecutive days, with a free session on the second day, where participants were able to trigger commands with the technique of their choice.



(a) Commands and patterns



(b) Recall rates by session



(c) Help usage rates by session

Figure 2.6: Memorization of rhythmic patterns.

Our analysis of the results revealed very similar results for the two triggering techniques. As shown in Figure 2.6b, recall rate (the percentage of correct answers in the testing phase without using the help) are significantly lower in the first session of the first day than in the second session of the first day and the session of the second day (which are not significantly different). Recall rate for both technique is very close (around 93%), except in the first session were it

is significantly higher for hotkeys than rhythmic patterns (81% vs. 74%). For the use of help, the only significant difference is between the first session and the two subsequent ones (see Figure 2.6c). The second day, 10 participants out of 14 used rhythmic patterns more often than hotkeys, seven of them using patterns more than 80% of the time. Participants reported to have constructed some mnemonics to remember the rhythmic patterns. For instance, one of them remembered the "boxing gloves" of command `CMD4` and the corresponding pattern `P9` (see Figure 2.6a) as a "pif paf boom" onomatopoeia that echoed the "short short medium" structure of the rhythmic pattern for him. Overall, these results show that rhythmic patterns can be as efficient as ordinary keyboard shortcuts and have promising applications in many contexts of use such as mobile and eye-free interaction or expressive interfaces. In addition, we also proposed design guidelines for building rhythmic patterns vocabularies, as well as a user-independent recognizer software to integrate this input method into real systems.

↳*rhythmic patterns are recalled as efficiently as hotkeys*

2.1.3 *Improving Users' Workspace*

A recurring issue in window-based systems is window clutter. During Julie Wagner's Ph.D. at in|situ|, co-advised with Wendy Mackay, we investigated users' behavior with application windows. This work started from Julie's observation of her advisors' desktops, which were most of the time overcrowded with many opened windows. She hypothesized that some of these windows were left-over (unused for a while), resulting in an unnecessary window clutter, and that identifying the reasons for keeping these windows open should help design appropriate tools.

↳*50 windows on a real user's virtual desktop*

We implemented a background event logger for Mac OS X, inspired by [Cha05], and conducted a two week field-study with 10 laptop users. A second hypothesis was that the use of a laptop computer would lead to fewer reboots, resulting less desktop cleanup. We collected the logs from our background application (mainly windows events), answers to critical incident questionnaires during the study, e. g. "*why are you keeping this window open?*", and conducted pre-experiment interviews in order to identify participants' habits (reboot frequency, desktop cleanup, etc.). The detailed protocol and results appear in our publication **[WMH12]**.

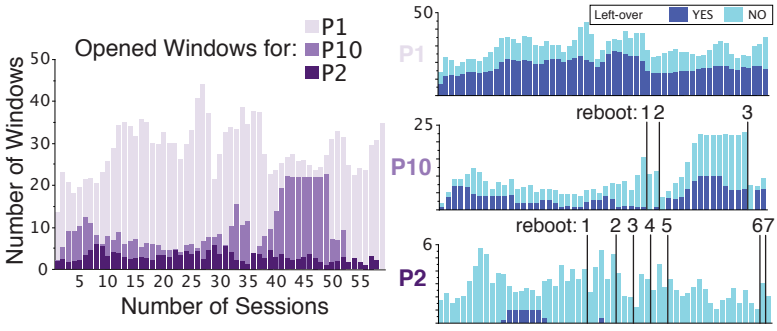


Figure 2.7: Open vs. left-over windows patterns: P1 has never rebooted, and has the most open and left-over windows; P10 rebooted 3 times, and has half as many open and left-over windows; P2 rebooted 7 times, ans has a few open and almost no left-over windows.

Overall, we found that interruptions during the day result in shorter work sessions but also increase the total number of left-over windows, with a corre-

sponding increase in window clutter, and that users who seldom reboot are more affected than those who reboot often (see Figure 2.7). We also identified a number of positive reasons for keeping left-over windows, including reminders, to do lists and facilitating future access to specific windows. However, participants also admitted that many windows were forgotten and caused undesirable window clutter. This provides several important results that could be used to inform the design of advanced desktop management tools, allowing users to quickly identify left-over windows and distinguish between those that serve a useful purpose and those that were simply forgotten.

Window clutter is also a common problem in more specific cases such as document editing with formatting or markup languages: editing environments are most often based on two or more synchronized side-views (editor and rendered document) that consume a lot of screen real estate and require efforts from the user to link the contents between views. We addressed this problem with Pierre Dragicevic (Inria-AVIZ) and Fanny Chevalier (OCAD University and DGP, Toronto), by designing and implementing *Gliimpse* **[DHC11]** (see full article p. 159), a complementary approach that provides in-place animated transitions between the edited markup code and the rendered document (see Figure 2.8 and video at http://www.aviz.fr/gliimpse).

Figure 2.8: Gliimpse. Animating from markup code to rendered documents and vice-versa.

Figure 2.9: Gliimpse. Animation of an HTML form.

Animation is triggered on demand by the user when she wants to quickly switch from the markup code to the final document without losing context and focus of attention. While not formally evaluated, the first user feedback we gathered suggests that Gliimpse could also be an efficient tool to help learning complex syntax (e.g., LaTeX formulae) since it allows seamless visualization of the mapping between the code and its results (see Figure 2.9). Our early prototype supports HTML, Wiki markup and LaTeX, and is freely available (see `http://www.aviz.fr/gliimpse`). We will see later in this section that behind this rather simple idea, the actual challenge of this work is its implementation for use in a real-world editing environment (see section 2.3.2, p. 37). In addition to the computer graphics algorithms that animate between characters in different fonts, and the stabilization method that maintains consistency between the views, the main challenge was to correctly map data from one view to the other. While one might think that software components in actual systems provide good APIs for introspection and accessibility, the reality is quite different, and we had to implement everything from scratch for our prototype to work.

This first set of projects already reveal some relationships between technology and interaction design. First, the design of novel interaction techniques does not only consist of studying users' needs or technological issues, then searching for the right solution (hardware and/or software). Novel interaction techniques can be inspired or influenced by technological features from the very beginning, in parallel with usage considerations (e. g. TorusDesktop or the PushMenu). Second, designing advanced interaction requires to implement prototypes at various levels of precision, for testing and evaluation purposes. As we will see in section 2.3.2 with Gliimpse, this can challenge existing technologies and even prevent potentially good solutions from being investigated further. The next set of projects will show that even when starting from user needs, technological constraints affect the design process.

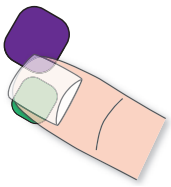
2.2 INTERACTION WITH MOBILE DEVICES

The ecosystem of mobile devices has evolved greatly since the appearance of the first Personal Digital Assistant (PDA) in the 90's, in terms of technology, as well as interaction. While most mobile devices have been shipped with touch-screens since the beginning, their interfaces often required the use of a stylus since they were based on the same WIMP paradigm as standard computers. The emergence of "smartphones" with multi-touch screens changed the landscape, leading to the development of interaction techniques specifically designed for these devices. However, whatever the technology, an old-school PDA or the latest multi-touch smartphone, interaction in mobility conditions creates strong constraints because of the devices themselves (small screen size, limited input, etc.) and also because of the context of use, mobility, interruptions and attention switching, etc. When designing mobile interactions, these issues must be addressed to ensure acceptable performance (time and errors), but also to improve and maintain an acceptable level of user satisfaction. My approach to the study of interaction on mobile devices was to address these problems in a global manner, with the explicit goal of defining high-level design guidelines inspired by the literature as well as my own research:

MOBILITY AND ATTENTION: Numerous studies and observations have shown that in mobility conditions, users are prone to use their small devices with only *one hand* [KBC08]. In fact, when walking in the street or traveling via public transportation, one must often carry a bag or grab a handle in a bus, limiting the availability of the second hand. Interaction techniques that can

be operated with only one hand, usually the thumb of the carrying hand, are preferable to stylus-based ones in these cases. Mobility also implies a lot of attention from the user, and in particular divided attention when simultaneously interacting with a device [PRM00], e. g. manipulating a navigation system while driving and observing the environment. We need to take this into account by designing "*interruptible*" interactions that can be put in stand-by and resumed later, without having to restart from scratch;

SMALL SCREEN SIZE: The physical size of the screen, despite the very high resolution available on recent high-end mobile devices, highly constrains the way data can be presented: data displays must be *compact*, *data-dependent* and/or *interactive*. For example, Focus + Context [CMS99] techniques such as *Table Lens* [RC94] can help adapt display to the user's current focus of interest; some techniques like *FaThumb* can improve search and navigation [Kar+06], and techniques such as *Collapse-to-Zoom* interactively adapt the size of the content depending on its relevance for the user [Bau+04];


↳*fat finger*

GRAPHICAL OCCLUSIONS AND INPUT PRECISION: Occlusions from the fingertip as well as the relative imprecision of direct touch for precise acquisition of small targets, known as the "fat finger" problem, is particularly acute with small touch-screens [VB07]. In mobility conditions, we thus face a contradiction between usage and usability: the need to interact with one hand, typically with the thumb while holding the touch-screen, makes interaction less precise and more error-prone on a small touch-screen; conversely, stylus-based or two-handed touch interaction can facilitate interaction and improve precision but are not well suited for mobility conditions. It is thus necessary to improve direct touch with specific designs and studies [HB10] or to favor indirect pointing methods with an offset cursor [PWS88].

I designed several new techniques for mobile interaction that illustrate this approach by addressing several issues: acquiring on-screen targets, displaying and manipulating lists of items and triggering commands.

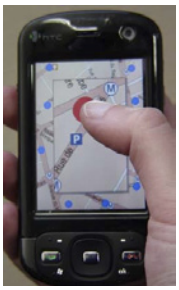
2.2.1 *Target Acquisition*

In collaboration with Anne ROUDAUT (then a Ph.D. student at Télécom ParisTech, now a post-doctoral fellow at Bristol University) and Eric LECOLINET (Associate Professor at Télécom ParisTech), we addressed the well-known problem of selection on a small tactile screen. As stated above, target acquisition on small mobile devices in mobility conditions raises the dual issue of making one-handed interaction with the thumb on a small touchscreen possible, without sacrificing performance and precision. We addressed three main problems:

1. **Visual occlusions** – occur when the finger tip or thumb hide parts of the screen;
2. **Accessibility of targets** – when operating a mobile device with one hand, targets close to the borders of the small screen are harder to acquire than those in the center [KB07; PH08], because the morphology of the thumb limits its reach;
3. **Selection accuracy** – touching a target on a small screen with the thumb is subject to inaccuracy because of the "fat finger" problem. A solution is to increase the size of the available targets in order to ensure minimal accuracy, as in several GUI for mobile systems. But this approach does not fulfill the requirements of many applications that require the display of many targets on the small screen, e. g. maps.

Related work only partially addressed these problems. Most target acquisition improvements on small touchscreen were inspired by the POTTER et al.'s *take-off* principle, which consists of moving an offset cursor by sliding the finger on the screen and releasing the finger when the cursor is on top of the desired target [PWS88]. This solves the accuracy problem, but visual occlusion and accessibility problems remain. *ThumbSpace* [KB07] was designed to improve access to targets on the borders, top and bottom of the screen with the thumb, but occlusions remain in the center. *Shift* [VB07] and *Escape* [Yat+08] address occlusions and accuracy but do not really improve accessibility, Escape even adds a level of complexity (gestures) that might make the technique too complex for casual users. We designed two selection techniques, *TapTap* and *MagStick*, with the goal of addressing occlusion, accessibility and accuracy concerns.

TapTap relies on a simple principle of "Tap-Zoom-Tap", as illustrated in Figure 2.10. With this easy two-step interaction technique, the user just has to tap the screen close to the target she wants to select in order to display a magnified area in the center of the screen, and to finally select the target. To keep the technique simple, the size of the magnification area and its zoom factor are not controlled by the user, but were carefully designed to make the technique efficient: a 80x120 pixel rectangle around the initial tap is zoomed in by a factor of 2, in which targets are zoomed in again by a factor of 1.5. More details about the design of TapTap can be found in our paper **[RHL08]** (see full article p. 124). Thanks to this design, this simple yet powerful approach addresses the three issues we raised previously. Zooming in an area of interest reduces visual occlusions and increases accuracy, since targets are larger. Positioning the area of interest at the center of the screen, wherever the initial tap occurred, improves accessibility of the targets that are outside of the extent of the thumb. More generally, TapTap coexists with the standard direct touch selection technique, since tapping directly on a target will select it without triggering the magnification area. It is also interruptible since the user can perform both interaction sequences at her own pace (and thus interrupt the interaction while the magnified area is displayed), and can be canceled by tapping on an empty space inside or outside of the magnification area.



↳ *TapTap*

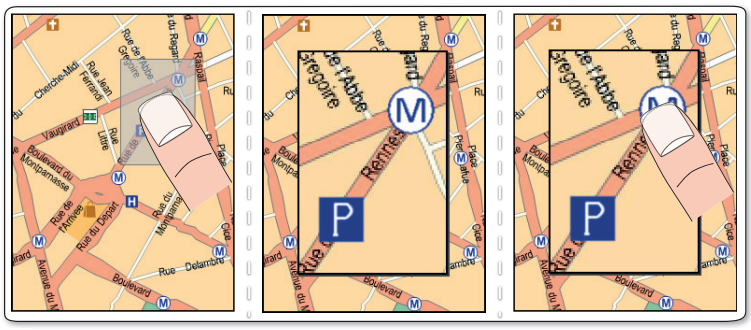
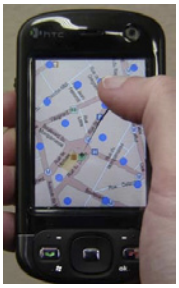


Figure 2.10: TapTap is a two-step selection technique: the user performs a tap in the area of the desired target, which is automatically zoomed in (x2) in order to make targets easier to touch directly.

The second technique, *MagStick*, extends the principle of the offset cursor [PWS88], as shown in Figure 2.11. When the user touches the screen, she defines a "reference point". Then, dragging the thumb makes a two-part stick appear around this reference point: The side of the stick which is under the thumb is for control and the opposite one holds a cursor. Cursor movements are mapped



↳*MagStick*

to thumb movements symmetrically with respect to the reference point. In order to support precise selection, the cursor is attracted by the targets when in their vicinity. Finally, a target selection is performed by lifting the thumb when the cursor hovers a target. This design, inspired by electronic billiards games, addresses several drawbacks of the standard offset cursor technique. First, symmetrical movement avoids graphical occlusions, since the thumb has to be moved in the opposite direction of the desired target, keeping the target and its surrounding context visible. Second, the cursor appearance when touching the screen, and its movements when dragging are easier to predict than with a standard offset. The later might require training to be mastered, especially to figure out where to touch the screen to access a particular target. With MagStick, the user can access all on-screen locations by always initiating the interaction from the center of the screen, which improve accessibility, but she can also anticipate the amount of movement she would have to perform *before* touching the screen, since the two ends of the stick around the reference point are of same length. Third, target magnetism avoids excessive adjustment movement, overshoots and empty selections that might sometimes occur because of small movements when releasing the thumb when confirming a selection.

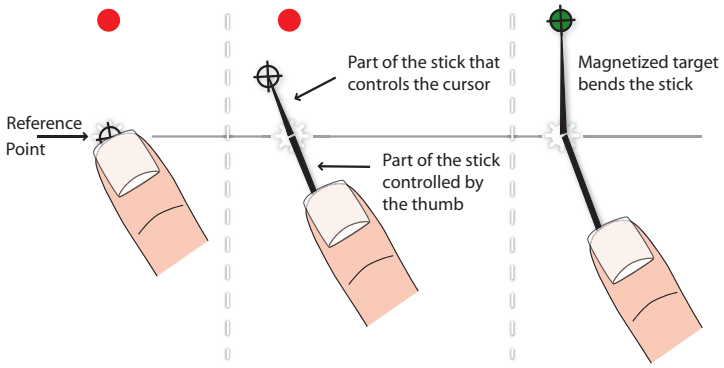
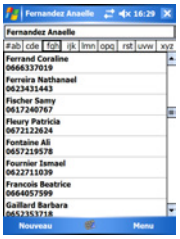


Figure 2.11: MagStick extends the offset cursor principle [PWS88] by enabling continuous control of a magnetized cursor at the end of an inverted telescopic stick.

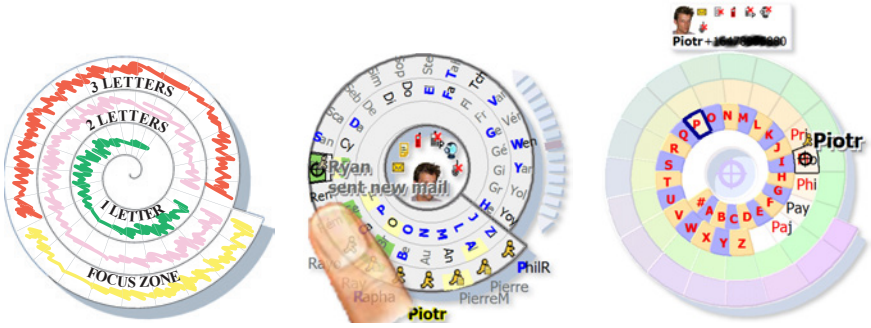
As described in detail in the related publication **[RHL08]** (see full article p. 124), we conducted a controlled experiment in order to assess user performance and accuracy with TapTap and MagStick, compared to the direct touch baseline, and state-of-the-art techniques at the time: *offset cursor* [PWS88], *ThumbSpace* [KB07] and *Shift* [VB07]. Our experimental conditions varied the on-screen location of fixed size targets (3mm, considered to be the smallest possible widget size on a mobile device [VB07]). While direct touch was clearly the fastest technique, it also caused too many errors to be really usable with such small targets. If we exclude direct touch, TapTap was the fastest and least error prone technique. MagStick was not significantly faster than the others but less prone to error. More importantly, both TapTap and MagStick have proven to have consistent performance (time and errors) for all on-screen locations, while other techniques exhibited sometimes important performance variations due to screen location. Participants' assessments were also very positive for both of our techniques.

2.2.2 *Manipulating Lists and Triggering Commands*

Visualizing and manipulating large quantities of data is a general concern in HCI, but is even more challenging on mobile devices for the reasons we described before (mobility, small devices and input methods). Even though mobile devices are not expected to be advanced visualization platforms, some of their primary uses depend strongly on the visualization and manipulation of data: as mobile phones, they might contain hundreds of contacts in an address book; as multimedia players, they might manage numerous entries and playlists in a library; as connected devices, they might contain many web bookmarks or navigation history entries; etc. When we started to explore this problem in 2007 with Eric LECOLINET at Télécom ParisTech, standard linear lists were the norm for displaying such data on mobile devices. These widgets were inspired from standard linear lists on desktop GUI, with a very similar behavior (scrolling and alphabetical shortcuts), requiring the use of a stylus for comfortable and reliable operation. New mobile platforms, i. e. *iOS* and *Android*, slightly improved the situation, since their list manipulation widgets were better designed for touch manipulation (larger items, speed-controllable scrolling with inertia). However, the linear presentation still has many drawbacks for displaying and manipulating large lists on such small screens. In fact, scrollable lists clip the data, presenting only a small subset of the items inside of a viewport. Beyond the increasing number of actions that are required to reach a specific item when the size of the list increases, the viewport principle prevents highlighting, does not support multiple points of interest in the list, and limits contextual information that could be presented to the user. This is an issue leading to poor designs where, for example, several missed calls or pending messages in a contacts list are presented in an additional sublist in another context.



↳*Windows Mobile 5 contacts list*



(a) Sectors and DOI zones (b) SpiraList (c) SnailList

Figure 2.12: SpiraList & SnailList. (a) General principal of the spiral layout for displaying lists; (b) SpiraList uses a *spatial* strategy where the whole list is displayed; (c) SnailList uses a *temporal* strategy where items are displayed and accessed in several steps.

We created a new interactive layout that addresses these issues, based on a compact spiral representation of data. Figure 2.12a shows our visualization technique made of a Focus + Context (F+C) [CMS99] spiral: each revolution of the spiral represents a Degree Of Interest (DOI), the innermost one being the lowest, and the outermost one being the "focus" zone. This compact layout makes it possible to display a large number of items inside the sectors, by reducing item size and grouping them using a semantic approach [PF93]. In our spiral F+C layout, items in the focus zone (the outermost revolution, in yellow in Figure 2.12a) are displayed completely. In the innermost revolutions, the number

of displayed letters of the items' labels decreases as we progress inside the spiral. Additionally, identical collapsed labels are grouped together, in order to optimize layout capacity and legibility. Some items can be kept outside of a group if they need to be highlighted, irrespective of their position in the spiral, thus allowing several points of interest in an *augmented context*. This layout, associated with the F+C strategy, suits the constraints of small screens while addressing many of the drawbacks of linear lists: the layout is compact and centered on the screen, making it easily accessible by direct interaction with the thumb; the variable DOI makes it possible to display a large number of items without hiding parts of the list as with a clipped viewport; items can be interactively moved from the context areas to the focus zone while keeping an overview of the whole data.
We explored this abstract visualization concept with two concrete designs, using two different strategies: *SpiraList*, which uses a "*spatial*" strategy, and *SnailList*, which uses a "*temporal*" strategy.

SPIRALIST

SpiraList **[HL06]** builds on a "spatial" F+C strategy. As shown in Figure 2.12b, items are displayed in alphabetical order all along the spiral layout in a continuous manner (the first and last visible items are contiguous). The focus area, containing the current selected item, is located at the bottom of the spiral and information about the selected items is inside of the spiral. Remaining items are displayed in the innermost revolutions, their labels reduced and collapsed as explained before. Unlike standard F+C visualizations, the location of the focus area cannot be moved and is fixed at the bottom of the visualization, making it well-adapted for small screens. Interaction with SpiraList is performed with an improved offset cursor interaction technique (the offset is adaptive, depending on the location of the touch, in order to help reach the borders of the screen). To change the focused items, the user must interactively "scroll" the list in the spiral with the blue arrows on its side, or to fly over an item with the cursor and to release its finger to make the list automatically scroll to get this item in the center of the focus area. Additionally, the background of spiral sectors can be colored and flying over items can display a tooltip, both conveying additional information about the items.

↳*SpiraList: a spatial strategy*

SNAILLIST

Another approach, that we instantiated with SnailList **[HL07b]**, relied on a "temporal" F+C strategy. While the spatial strategy of SpiraList has proven to be quite efficient, it scales poorly for very large or unbalanced lists, since many items are likely to be collapsed into the inner revolutions of the spiral, thus requiring many actions for the selection of an item. Keeping our compact spiral F+C concept, we explored a different approach based on temporal multiplexing. In SnaiList, a fixed context area always lies in the innermost revolution of the spiral. In this zone, as shown in Figure 2.12c, only the first letters of collapsed items are displayed (as well as some possible "always visible" items). Then, the user selects a letter in order to unfold the corresponding items after the context area. Finally, selecting an item puts it in focus on top of the visualization. As with SpiraList, interaction is performed with an offset cursor. The technique however deals with three levels of details (context, intermediate and focus) that are successively invoked by the user, hence the temporal approach. In comparison to the spatial approach, we expect the temporal one to reduce the complexity of visual search, by splitting it in simpler successive steps, while maintaining the good properties of our overall concept (compact layout and possibility of multiple points of interest).

↳*SnailList: a temporal strategy*

We compared our SnailList design with the standard Windows Mobile 5 scrollable list (which was the standard list widget on mobile devices in 2007), both

techniques being operated with the thumb of the carrying hand **[HL07b]** (see full article p. 110). Participant were asked to find specific items (names of persons) in balanced lists of 100, 250 and 500 items. While we did not find significant differences in search time between the techniques, the completion time was less sensitive to the size of the list with SnaiList. Regarding errors, SnailList was 3.7 times more accurate than the scrollable list, and results show that error rate with SnailList is slightly increasing with list size whereas it is always high for the scrollable list, whatever the size of the list.

In this section, I demonstrated how interaction design can follow a user-centered approach by studying specific aspects of the task and its context of use, e. g. mobility conditions, but can also account for technological specificities and constraints, e. g. screen size, device form factor and input precision. This improves the quality of interaction, gives insights into the advantages and limitations of a given technology and points to possible improvements. Touch interaction, which is the technological aspect of these studies, is still an on-going research area and section 2.3.4 will show that researchers are exploiting similar design principles to address both usability and technological issues.

2.2.3 *Advanced Mobile Interaction*

Recent hand-held devices also allow to explore new kinds of interaction techniques, thanks to their advanced technological capabilities, including multi-touch and high resolution screens, camera, sensors (gyroscopes and accelerometers), localization, etc. These enable new input and output methods, such as touch and mid-air gestures or contextual adaptation, which could address some of the issues we identified before, i. e. mobility conditions, limited display and input means, as well as to support more everyday activities, e. g. navigation, media authoring. In this section, I describe two projects designed to explore advanced interaction with mobile devices in order to (i) help users remember their interaction with physical objects by using Augmented Reality (AR), and (ii) overcome the limitations of multi-touch interaction in mobility condition by introducing bimanual interaction techniques on mobile tablets.

2.2.3.1 *Mobile Augmented Reality for Interacting with Physical Objects*

Interactive objects and appliances require more or less practice to be operated. But even after having mastered them, we often need to memorize how to set them up for a particular purpose: entering a code on a keypad to open a door, launching a program on a washing machine or an oven, or setting up a particular sound on a guitar amplifier. These appliances come with operating manuals explaining how to operate them, and users are sometimes keeping track of particular actions or settings on paper, or *Post-It* notes, or in their hand-held devices (with text or picture notes). But while these instructions and note-taking methods are efficient to describe the goal of the operation, their static nature does not always provide meaningful help on "how" to do it, or at the cost of multiple instructions (several lines of text or multiple pictures). In fact, they lack dynamic and contextual information, as it can been done with video instructions for example. When the instructions are long or complex, these "Alternating Attention" tasks [SM89] require to repeatedly switch between sub-tasks, which can be highly demanding for the user as it requires memorizing instructions, visually finding the objects of interest, retrieving the next instruction, etc.

↳*what was my setting for this song?*

During the Master's internship of Can Liu at in|situ|, which I supervised in collaboration with Jan Borchers and Jonathan Diehl from the Media Computing Group (RWTH Aachen University, Germany), we explored the possible use of mobile devices for assisting users in keeping track of their interactions with physical objects through Augmented Reality (AR) interactions. In fact, since the early KARMA prototype [FMS93], AR has proven to be useful in assisting with complex operational tasks such as assembly [Tan+03] or maintenance [HF11]. AR combines the advantages of text and pictures, by easing the localization of physical objects with additional "in-place" information [HF11]. There are many AR solutions that have been proposed so far for this application (see the complete review in Can Liu's master thesis [Liu12]), but most of them are focused on professional activities and are using a Head-Mounted Display (HMD). While the HMD provides the best solution for AR in terms of quality of the display and immersion of the user, this is obviously an impractical setup for our objective of helping casual users to interact with everyday appliances. Conversely, thanks to their technological advancement, recent smartphones offer an alternative for "Mobile Augmented Reality" applications to provide in-place guidance anywhere and for everyone. In fact, high-resolution screens and cameras, high-end mobile CPU and GPU with hardware accelerated graphics, and various connected technologies (Wi-Fi, bluetooth, GPS) are now making Mobile AR possible with off-the-shelf hardware, with a growing interest in both academia (a workshop dedicated to Mobile AR is held every year at the Mobile HCI conference) and industry (e. g. the Layar application[5]).

↳*AR instructions for casual users*

Our objective was twofold: (i) designing a mobile interface and related interaction techniques to create AR notes. AR systems are in general based on existing content, which is authored and managed by the designer(s) and the developer(s) of the application. In our case, the augmented instructions have to be personal notes and should be authored by the end-user; (ii) designing an AR layer on a mobile phone that enables the user to retrieve her notes and eases the manipulation of physical objects in real-time while following AR instructions on the mobile device (e. g. seeing the results of turning a knob in real-time). Similar approaches have been proposed since the early 2000 – e. g. "The Augmented Reality Personal Digital Assistant" [Gei+01] or Wagner and Schmalstieg's navigation system [WS03] – but they are mostly addressing the technological issues, they do not allow end-user authoring of the augmented data and they present "static" augmentation of the real world (text or video instructions, but without real-time updates from the changes in the environment). The challenge was then, for this second objective, to design and assess the benefits of a real-time AR instruction method.

AR NOTE AUTHORING We iterated over several designs for a mobile AR note authoring interface, whose objectives and design alternatives are described in details in our Mobile HCI 2011 workshop paper **[Liu+11]** and in Can Liu's thesis [Liu12]. Figure 2.13, describes a paper prototype of the interface. The principle is to enable semi-automatic detection of common physical widgets (e. g. knobs and rotary controls, sliders, buttons) and of their actual values when presenting the hand-held device in front of an appliance (Figure 2.13a & b). The detection could rely on computer vision algorithms and/or direct connection between the application and the hand-held device (e. g. through Wi-Fi or bluetooth). But it would also require user interaction in order to correct miss-detections and to modify the states of some widgets according to the settings in order to record personal instructions.

5 see http://www.layar.com/

Steps a to d in Figure 2.13 show a possible sequence: the system detects the widgets and highlights them in an AR layer on the device screen (Figure 2.13a & b). The user confirms the widgets that were recognized properly by tapping them on the screen. Some of them can be discarded if their values do not need to be recorded in the current note (the knob an the slider that remain red in Figure 2.13c are discarded in Figure 2.13d). Some of the widgets might not have been detected, such as the button in Figure 2.13b, or their detected values/positions might be erroneous, such as the rightmost slider in Figure 2.13d. The interface should then provide the users with means to correct detection: this could been done by directly adding or editing the AR layers and the recognized controls. For instance, in Figure 2.13e & f, the user adds a button on the AR layer and anchors it to the physical control. In Figure 2.13g, she corrects and re-calibrates the miss-detected value of the slider. Finally, the user could add some additional steps to the note (with the + button on the top right of he interface), in order to specify complex settings that would require to be performed step by step.

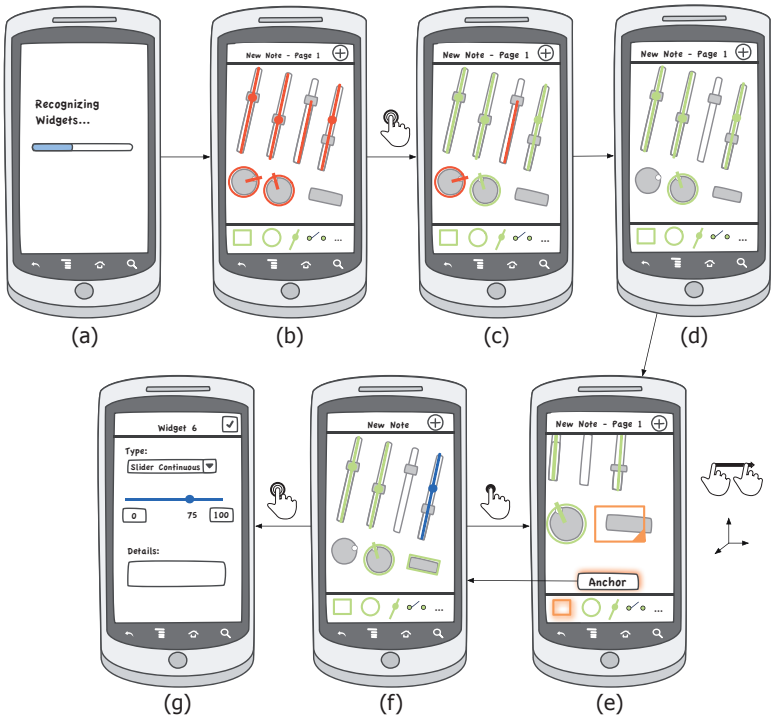


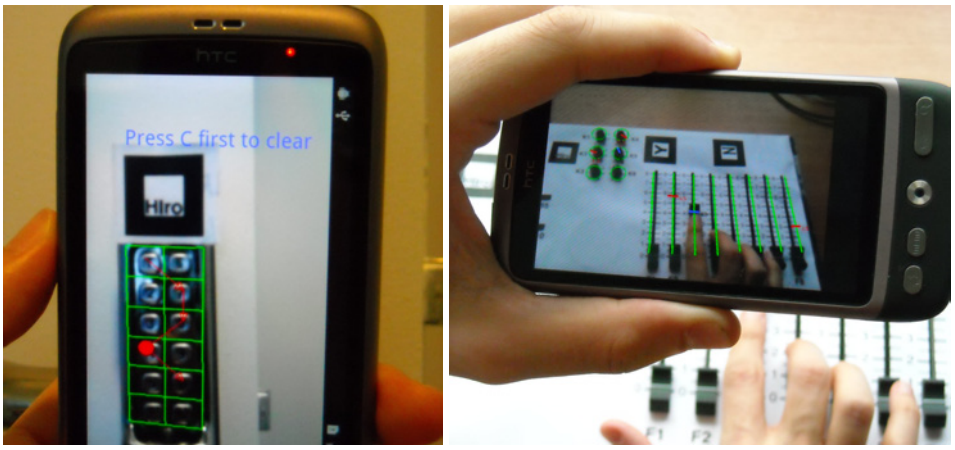
Figure 2.13: Storyboard and paper prototype interface for authoring mobile Augmented Reality notes (picture from Can Liu's master thesis [Liu12]).

This interface remains a mock-up for several reasons. Implementing it completely and properly would have required to address many technical issues, e. g. computer vision, ubiquitous devices connection, which where not our primary objective considering the limited time for this project. But more importantly, we focused our work on the second objective: assessing the benefits of real-time AR for setting physical controls. In fact, considering the cost of developing such an advanced mobile AR authoring system, one should first question the usefulness of the instructions it will help to create.

BENEFITS OF AR REAL-TIME INSTRUCTIONS Similarly to the authoring application, the instructions system consists in detecting an appliance when it is in front of the camera of the hand-held device, and to display an AR layer on top of its

control in order to present the setting instructions, similar to a *Magic Lens* [Bie+93]. Beyond controls detection and personal instructions retrieval, the main challenge was to design proper visualization and visual feedback for the instructions to be efficient. Figure 2.14 depicts the two approaches that we investigated. We first designed an "offline" AR feedback, which is the way most AR applications work: as shown in the example of Figure 2.14a, the instructions are displayed on-screen, on top of the physical controls, here a door keypad. The instructions are displaying the sequence of keys to press in order to unlock the door. The instructions can be displayed in a static or dynamic way (animation of the sequence), but they do not reflect the actions of the user on the physical objects in real-time. While making the implementation simpler, since it does not require to interpret user actions, this approach provides limited feedback on what was already done, and it might also be time consuming when performing complex sequential actions (since one could wait for the whole sequence to replay after missing a step for example). Moreover, visual occlusions from the user hand while manipulating the physical controls (between the AR display and the controls) could prevent the system from correctly mapping the values of the controls to their on-screen representation, thus requiring to switch attention between the AR view and the physical world as with picture instructions. Our second design addresses these issues by adopting a "real-time" approach, where the AR application gives real-time feedback on the actions of the user while manipulating the objects. In Figure 2.14b, we can see a set of physical controls that are manipulated through AR instructions. The values to reach are displayed in red, and actual values of the controls are updated in real time while the user manipulates them. When the correct value is reached, the on-screen feedback turns blue.

↳*how to display mobile AR instructions?*



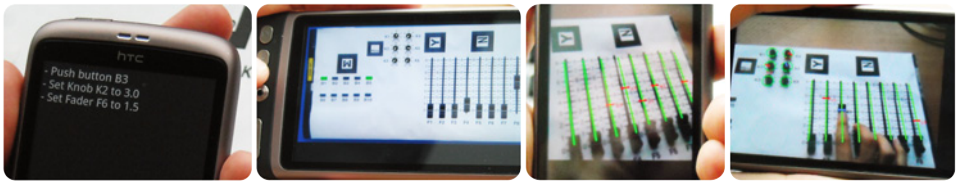
(a) Entering a code on a keypad (b) Setting continuous controls

Figure 2.14: Mobile augmented reality for setting physical controls. (a) An "offline" approach: values to be set are displayed on top of the physical control, in a static or dynamic way (animation), but without knowledge and feedback on the actual values. (b) A "real-time" approach: values to reach are displayed in red, and feedback is updated in real-time while the user is manipulating the controls. The feedback turns blue once the control is correctly set.

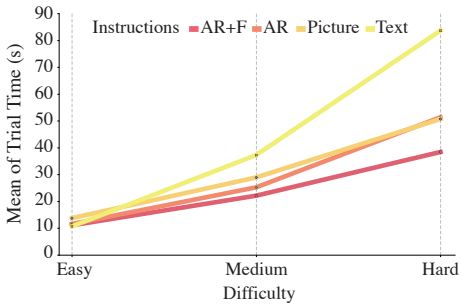
Again, our objective was not to fully implement the system, since it would have required complex computer vision algorithms and/or inter-device communication to work efficiently with real-time feedback and with any kind of controls. We implemented a "high-fidelity" working prototype in order to formally evaluate the approach. As shown in Figure 2.14b, we used fiducial markers and an Android

↳*an AR instructions prototype*

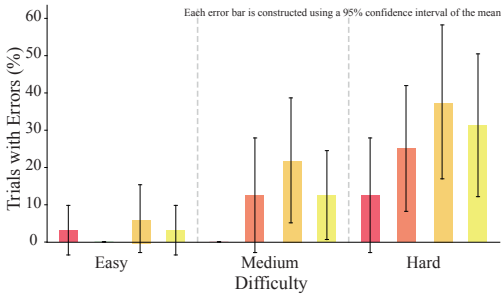
mobile implementation of the well-known ARToolkit library (NyARToolkit[6]) in order to detect the location of the physical controls with an HTC Desire mobile phone. Three markers were put on a JLCooper CS-10[2] MIDI control station that features controls commonly found on physical appliances (buttons, knobs, sliders and a jog wheel). In order to provide real-time feedback, the data from the controller (the values of the controls) is transmitted in real-time to the mobile application through the MIDI protocol and OSC messages. When the mobile phone recognizes a marker, it displays an AR layer on top of the real-time camera image with outlines of the physical controls and in-place instructions to perform the task, e. g. a value to enter or a button to press. The user can move the controls while looking through the phone, and see the real-time feedback on its screen, as explained before.



(a) Instructions Techniques (text, picture, AR and AR+F)



(b) Trial time



(c) Error rate

Figure 2.15: Mobile augmented reality experiment. (a) Experimental conditions; (b) Mean time for setting controls with each kind of instruction and by task difficulty; (c) Error rate for each kind of instruction and by task difficulty.

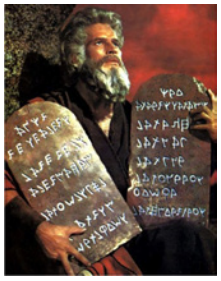
We have used this prototype to conduct a controlled experiment, which is described in details in our CHI'12 short paper **[Liu+12]**. Our goal was to assess the potential benefits of both AR methods as instructions for setting physical controls, with or without real-time feedback. Participants were asked to set the controls of our prototype with four kinds of instructions: traditional *text* or *picture-based* instructions, *AR* or *AR with real-time feedback (AR+F)*. These experimental conditions are shown in Figure 2.15a. We also investigated three levels of difficulty for the settings, which, from our observations during pilot studies, were based on the number of controls to set: 3 controls for *Easy*, 9 for *Medium*, and 18 for *Hard* (the types of controls to set in each settings were also balanced). The results validated the benefits of the additional feedback as well as the performance improvement of AR versus traditional text or picture-based instructions: (i) Augmented Reality techniques significantly increase users' performance (faster and more accurate, as shown in the graphs of Figures 2.15b and 2.15c); (ii) adding real-time feedback about user actions into the AR layer helps users performing even better: in fact, in accordance with our hypotheses, we observed that participants were using AR without real-time feedback in a similar way as picture-based instructions,

↳*real-time feedback improves mobile AR*

6 see http://nyatla.jp/nyartoolkit/wp/

because of hand occlusions limiting the possibilities of looking directly "through the phone" while manipulating the controls. Conversely, real-time AR feedback enabled the use of the hand-held device as a Magic Lens, reducing the need to switch attention between the instructions and the physical controls.

2.2.3.2 *Bimanual Interaction on Multi-touch Tablet Devices*



↳*bimanual interaction while holding tablets is an old story...*

Multi-touch tablets recently appeared on the market as affordable and accessible mobile devices, which was quickly confirmed by a wide adoption by many users. Their intermediate form-factor, half-way between a notebook and a smartphone, makes them adapted for a variety of nomadic usages, from professional to personal applications, in situations in which the user stands or walks: teachers can control simulations in class, nurses can track patients on interactive clipboards [Fon+10], etc. Interaction with multi-touch tablets is similar to smaller devices, based on gestures performed with one or several fingers of one hand. But their larger screen could also enable bimanual interaction, which can increases performance [MH08] and precision [BWB06] in the context of multi-touch input, and enhances the user experience [LZB98; WB03]. Bimanual interaction with a multi-touch tablet however poses an obvious problem: the non-dominant hand is carrying the device. Most existing bimanual interaction techniques were designed for independently supported displays or tabletops. Exceptions are *RearType* [Sco+10], which augments a tablet PC with a physical keyboard on its back, *Lucid Touch* [Wig+07], a prototype of see-through tablet, or *Gummi* [SPM04], a prototype bendable tablet device. These devices enable bimanual interaction with a tablet device, but this was not the primary intent of their design. Our objective was to better understand how bimanual multi-touch interaction can be achieved on hand-held tablets, and to inform designers about how to leverage the benefits of bimanual interaction techniques.

↳*how do people hold tablets?*

This work is part of Julie Wagner's Ph.D. thesis, which I co-supervised with Wendy Mackay. It is reported in detail in our CHI'12 paper **[WHM12]** (see full article p. 165) as well as in Julie Wagner's Ph.D. dissertation [Wag12]. Our methodology was first to study how people "naturally" hold multi-touch tablets, in order to identify holds that would makes interaction with the carrying hand possible while simultaneously interacting with the other hand. We recruited eight participants, four owned iPads and four had never used a tablet. We asked them to perform distractor tasks, i. e. pointing and scrolling, in different *tablet orientation* (landscape, portrait) and *stance* (sit, stand, walk), and we observed how they unconsciously held the tablet. We did not find a single optimal hold. Surprisingly, the four novices used the same awkward and uncomfortable hold, with the fingers, thumb and palm of their non-dominant hand supporting the center of the tablet like a waiter holding a tray. In contrast, participants who were used to manipulate a tablet found a variety of secure and comfortable holds, according to the device orientation. These five holds are presented in Figure 2.16 for the portrait orientation, holds in landscape orientation being the same. Participants also confirmed the accessibility of the borders of the multi-touch screen with both the thumb and the fingers of the support hand. These results suggest that we should seek a small set of roughly equivalent bimanual interactive holds that are easy to shift between, in order to alleviate the effects of fatigue (alternating between thumb and fingers for bimanual input).

BIPAD INTERACTION TECHNIQUES AND TOOLKIT We have thus implemented an iOS toolkit, *BiPad*, which help designers to add three predefined bimanual interactions to their applications. The toolkit defines the five interactive zones where the user can interact with the thumb or the fingers of the supporting

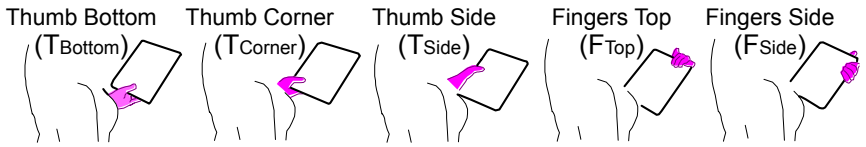
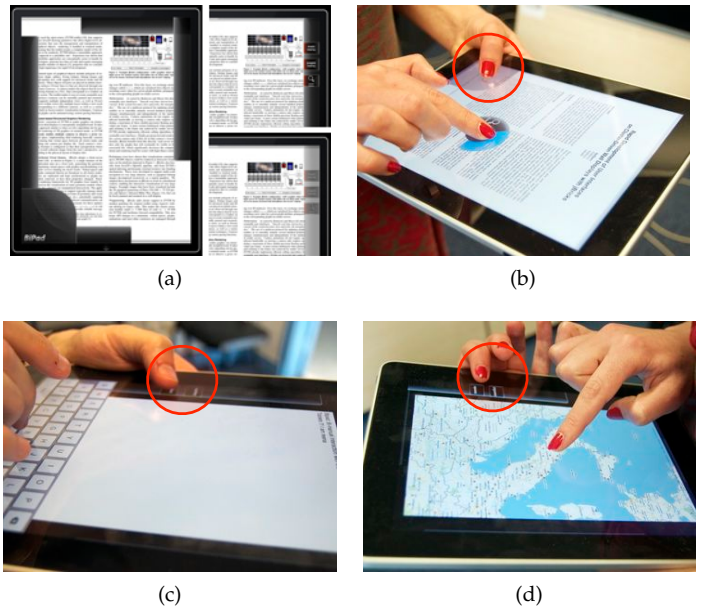


Figure 2.16: Hand postures while holding a tablet device. The five spontaneous holds identified for portrait orientation in our preliminary study. Holds are similar in landscape orientation.

hand, corresponding to the five spontaneous holds identified in the study (see Figure 2.17a). The application designer defines BiPad-enabled functions that can be mapped to interactions with the support hand through a simple callback mechanism. For example, a text editing application could define `shift` and `num` functions equivalent to pressing the shift or number keys of a virtual keyboard. Figures 2.17b, 2.17c and 2.17d give some example applications were different functions are mapped on bimanual interactions: navigating in a document, an augmented virtual keyboard or map navigation interactions (pan and zoom). BiPad additionally implements three predefined interaction techniques: bimanual *Taps*, *Chords* and *Gestures*.

↳*the BiPad toolkit (`http://insitu.lri.fr/Bipad`)*



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

Figure 2.17: Bimanual interaction on a multitouch tablet with BiPad: a) BiPad interaction zones for the non-dominant hand (which holds the device); b) navigating in a document; c) switching to uppercase while typing on a virtual keyboard; d) zooming a map. The non-dominant hand is holding the device and could perform 'taps', 'gestures' or 'chords' in order to augment dominant hand's interactions.

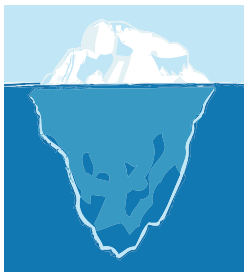
EVALUATION Thanks to the BiPad toolkit, we studied bimanual interaction with multi-touch tablets more thoroughly. As a first step, we designed the *BiTouch* design space, an extension of GUIARD's "Kinematic Chain Theory" [Gui87] that explicitly accounts for the *support function* while interacting with hand-held devices. We have defined three dimensions, *Framing*, *Support* and *Interaction*, which account for the different parts of the body that support the device, frame

the interaction and interact on the device (see details in **[WHM12]**, p. 165 and [Wag12]). We used BiTouch as a tool to analyze the five holds and the three BiPad bimanual techniques, and we conducted a controlled experiment as a preliminary exploration of how our design space captures this impact of the support.

We recruited 12 participants and asked them to perform a series of bimanual Taps, Gestures or Chords, using the thumb or fingers of the non-dominant support hand to modify the actions of the dominant hand. Tasks were performed while standing, and we designed 30 unique bimanual conditions involving the three BiPad interaction techniques, within the five possible holds and the two device orientation, with an additional unimanual control task. We found that bimanual BiPad techniques outperform the standard one-handed interaction for a similar task, with slight differences according to the device orientation. We also found that bimanual Taps were performing better than Gestures and Chords, and also get the preference of the participants: this suggest to consider a trade-off between simplicity (Taps) and expressive power (Gestures and Chords) when designing tablet's applications with such bimanual techniques. Regarding the BiTouch design space, our results suggest some interesting preliminary observations on the trade-offs when combining holds and orientations (thus changing the way to support the device) with our BiPad techniques. For example, considering performance, we observed that bimanual taps and gestures are significantly faster in holds with thumbs on the side, compared to holds with fingers on the side. In contrast, thumb on the side is perceived as less comfortable than fingers on the side. If we examine thumbs and fingers, we see that holding the device with the thumb on the side leaves only two joints available for interaction, whereas holding it with fingers on the side has three. This suggests that performance will be better with interaction techniques that offer a wider range of movement. We also observed a major effect on tablet orientation in some conditions, depending on the length of the support, e. g. the hand or the forearm, which results in a "lever effect" on the device during interaction. Overall, this suggests possible relationships between (i) performance and comfort, (ii) length and mobility of the support link and (iii) device form-factor and distance between the support and the balance point. Investigating these possible relationships by conducting dedicated experiments would probably help to extend BiTouch and to increase its predictive power, in order to inform the design of both new interaction techniques and interactive devices.

↳*taps rock and support matters*

2.3 INTERACTION DESIGN CHALLENGES TECHNOLOGY



↳*there may be more to this than meets the eye*

This overview of the research projects I have worked on during the past years highlights several of the methods and approaches we are adopting in HCI: *design* of new interaction techniques to address specific problems, *theoretical* framing of these designs and their *empirical* evaluations to validate their performance, accuracy, usability, etc., but also, and more importantly, to get a better understanding of their use in order to inform future designs. These are the most visible contributions of this research in several areas – pointing, desktop interaction, mobile interaction –, the contributions that we have emphasized in the related publications. However, this only the visible part of the iceberg, hiding the more laborious one that made these contributions possible: their *implementation*. In fact, most of these new interaction techniques required tedious and intensive coding, hacking, and sometimes pushing the limits of the underlying systems in order to be implemented "for real". Although our goal was not to develop final products, but working prototypes that were precise enough to assess both the usability and

the feasibility of our approaches, they required a lot of engineering.

There is a long lasting interest and discussion in HCI about prototyping methods and the kinds of prototypes to use when designing novel interfaces and interaction techniques. RUDD et al. distinguish between two kinds of prototypes [RSI96]: low-fidelity (Lo-Fi) prototypes, e. g. paper drawings[7], video [MF99] or computer mock-ups and storyboards, etc.; or high-fidelity (Hi-Fi) prototypes, e. g. working computer interfaces and interactions, with some of the functionalities of the final product being implemented or simulated. BEAUDOUIN-LAFON and MACKAY give a deeper analysis of the nature of prototypes, which accounts for the *representation*, the *precision*, the *interactivity* and the *evolution* of prototypes [BM07]. Regardless of the adopted design methodology, e. g. [EK84; BB95; MF97; CG04; Bux07b; BM07; Mac08; BK11], even the higher-level ones rely on prototyping, often throughout the whole design process and in different forms [BM07; LST08]: paper, video, cards, software. The important point to consider is the prototyping continuum in the design of interaction. Independently of varying definitions and analysis of prototyping methods, designers and researchers in HCI agree that in the early stages of design, rough prototypes have the advantage of being a useful way to communicate and to iterate on a proof-of-concept at very low development costs, helping to fix early issues [Bux07b]. The counterpart is that they have very limited utility after the early stages of design, whether for evaluating the usability or usefulness of the system, or for implementing the final version/product. Conversely, precise and interactive prototypes have more functionalities (vertical and/or horizontal) and a look and feel quite similar to the target product, enabling more thorough testing and evaluations, and even sometimes reuse for the final version. However, they are time-consuming to develop, requiring a lot of technical efforts, and are *de facto* limiting iterative design [MHP00]. The solution is then to adopt the right prototyping method at the right stage of the design, according to the objectives and the available resources, sometimes reusing or mixing prototypes at different "levels" for the next stage [McC+06].

↳*prototyping in HCI*

During our projects, my collaborators and I have faced this tension between the need for early rough prototypes for the exploration of design concepts, such as the paper prototype for the AR note interface in section 2.2.3.1, and more precise and functional ones for conducting quantitative and qualitative studies to evaluate and understand the interactive phenomenons we had created. In addition, since our intent was also to promote advanced interaction, high-precision computer prototypes were a way to integrate these new techniques into working systems and environments, pushing further the proof-of-concept to a proof of feasibility and "existence". In fact, although it may not be the current trend in HCI research, I think that as researchers, we should also promote and demonstrate the integration of our "inventions" in order to help progress towards "innovation" (referring again to BUXTON's "long nose of innovation" [Bux08]). In a "perfect world of interaction design", building advanced prototypes should be as easy as building rough ones, thus leveraging the benefits of both methods. But we are not there yet, and there is still a long way to go, despite some promising advances in that direction. Currently, building functional prototypes requires heavy coding efforts and a deep immersion into underlying architectures (hardware, software, languages and APIs), which is distracting from the primary objectives and time consuming. Paradoxically, implementing such advanced interaction techniques into real systems often relies on bricolage [TP92], tinkering and hacking, which are more common practices of early prototyping.

7 And not design sketches which are coming before prototypes [Bux07b, p. 139].

As a result, designing and implementing advanced interaction techniques often challenges existing technology, especially systems' architectures (hardware and software), models and patterns, and programming languages, APIs and programming tools. As it was already observed by OLSEN, "*our existing system models are barriers to the inclusion of many of the interactive techniques that have been developed*" [Ols07]. In the following section, from the lessons learned while conducting these projects, I discuss their outcomes from a new angle, namely from three different points of view that stress the advantages and disadvantages of the most common situations where *Interaction Design is Driven by Technology*, *Interaction Design is Constrained/Limited by Technology* and *Interaction Design Improves Technology*.

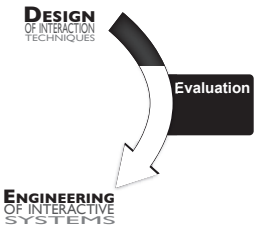
2.3.1 *When Interaction Design Is Driven by Technology*

Design of novel interaction techniques can be inspired or driven by some of the characteristics of the targeted system or architecture. These might be features, that are exploited in a beneficial way for improving interaction, but these can also be limitations, which will thus influence the very early design of the new technique. In the case of limitations, I do not mean that the objective of the newly designed technique is to overcome these limitations, but that those limitations will be considered while addressing another issue. TorusDesktop, the pointing facilitation technique that I presented in section 2.1.1 p. 10 is a good example of the latter, while the PushMenu (section 2.1.2, p. 13) illustrates the first case.

DESIGNING WITH LIMITATIONS As I explained when I introduced TorusDesktop, pointing techniques can be classified in two categories: "target-aware" or "target-agnostic" [Wob+09]. According to FITTS' law [Mac92], artificially increasing target size or reducing movement distance should improve pointing time. Intuitively, this suggests that if a pointing facilitation technique is able to adapt or give control to the user (implicitly or explicitly) on these parameters of the task, it might improve pointing performance and accuracy. In turn, this implies that to design and implement such a technique, one should be able to know the potential targets. As a matter of fact, such target-aware techniques are the most thoroughly investigated in the literature and are evaluated as the most efficient ones, whether they control the target size [BGB04], the cursor activation area [GB05; CLP09], distance to the targets [Bau+03], or even try to predict the user's targeted objects [Asa+05]. While these techniques were very carefully designed and effectively improve the performance of targets acquisition tasks, they all suffer a major drawback: they are very difficult to integrate into current interactive systems, since they do not provide any open and simple mechanisms to access and control potential targets. In fact, according to the related publications and my knowledge of existing interactive systems, these techniques were only implemented as ad-hoc prototypes, or with experimental toolkits designed for advanced interaction (e. g. the *DynaSpot* technique [CLP09] used the ZVTM toolkit [Pie05]). While the latter may help integration into specific applications, it would not work at the system-wide level, for all applications. The challenge is then to find a trade-off between studying the best design, which is indeed one of our objective as researchers in order to get a better understanding of interactive phenomenons, and studying the best "possible" design, which is also necessary from a practical point of view.

Our methodology for designing TorusDesktop was thus to consider a target-agnostic technique from the very beginning. First, because these kind of techniques have not been studied as much as target-aware ones, probably because they

are more difficult to design in order to be efficient. Second, because we accounted for the limitations of current desktop environments, which do not give easy access to existing targets, applications and user intentions. Considering that target-agnostic techniques can easily be integrated in real systems, even if the benefits are smaller than with target-aware techniques in terms of performance, the actual practical impact could be higher thanks to the system-wide implementation. This partly motivated the design of TorusDesktop, which does not mean that the technique was easy to design, tune and implement, as explained in section 2.1.1. But it was feasible "by design", and in addition to our CHI '11 publication [HCD11], it resulted in a Mac OS X application that can be downloaded and used by anyone in a standard desktop environment (see http://insitu.lri.fr/TorusDesktop).



↳*Interaction Design leads to evaluate Technology*

DESIGNING FOR FEATURES Conversely, and this should be a more usual situation, the design of interaction techniques can also be driven by specific technologies and the advanced capabilities of some systems, as a means to augment and improve interaction. These can be hardware or software features, or both. Multi-touch devices, for instance, which were first released with a limited set of interaction techniques, inspired many interaction designs such as the bimanual BiPad interaction techniques presented in section 2.2.3.2, p. 30. In the case of BiPad, the available technology inspired and made the design of bimanual techniques possible, improving interaction efficiency with hand-held tablets. Another example are pressure-enabled input devices, such as digitizer pens or recent trackpads[8]. Originally designed for basic control of stroke size in computer-based drawing tool, they were used early on to augment interaction in a more general way. Pressure Marks [RB07], for instance, augment gesture marks with pressure for triggering commands, and RAMOS et al. further explored a larger design space of pressure-based interaction with Pressure Widgets [RBB04]. I adopted a similar approach for the design of the PushMenu (section 2.1.2, p. 13), by redesigning the well-known Marking Menu technique [KB91] to account for an additional pressure input channel, thus increasing its capacity while maintaining usability. Regarding software, system-level features are probably not the primary source of inspiration to design advanced interaction. Higher-level software technologies or toolkits are more likely to provide features that suggest new designs. For example, *ARToolKit*[9] or *OpenCV*[10] have had a significant impact on the implementation of tangible and vision-based interactions.

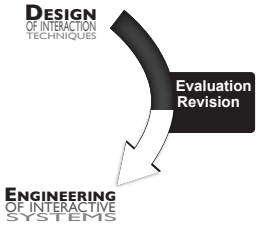
IDENTIFYING THE LIMITATIONS OF INTERACTIVE SYSTEMS Finally, many interactive technologies are widely used despite some well identified technological or usability issues. An obvious example is the "desktop metaphor" and its application windows paradigm, which is the basic User Interface (UI) of today's desktop computers. Despite a vast literature on the issues of window management and the many tools for improving the situation, desktop interfaces and interactions have not really changed over the past 30 years. This could be due to the difficulty of integrating such improvements into standard environments, as I already discussed earlier and that I will discuss again from the angle of engineering in the next chapter (see section 3.3.3, p. 64). But the fact is that designing new tools for fixing issues with a system first requires to identify these issues and their causes, which are sometime underestimated, thus leading to solutions that address only the surface of the problem. The study of left-over windows and their causes that I presented in section 2.1.3, p. 17 illustrates this approach. It gives insight into the way real users behave with existing technology,

8 See the Synaptics ForcePad™: http://www.synaptics.com/solutions/products/forcepad.
9 See the ARToolKit website: http://www.hitl.washington.edu/artoolkit/
10 See the OpenCV website: http://opencv.org/

identifying both the issues and the causes, which in turn gives insights into limitations or missing functionalities of the system. While this is not directly an influence of technology on interaction design as in the two previous cases, such studies are paving the way for future designs. As a side note, one should also consider the technical difficulty for conducting such field studies in real environments, that I will address later in the section 3.3.2 of the next chapter.

2.3.2 *When Interaction Design is Constrained by Technology*

In the previous section, I discussed situations where system limitations inspired a design for solving a particular problem, or avoiding these limitations while addressing a different one. Conversely, technology can constrain or limit possible designs. The design and implementation of the mobile interaction techniques presented in section 2.2, p. 19, illustrate this case. Beyond the specificities of mobile interaction, in terms of usage and usability, these platforms are also very constraining in terms of technology, especially at the time where I conducted this research: limited computing resources (CPU and memory), relatively poor graphical capabilities (no GPU or high-level graphical API), advanced but not always robust input technologies (reliability of touch sensors, accelerometers, etc.). Even nowadays, despite the obvious technological improvements of mobile platforms, there are still limitations with related software and development tools. In fact, although they are more specialized, mobile operating systems and development APIs are very close to those of standard computing platforms (desktop or laptop computers): encapsulated and controlled access to low-level resources such as input sensors; limited possibilities for implementing system-level interaction techniques (no access to the UI of the system or existing applications); very constrained development environments and APIs. While these restrictions ensure the homogeneity of interfaces and the respect of good practices and guidelines when implementing applications[11], they strongly restrict the possibilities for designing new interactions, leading to revisions of the technology.



↳*Interaction Design leads to revise Technology*

We faced many technical challenges during the design of the mobile pointing techniques and list manipulation techniques of section 2.2. First, implementing advanced graphics on the Windows Mobile 5 platform required pushing the limits of its very low-level and basic graphics API. A higher-level library, defining more advanced graphical objects and methods such as those of standard toolkits, e. g. Java2D, would have save a lot of time. It also required serious code optimization in order to make the technique and animations interactive, because of the limited graphical hardware and memory of the platform. For example, for the the graphics of SpiraList and SnailList in Figure 2.12, we had to implement lookup tables to optimize the computation of color gradients, transparencies, etc. We implemented these tools into a toolkit, which eased the implementation of further prototypes. But it was a constraint which could have limit our original design. Beyond these issues, the TapTap and MagStick pointing techniques also raised the problem of "integration". In fact, it would have been of great value to test these two techniques within real applications. But even though the platform was enabling system-level implementation of "InputMethods", it only consists in extended input panels for text entry. Both of our techniques were far more advanced, requiring to intercept touch events in order to create pointing events and, above all, to add complementary graphical feedback during interaction (see

11 See Apple's *iOS Human Interface Guidelines*: http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html and Google's *Android Design*: http://developer.android.com/design/index.html

Figures 2.10 and 2.11). This was unfortunately impossible, and we ended up with custom incomplete re-implementations of some standard applications in order to test our designs. Contrary to graphical hardware and APIs for mobile devices which are more advanced now, these issues with system-wide input management and access to applications interfaces still remain and we faced them again while developing the BiPad interaction techniques (see section 2.2.3.2, p. 30).

GLIIMPSE PROTOTYPE Finally, another project that illustrates many of the problems we can encounter while designing and implementing advanced interaction techniques is the Gliimpse markup language editor (section 2.1.3, p. 18). Gliimpse enables smooth transition between markup or text formating language editor and rendered documents in the same window. Our objective, while designing Gliimpse, was to demonstrate the usability and the feasibility of the approach with commonly used formatting languages: HTML, Wiki markup, RTF and LaTeX. Rather than conducting an empirical study, which would have required to define an experimental protocol from scratch since formal evaluations of such techniques are not common, the purpose of this implementation was to let users "feel and experience" the usefulness of the technique, by testing it in an editing environment as close as possible to a real one. But the trade-off was again to balance between the precision and interactivity of the prototype [BM07]: a basic software mock-up within an environment such as *Adobe Flash* would have enabled us to define simple predefined animations with fake code, but would have not supported free editing of the text with background generation of the resulting document and the real-time animation of custom content. While technically more challenging, the latter option was for us the best way to assess the potential and the feasibility of the approach.

We chose to implement a prototype in Java (see the details in our paper **[DHC11]** (see full article p. 159)). First, smoothly animating between graphical objects of different natures – e. g. groups of glyphs rendered with various fonts and sizes, pictures, graphical objects or form components – required the implementation of several non-trivial computer graphics algorithms. This was to be expected, since these features were at the core of the technique. However, the main challenge was to determine the actual mapping between the objects to animate, i. e. the text of the editing window and the resulting text and objects of the rendering window. This mapping should be at the character level, in order to create precise and smooth animations. We were expecting to be able to retrieve the mapping by inspecting the views (with an accessibility API) and to link this information with the output of the language interpreters (HTML or Wiki code interpreter, or LaTeX compiler). While this is close to our actual solution, the implementation turned out to be much more complicated than we expected.

Figure 2.18 illustrates the problem. It shows the relationships between the edited text (T_0), the code view (V_0), the document view (V_1) and the document in raw text format (T_1). $X_A Y_B$ denotes a function that maps subsets of X_A to subsets of Y_B. The mapping function $V_0 V_1$ is the one needed to compute animations between V_0 and V_1 (collections of glyphs and other graphical objects with all the information needed to render them individually, e. g. bounds, font, etc.), which no programming library or API provides. In order to reconstruct this mapping, we had to determine many other intermediate mappings because of the limitations of both programming toolkits (Java's inspection API) and languages interpreters or compilers: $T_0 V_0$, the mapping between the source code and its rendering in the text editor; $T_0 T_1$, the mapping between the source code and a raw text version of the final document; and $T_1 V_1$, the mapping between the raw text version of

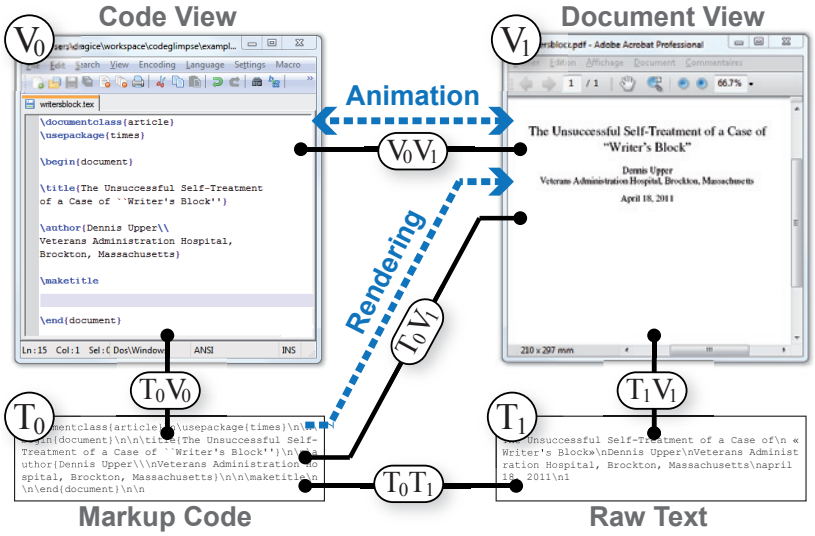
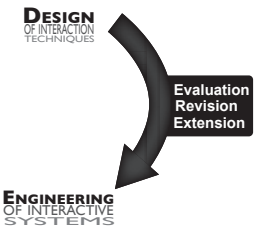


Figure 2.18: Gliimpse. Mappings between the code (T0), its view (V0), the document view (V1) and its text version (T1).

the document and the document view. Obtaining these mappings also required to overcome many technical challenges: extending inspection mechanisms for T_0V_0; implementing a custom document rendering view to help maintaining T_1V_1 and to enable animation rendering (especially for LaTeX to PDF documents); implementing custom source interpreters and "diff" algorithms to get T_0T_1. By composing these mappings, our prototype accurately rebuilds the character mappings between markup code and the raw text version of simple documents. It is however not robust enough for a final product: duplicate text regions and complex mappings can defeat the "diff" algorithm and cause document regions to be either incorrectly animated or not animated at all. As a temporary solution, we implemented a T_0T_1 mapping editor that can be used to author more complex sample demonstration scenarios, but it defeats our initial objective of a fully automated approach. Our Gliimpse prototype is usable for experimentation purposes and to demonstrate the approach. It also highlights some technical issues that could inspire the implementation of more accurate, robust and usable APIs for the exploration of similar preview methods. This leads to the last part of this section, discussing situations where the design of new interaction techniques helps revise and improve technology.

2.3.3 *When Interaction Design Improves/Extends Technology*

The last situation is when the technological issues raised while designing novel interaction techniques inform on the weaknesses and limitations of the technology. The solutions implemented for ad-hoc prototypes sometimes lead to more general tools that, beyond the original design itself, can be reused to facilitate other designs by solving some of the technological issues of the system. Taking again the example of multi-touch technologies, many protocols and frameworks for implementing advanced multi-touch interaction techniques come from specific needs in interaction design, e. g. the *TUIO* protocol [Kal+05] that enables abstraction and network connection of multi-touch devices and tangible interfaces, the *Proton* framework [Kin+12b; Kin+12a] that facilitates the design and recognition of multi-touch gestures, and many more tools and toolkits dedicated to extend systems to support multi-touch technologies. The toolkit for



↳*Interaction Design leads to Extend Technology*

developing advanced mobile interaction techniques mentioned in section 2.3.2 lies in this category, as well as the BiPad toolkit for bimanual interaction on multi-touch tablets (see section 2.2.3.2, p. 30), and our Rhythmic patterns recognizers (section 2.1.2, p. 14). The BiPad toolkit addresses only a few technological limitations, but its advantage is to promote our novel interaction method: The toolkit could help researchers and interaction designers to develop new bimanual interaction techniques and to integrate them (or the ones shipped with the toolkit) into their applications. Additionally, it also facilitates replication by giving the opportunity to reuse the actual implementation of our techniques as a baseline for comparison and evaluation of future designs, which is an important issue in HCI when conducting controlled experiment: One often has to re-implement former techniques that are not always described finely enough in academic publications, thus leading to uncertainties in the implementation and potential flaws in the experiments [Mac+07]. I will discuss this issue in more details in the section 3.3.2 of the next chapter.

Overall, beyond testing the techniques, the development of precise and functional interactive prototypes informs on possible technical issues, and can also be useful as a reference implementation for a more complete version. It can also result in an abstraction of several tools with the same purpose implemented in a toolkit that extends technology and developers possibilities [BM07; Gre07; Ols07]. As observed by GREENBERG, "*iterative prototyping would be far easier if we took the time to build a robust toolkit. [...] This often meant that we had to defer work on our main human factors goal*" and concluding that "*the payoff*" is "*rapid prototyping*" [Gre07]. However, prototypes implementations are most of the time very specific, addressing many technical issues, and the code is unlikely to be reused at a larger scale (e. g. in applications or in future toolkits).

2.3.4 *Revisiting Interactive Technologies*

Actual relationships between interaction design and UI technology are obviously not as simple as described above, and can certainly not be summarized by these three situations only. These situations are likely to overlap and to be intertwined during the design process of a novel interaction method. Our study of bimanual interaction on multi-touch tablets, for example, is a case where both "designing for features" and "improving technology" (toolkit-level implementation) were considered. However, the analysis of these simple situations already provides interesting insights. First, and not surprisingly, interaction design is influenced by technology. It can be a positive influence, taking advantage of specific features (e. g. pressure sensing) or even of some limitations that open different perspectives for design. It can also be a negative influence, when technology constrains the initial design, making it difficult to prototype/implement. Overall, this leads to a kind of retrospective evaluation of what can be achieved with a given technology, of its "possible". Because advanced interaction often pushes the limits of technology and is often based on unintended and unexpected uses of existing systems, it makes it easier to assess these possibles, beyond what was already expected and documented when the technology was first created. This leads to the second outcome of this discussion, i. e. that interaction design helps extend technology, by identifying possible revisions to address the encountered limitations/issues, and by suggesting or even implementing the related improvements.

A good example of this intertwined situation is, again, touch technologies. Touch sensing opens new possibilities for design but also limits these possibilities because of hardware constraints and user capabilities, as mentioned in sec-

tion 2.2.1. Sensor precision, occlusions and the "fat finger" problem are major issues that have driven research in this area and generated the design of alternatives to the more natural but imprecise direct touch, e. g. offset cursor [PWS88], our Tap-Tap and MaggStick techniques [RHL08], multi-finger and bimanual techniques for precise selection [BWB06]. These innovations have significantly improved touch-based interaction, but recent research has investigated how to overcome these limitations by improving the sensing technology itself, with for example fingerprints recognition [HB10], or better understanding of touch interaction [HB11]. Over time, we can observe an evolution from design for "features" and with "limitations", leading to an evaluation of the "constraints" of the technology and to "revision" and "improvement".

The evaluation of UI systems and technologies has been discussed extensively in the literature, pointing out the need of adapting the evaluation tools and methodologies to the problem at hand. Olsen [Ols07] has assembled a set of criteria into a framework for the evaluation and comparison of interactive systems. But depending on the problems and situations being considered, and simply because of the varying knowledge that designers/developers have of a given technology, these "measures" might be hard to assess *a priori*, and many issues might arise too late in the design process (e. g. while prototyping). A real progress would be to make such evaluations possible *a priori*, or at least in better coordination with the design process, especially during its early stages, in order to better inform the design. This would however require to rethink interactive technologies (i. e. systems, toolkits and associated tools) to better support advanced interaction (a long quest in HCI), to better disclosure their capabilities, and thus to better support interaction design.

The next chapter, after presenting some of my research work in engineering of interactive systems, will continue this discussion by addressing the complementary issue of how "Engineering Unleashes Interaction Design" (section 3.3).

"The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at or repair."

Douglas N. Adams – Mostly Harmless (1992) [Ada92].

3

FROM ENGINEERING INTERACTIVE SYSTEMS TO DESIGNING INTERACTION

Technology has some positive and negative effects on the design process depending on whether the required technology is available, accessible, adapted or not. From the *Interaction Design* perspective, technology is often challenged, and rarely provides the adequate tools to prototype, test and implement new designs. This new point of view on my research revealed the importance of this "hidden part of the iceberg" of interaction design for most, if not all, of the projects I have worked on. This chapter takes a deeper look at the underwater part of the iceberg – technology and, in particular, *Engineering of Interactive Systems* – to illustrate how it bears the tip of the iceberg – interaction design.

New UI toolkits often emerge once a specific problem has been explored with ad-hoc prototypes, and a set of tools has been built to abstract it out [Gre07], e. g. WIMP interfaces [MA88], zoomable interfaces [BH94; Pie05], groupware [TG04], mark- or ink-based interaction [HL00; AZ09], state machines support [BB06; AB08], multimodal interaction [BNG04; HDS11], multiple input support [DF04; CLV07; KRR10], multi-touch [Kin+12a]. For example, during my Ph.D., I addressed many technical issues to study novel interaction techniques for architectural design, which was my main research topic **[Hu005]**. Over time, many of the software tools that I implemented proved to fit together in a consistent framework for the prototyping and implementation of multi-input post-WIMP interaction. It became the *MaggLite* **[Huo+04a]** toolkit and its *Mixed-Graphs* model **[HDD06]**, and eventually the major contributions of my Ph.D. work. This specialization of software tools for advanced interaction highlights an important technological need: "using the right tool for the right task". Designing interaction with the

↳*the right tool for the right task*

wrong technology is like "driving a nail with a screwdriver": it can have bad consequences on the design process (as discussed in section 2.3.2), but also on the designer! Conversely, we should also be cautious with toolkits that address too many purposes, as they can lead to sub-par solutions and to an unmanageable complexity (in terms of use and maintenance). The challenge is then to determine the right level of granularity for the set of problems that a toolkit addresses, without sacrificing its generality [Olso7]. But most of all, since designing novel interaction techniques is by nature exploring a design space and often leads to combine paradigms and technologies, *modularity* and *interoperability* are important requirements for UI technology that supports interaction design. For instance, can a designer/developer easily implement a novel zoomable interface with the *ZVTM* toolkit [Pieo5], describe its interaction logic with state-machines through the *HSM* toolkit [BBo6] and support mark-based [AZo9] and multi-touch [Kin+12a] input?

Following the same approach as in the previous chapter, this chapter discusses this relationship between *Engineering* and *Interaction Design* through examples after reporting on my research work in the field of software architectures and tools for interaction. In the last section, I demonstrate that one should consider the specificities of interaction design when engineering new technologies.

3.1 UNIFYING TWO MODELS FOR DESCRIBING AND PROGRAMMING INTERACTION

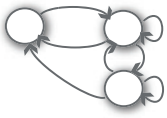
↳*the spaghetti incident*

A common problem in programming interaction is that handling input devices and describing the "logic of interaction" is mostly based on "callbacks", procedures that are triggered in reaction to events (from input devices, system, application, etc.). These callback procedures, usually implemented with listeners or delegates in object-oriented languages, result in an entanglement of procedure calls that makes the code of interactive applications difficult to understand, modify and maintain [Mye91]. It also introduces a high level of indirection between how interaction techniques are commonly thought of or described, and the way they are implemented: in fact, we are more inclined to think about "interaction" in terms of states and transitions between these states than in terms of reactions to events. *State-transition* formalisms are often used in the literature as a means to abstract and present interaction techniques [Bux90; HCS98]. Finally, if we consider the lower level of input management, the current events/callbacks approach is most of the time rigid, requiring to bind inputs with the interaction logic of the application in a static and quasi-immutable way [DF04].

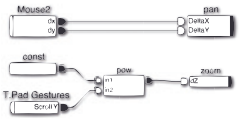
BEYOND THE STANDARD EVENT-DRIVEN ARCHITECTURE Many solutions to improve interaction programming beyond the event-driven architecture exist, and among the approaches that have been proposed so far, we can identify three main trends: *improved event-based mechanisms*, *state-based paradigms* and *data-flow models*.

Some architectures have improved over the basic event-based mechanisms. The *subArctic* [HMS05] toolkit, for instance, redefines the dispatch policy of events and makes it more controllable by the developer. *Garnet/Amulet* [Mye+90; Mye+97] introduced the concept of "Interactors", high-level objects that reify interaction techniques by intercepting events and transforming them into higher level operations on graphical objects. While they improve the development of standard WIMP interfaces and interactions, these approaches do not really enrich the interaction vocabulary. In addition, these toolkits are relatively opaque for the programmer, making it difficult to improve or extend them.

State-transition paradigms have proven to be a good, or at least useful, way to describe interaction techniques in the literature [Jac85]. In terms of interaction programming, state-transition approaches focus only on the states of the interactive system and consider only the external events that are relevant to the current state. This makes interaction the first-class object [Bea04]. Description and programming are more declarative, as well as more compact, clearer and more explicit than programming callback functions for every possible event and managing the state of the system with global variables. State-transition paradigms have been successfully used in toolkits such as *PetShop*, with the *ICO* formalism [BP99] that relies on Petri nets for specifying interaction, *hsmTk* [BB06] which introduces hierarchical states machines in C++, and *SwingStates* [AB08] which extends the Java language for defining state machines as control structures.



Data-flow approaches, on the other hand, are based on a cascade of processing devices, inspired by reactive languages. These approaches are highly modular, since processes and actions are encapsulated in independent bricks that only exhibit their inputs and outputs. The data-flow model makes it easy to describe sequential and parallel processing, and is well adapted to visual programing. Originally, it was mostly used for image, video or sound processing [Cyc; Mes] or for prototyping 3D interfaces and interactions [Das], but it was also successfully applied to advanced interaction: e. g. *ICon* [DF04] and *Squiddy* [KRR10]. It is now even used in the standard WIMP toolkit *Qt*[1], as an alternative to callbacks. The fine level of granularity of the data-flow model has obvious advantages in terms of modularity and flexibility. Its major drawback for describing interaction is however that some behaviors can become very complex to describe, especially if they include a lot of control such as conditional treatments and modes.



In summary, data-flow is well adapted to the description of low-level dynamic connection between input/interaction channels and logical components of application, but it does not scale well for describing states and modes. Conversely, state machines are well adapted to the description of the logic of interaction, but in a more static way. Combining both should increase flexibility and expressiveness when programming interaction. This is what we experimented with *FlowStates*[2].

FLOWSTATES FlowStates **[App+09]** (see full article p. 132) combines the data-flow and state-transition paradigms of the ICon and SwingStates toolkits. In FlowStates, ICon handles low-level events e. g. from input devices. ICon modules, which are normally connected to an application [DF04] or a scene-graph **[Huo+04a]**, are linked to SwingStates state machines by forwarding the signals they receive from their input slots as high-level abstract events that in turn trigger transitions between states. State machines then call application functions, e. g. to update the graphics or change the underlying data, through the actions associated to their transitions (see Figure 3.1). State machines can also inject abstract events into the data-flow and be disconnected from the rest of the application, which supports the combination of interaction techniques as suggested by the *Instrumental Interaction* model [Bea00].

This three-layer model (input devices, interaction logic and application objects) combines the benefits of each model for processing events at different levels of abstraction. The associated styles of specification – i. e. visual language for ICon and imperative language for SwingStates – address different issues: input devices

1 see Qt website: http://qt-project.org/.

2 FlowStates is a collaborative work with Caroline Appert (CNRS-in|situ|), Pierre Dragicevic (Inria-AVIZ) and Michel Beaudouin-Lafon (Université Paris-Sud-in|situ|).

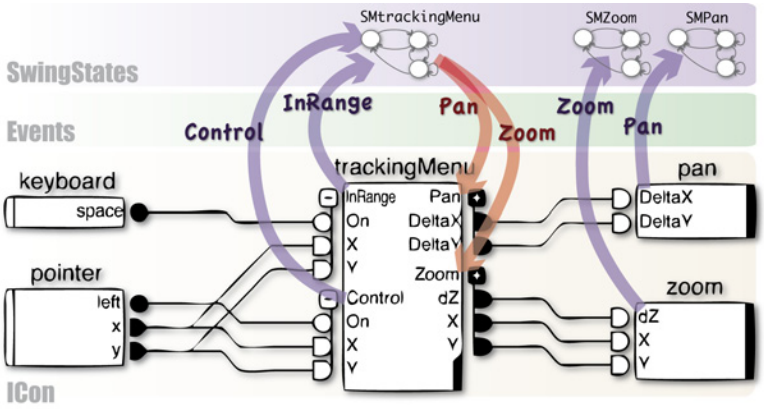


Figure 3.1: Example FlowStates configuration. Low-level ICon devices – *keyboard* and *pointer* – are connected to interaction devices – *trackingMenu*, *pan* and *zoom* – implemented with SwingStates state machines. The *Events* layer shows custom FlowStates events that trigger state machines' transitions when the appropriate slots of an ICon device are updated.

might change and need to be reconfigured often, while the logic of interaction is more stable and tied to the application [92]. However, we designed FlowStates in such a way that the frontier between the two models is not frozen, giving the programmer complete control over which model to use for each purpose.

Figure 3.2 shows a simple example of a FlowStates code construct. It uses an extended syntax derived from SwingStates, allowing to specify *state machines*, *transitions* and *abstract events*. As we can see in the figure, each instance of the `IConStateMachine` class will be instantiated into a corresponding ICon device (the "zoom" and "pan" state machines of Figure 3.2). Transitions that are defined inside a state machine are bound to groups of input slots in the corresponding data-flow processing device, according to the event class which is responsible for triggering the transition (the "Zoom" group of slots of the "zoom" state machine). Finally, input slots are derived from introspection of the transition event classes: an input slot is created for every pair of `setSlot/getSlot` methods defined in a transition class (the `setSlotDZ` and `getSlotDZ` of the "Zoom" event class).

In practice, when the generated device will be connected inside of a running ICon configuration, the FlowStates engine will generate instances of the corresponding events each time an input slot is updated. This event will be propagated to the linked state machine which will then update its state accordingly, as shown in Figure 3.1. This very simple code snippet illustrates the basic principles of FlowStates. More complete constructs that demonstrate extended features and possibilities can be found in our paper **[App+09]** (see full article p. 132). In particular, we have shown that FlowStates can be used to simply implement state-of-the-art interaction techniques such as the "*Tracking Menu*" [Fit+03]. Not counting the code of the graphical part of the technique, the FlowStates implementation consists of about 30 lines of code to describe the behavior of the technique (see Figure 3.3). Besides this relative simplicity and expressiveness, it makes the technique generic and easily reusable in other applications simply by plugging the generated ICon device into input devices or methods in a data-flow configuration at run-time. Moreover, it improves the level of control and customization of the technique by supporting a large variety of input devices, and even changing some of its behavior: for instance, in **[App+09]**, we easily transformed the tracking

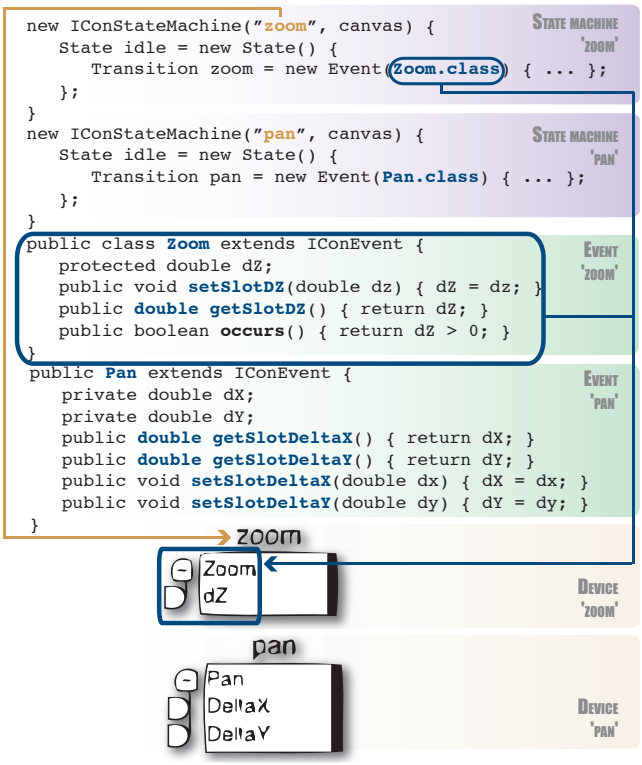
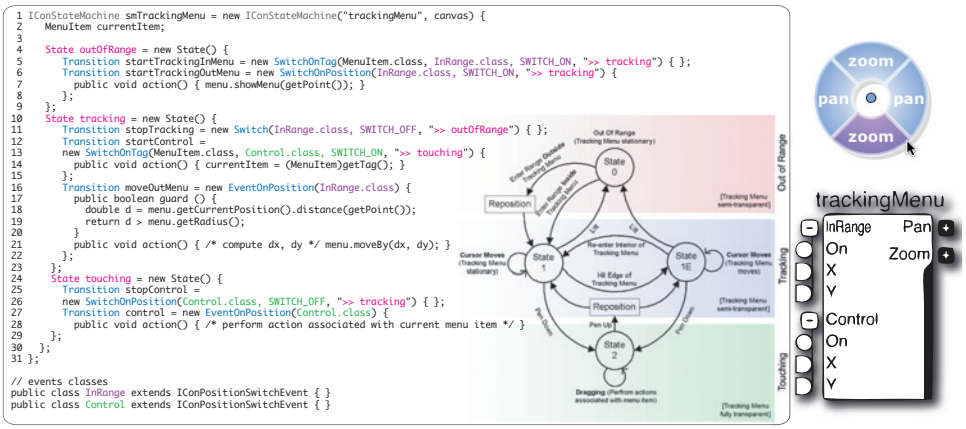


Figure 3.2: Basic FlowStates code constructs. *IConStateMachine* classes are instantiated as ICon device prototypes. *Event* classes that trigger state machines transitions, are instantiated as groups of ICon slots and added to the devices of the state machines that are using them.

menu into a "*Trailing widget*" [FVB06] by simply connecting an inertia processing device to the values that are sent to its positional input slots. Finally, thanks to the possibility of injecting outputs of the state machines into the data-flow through the output slots of the processing devices, several state machines can be dynamically connected and reconnected together. This makes it possible to decompose the logical part of an application into several small and reusable blocks.

While FlowStates is not the first and only "hybrid" attempt to combine state-oriented and data-flow paradigms – e. g. PMIW [JDM99], ICon+PetShop [Nav+06], IntuiKit [CLV07] or StateStream [HP09] – it is probably the most integrated one while at the same time letting the programmer decide which model to use in each particular case. The "link" between state-transition and data-flow paradigms can be instantiated by the developer closer to the toolkit or closer to the application. For instance, the `Pan` and `Zoom` devices and their associated state machines in Figure 3.1 are closer to the application logic, since they define some parts of its functionalities (pan and zoom). Conversely, the `trackingMenu` is closer to the toolkit, since it does not define any part of the application logic but a standalone interaction technique. This provides a controllable level of granularity to the developer, an approach that is particularly modular and opens many possibilities for prototyping new interaction techniques that can be changed and combined together and with various input devices or methods at runtime.




```
1  IConStateMachine smTrackingMenu = new IConStateMachine("trackingMenu", canvas) {
2    MenuItem currentItem;
3
4    State outOfRange = new State() {
5      Transition startTrackingInMenu = new SwitchOnTag(MenuItem.class, InRange.class, SWITCH_ON, ">> tracking") { };
6      Transition startTrackingOutMenu = new SwitchOnPosition(InRange.class, SWITCH_ON, ">> tracking") {
7        public void action() { menu.showMenu(getPoint()); }
8      };
9    };
10   State tracking = new State() {
11     Transition stopTracking = new Switch(InRange.class, SWITCH_OFF, ">> outOfRange") { };
12     Transition startControl =
13     new SwitchOnTag(MenuItem.class, Control.class, SWITCH_ON, ">> touching") {
14       public void action() { currentItem = (MenuItem)getTag(); }
15     };
16     Transition moveOutMenu = new EventOnPosition(InRange.class) {
17       public boolean guard () {
18         double d = menu.getCurrentPosition().distance(getPoint());
19         return d > menu.getRadius();
20       }
21       public void action() { /* compute dx, dy */ menu.moveBy(dx, dy); }
22     };
23   };
24   State touching = new State() {
25     Transition stopControl =
26     new SwitchOnPosition(Control.class, SWITCH_OFF, ">> tracking") { };
27     Transition control = new EventOnPosition(Control.class) {
28       public void action() { /* perform action associated with current menu item */ }
29   };
30 };
31 };

// events classes
public class InRange extends IConPositionSwitchEvent { }
public class Control extends IConPositionSwitchEvent { }
```

Figure 3.3: Tracking Menu implemented with FlowStates. The state machine describing the behavior of a "Tracking Menu" (from [Fit+03]) and the corresponding FlowStates code. On the right, the ICon device which will be automatically generated in order to control the state machine.

3.2 DISTRIBUTED GRAPHICS AND INTERACTION IN MULTI-SURFACE ENVIRONMENTS

The "Personal Computer", whether it is a standard desktop/laptop computer or a mobile device, is still the norm for the vast majority of users of interactive systems. Since the late 90's, these devices access remote or distributed data more and more often, e. g. web pages, databases, multimedia resources. However, during the past decade, we have witnessed a progressive swift from "data-only" distribution, to a richer but more complex distribution of computing resources. From the user's point of view, we can identify two main trends. The first one is commonly called Cloud Computing: while the user is still interacting with her personal client device, she makes transparent use of distributed services, such as data, storage, applications, computing power, etc. This "evaporation" of computer functionalities has many advantages, including enabling interaction with anything from anywhere with any device, freeing users from hardware and location constraints, and improving distant collaboration by facilitating the sharing of resources. It also raises many technical (e. g. infrastructure, security, performance, scalability) and usability (e. g. accessibility, awareness, privacy, reliability) problems that I will not discuss here [3]. The second trend, which is more central to my research, concerns the distribution "across" end-user devices, and relates partly to the concept of Ubiquitous Computing [Wei91]. Here, applications, and interaction, can be split across multiple devices that the user will operate simultaneously or in sequence. Whether they are called "multi-devices", "multi-surface", "smart", "ambient" or "interactive" environments, they all share the same concept of multiple interconnected input and output devices that cooperate seamlessly and must be combined together for the user to accomplish tasks. Again, the concept of such environments is not new and emerged several decades ago [Wei91], but recent technologies – e. g. display, input, network – made deeper exploration possible only recently.

↳towards the distribution of interaction

Multi-surface environments, as I will call them in the following, have been the subject of much research in HCI, addressing both interaction design and

3 See for example the recent (2012) report from the "CLOUD Computing Expert Working Group" of the European Commission (http://cordis.europa.eu/fp7/ict/ssai/docs/future-cc-2may-finalreport-experts.pdf).

engineering. Noticeable breakthroughs in terms of interaction include REKIMOTO's *Pick-and-Drop* [Rek97] and *Augmented Surfaces* [RS99], *Dynamo* [Iza+03], which supports multi-user data sharing across multiple surfaces. More fundamental studies have investigated e. g., pointing [Nac+06] or perception [Nac+07] in multi-surface environments. For technology and engineering, which are the primary concerns of this section, the issues of modeling, designing and implementing such interactive systems and their interaction techniques, was addressed by several research fields, with different perspectives: Distributed & Parallel Computing, Computer Graphics, Software Engineering, and of course HCI. Before giving a short overview of the state of the art of this research area and summarizing my own contributions, I briefly describe the WILD platform, which was developed in the in|situ| group as a testbed for conducting this research.

3.2.1 *The WILD Project*

Wall-Sized Interaction with Large Datasets (WILD) was a project initiated by Michel BEAUDOUIN-LAFON and coordinated by Emmanuel PIETRIGA, involving three research groups: in|situ|, Inria-AVIZ and LIMSI-AMI. The main objective was to design an interactive platform for studying both single-user and collaborative visualization and manipulation of massive datasets. The motto of the project was "to design an extreme environment for designing with extreme users", a platform which, by pushing the limits of both hardware and software technologies for highly demanding working conditions, would inform the design of future interactive systems. Several laboratories from the Paris-Saclay campus – astrophysics, particle physics, chemistry, molecular biology, neuroscience, mechanical engineering, and applied mathematics – were invited as associate partners, since their research activities require a large variety of tools and techniques to manipulate and understand massive and complex scientific datasets. We conducted workshops with them as potential end-users, discussing and investigating relevant interactive tools to support their work-flow in such an environment. As summarized in our recent IEEE Computer paper **[Bea+12]**, the outcomes of the project – which ended in 2012 but is now continued as part of the larger Digiscope project – range from fundamental HCI to novel interaction techniques (see for example Mathieu NANCEL and Julie WAGNER Ph.D. theses [Nan12; Wag12] and related publications), and from design methods to engineering of interactive systems (see the *Shared Substance* [Gje+11] and *jBricks* **[Pie+11]** frameworks described below).

↳*a platform for scientific exploration*

The *WILD Room* is a multi-surface environment arranged around a wall-sized display as its central component (see Figure 3.4). This display is a tiled surface made of 32 30-inch high-resolution LCD monitors (Apple Cinema HD) arranged in a 8×4 grid for a total of 131 million pixels (20480 x 6400). The screens are mounted on four independent mobile carts that make it possible to change its spatial configuration (flat or "curved") The resulting 5.5m wide by 1.8m high "*wall*" is driven by two front-end computers and a cluster of 16 computers with two high-end graphic adapters each: each computer drives two screens. A defining characteristic of WILD is its very high pixel density (about 100 dpi), which is rare on wall displays, most commonly made of rear-projected panels (for a pixel density of 30 to 60 dpi). Another characteristic that most other wall-sized display miss, even larger ones such as UCSD's *HIPerWall* is interactivity: WILD also features a high-end marker-based motion tracking system with 10 infrared cameras. The system has very low latency and a precision of less than one millimeter, enabling accurate 3D tracking of users, devices, as well as advanced mid-air interactions. A 1920×1080 FTIR interactive tabletop that can sense up

↳*into the WILD*

to 32 contact points, with RFID capabilities, allows several users to collaborate, manipulate and exchange data with the wall (see Figure 3.4). Finally, various hand-held interaction devices are available – e. g. gyroscopic mice, wiimotes, multi-touch smartphones and tablets, Anoto pens, physical props –, while the platform's wireless network provides users with a way to connect and use their own personal devices (laptops, smartphones, tablets, etc.). Beyond designing new interaction techniques (interactive visualization of large datasets, collaboration over multiple surfaces and devices) and working with scientists as end-users (specific workflows and methodologies, personal habits, limited availability), WILD raises a number of software engineering challenges **[Bea+12]**.

(a) Wall display and tracking system (b) Hand-held devices (c) Collaboration

Figure 3.4: WILD is a multi-surface environment designed for studying collaborative visualization and manipulation of massive datasets. It is made of a high-resolution tiled display with 32 30-inch monitors arranged in an 8×4 grid for a total of 131 million pixels (20480 x 6400), a high-end tracking system with 10 infrared cameras, a 1920×1080 FTIR interactive table, and a variety of hand-held interaction devices: gyroscopic mice, PDAs, tablets, etc.

3.2.2 *Technical Issues of Distributed Graphics and Interaction*

The applications shown in Figures 3.4b and 3.4c illustrate some of the technical issues encountered in such environments. High resolution enables high-quality 2D graphics, opening wall-sized display platforms such as WILD to new applications, e. g. in astronomy, geospatial intelligence and visual analytics at large. These applications essentially combine very large bitmap images, high-quality text and 2D vector graphics, e. g. satellite imagery augmented with data layers (see Figure 3.4b), or information visualization techniques for displaying large datasets, e. g. for the visual exploration of large networks, or the comparison and analysis of similar data (see Figure 3.4c). In Figure 3.4b, the user is using a multi-touch tablet to interact with high-resolution images of astronomical data displayed on the wall display. The hand-held device could be used to manipulate the pictures (panning and zooming), but also to apply filters (e. g. colors, frequencies) in order to explore the dataset. The example of Figure 3.4c involves two users in a collaborative task. They analyze and compare 3D models of Human brain scans (e. g. tagged sulci, connections, pathologies), which are displayed on both the wall display and the interactive table. The user in front of the wall takes advantage of the high resolution of the wall display to make precise observations and comparisons. He can manipulate the displayed 3D brains by moving a plastic brain in his hand. The orientation of the brain is tracked by the VICON, so is the position of a wand to designate areas of the brain. The user at the table has a less detailed view of

the data due to the lower resolution of the display, but a better overview of all the brain scans. Using touch interactions, he can rearrange the layout of the brains on the wall, according to the observations and potential matches or differences among them. The two users clearly have different roles and tasks, depending on the components of the platform they are using: the user in front of the wall acts as an observer of the data, while the user at the table is the coordinator of the overall task. These two examples and scenarios are sufficient to point out the five major technical issues commonly encountered in these environments:

DISTRIBUTION OF DATA PROCESSING AND RENDERING: As illustrated in both examples, data visualization can be continuous along a surface, as in Figure 3.4b where the wall is considered as a unique panel, or split across surfaces, like in Figure 3.4c where brains are displayed in a grid of 3D models on the wall, and possibly in another form on the table. With an underlying architecture made of 19 connected computers – two front-ends, a cluster of 16 computers, and one computer for the tabletop – driving 33 displays – the wall and the tabletop – plus additional computers and displays such as hand-held devices or laptop computers, how do we create an integrated, seamless experience for the user? The problem is to create strategies that are adapted to this distribution, specifying which parts of the platform are processing data, performing the rendering, and how they communicate (e. g. centralized or distributed processing, replication or sharing of data).

↳where is the data & where does the rendering take place?

DISTRIBUTION OF INPUT CHANNELS AND INTERACTION LOGIC: This issue resemble the previous one but focuses on interaction. How and where are the input events handled, and how are interaction techniques defined in the system? For instance, the multi-touch tablet in Figure 3.4b might be used as both a pointing device for the wall – i. e. by tracking its 3D position – and a track-pad to manipulate the pointed data – i. e. by sensing user touches on its surface. From the user's point of view, it is a single input device. From the designer's and the developer's point of view, multiple input channels, coming from different sources in the environment, must be combined in order to create a "logical device" with its related interaction logic. The question is then to determine which part of the architecture is performing this combination, and where to define the logic of interaction, i. e. the response to the actions on this logical device.

↳where is the interaction?

More generally, these first two points question the concept and the model of an interactive "application" in these environments, which becomes closer to distributed and cloud computing principles, where an application is a dynamic composition of services. However, in cloud computing, the distributed part mostly concerns data storage and the functional part of the application, but rarely interaction which is typically running on the user's personal computer or device.

USING LEGACY APPLICATIONS AND TOOLS: Another problem is to reuse existing applications in such environments. The brains comparison example of Figure 3.4c comes from a real task by the neuroscientists we worked with. It must use both the dataset and the rendering that they are used to work with, and thus the application and the functionalities that are essential to their work-flow, augmented with the specific capabilities of the platform, e. g. high-resolution rendering, multiple display surfaces, advanced interaction. Rewriting the complete application from scratch would be the best solution to adapt it to the new environment, but it is likely to be tedious and

↳where is my app.?

time-consuming, without any guarantee that it would fulfill the requirements of the original one. Conversely, porting legacy applications to such environments is not straightforward, and highly depends on their level of "openness": Is the source code available? Does the application provide libraries or APIs? Is the application scriptable or does it provide a remote control protocol?

PROGRAMMING AND PROTOTYPING: The previous issue also extends to the development of new dedicated applications. Since programmers are more likely to use languages, tools and APIs they already know and master, e. g. OpenGL for rendering), how to enable their use on this architecture? Should we completely hide or on the contrary expose its distributed nature? Our own experience with WILD is also that such a platform requires maintenance downtime or is sometimes running at full capacity, making it unavailable to some users. This raises the issue of tools that would let users (developers) program, simulate and test applications and interaction techniques outside of the platform, but minimize the overhead when porting them back into the actual environment. For instance, the graphical rendering of the two aforementioned applications could be tested on a dual-monitor setup and the interaction techniques developed with cheaper devices (e. g. MS Kinect, Nintendo Wiimotes) before being deployed and fine-tuned "in the WILD". Beyond that, we should also facilitate iteration and prototyping of new designs with appropriate tools. As already discussed in section 2.3, p. 32, developing precise prototypes of novel advanced interaction techniques often challenges "mainstream" technology (desktop computers and mobile devices). In extreme interactive environments such as WILD, this could be even more challenging without appropriate tools. In addition to the increased difficulty and amount of work to implement prototypes, it might have a negative impact on the overall project schedule and especially on end-users' involvement if there is too much time between successive prototypes.

↳*where is my API?*

COLLABORATIVE INTERACTION: The last issue is the support of collaborative interaction, with appropriate *groupware*, the technology enabling Computer-Supported Cooperative Work (CSCW). Groupware applications and supporting toolkits should: account for the different *roles* and the *awareness* of their users; define how *data sharing* is achieved (replication, centralization or mixed); enable *synchronization* of changes on data; and provides mechanisms for controlling *concurrency*. For example, in the brains manipulation example of Figure 3.4c, data (the brains' models) is shared across both surfaces. Loading new brains' models could be restricted to the tabletop user only, as a coordinator of the task (role). When this user loads a new brain model, this should be reflected on both side (synchronization). While the user who initiated the change will notice it, his colleague may be immersed in another task and should be notified (awareness, which is even more crucial for remote collaboration). Finally, some manipulations such as changing the position of a brain scan, should be exclusive in order to avoid conflicts.

↳*where is my mind?*

These issues have been addressed in several ways, by different research fields, with an abundant literature. The reader can refer to Ni et al.'s survey of applications and technologies for large high-resolution displays [Ni+06], Endres et al.'s survey of software technologies for Ubiquitous Computing [EBM05] or the book "*Computer Supported Co-operative Work*" for groupware technologies [Bea99]. In summary, while the aforementioned issues of distributed graphics and interaction have been addressed in many ways and from many angles, no single solution or tool has emerged to solve both problems. Computer Graphics methods [EMP09; Hum+02; Jeo+06] address the quality and performance of

distributed rendering, but lack high-level structured graphics and toolkit support for advanced interaction **[Pie+11]**. Conversely, several toolkits dedicated to Post-WIMP interaction provide such mechanisms [BGM04; Huo+04a; Pie05; AB08], but without supporting distribution. Studies of software architectures for Ubiquitous Computing and Ambient Environments have led to elegant and powerful descriptive models [Cou06; Dem+08; MVV11] of distributed interaction, but lack concrete and reliable prototyping and implementation tools, especially for Post-WIMP interaction and distributed rendering [Cou10]. Finally, groupware solutions define and implement efficient multiple-input/multiple-user architectures and mechanisms [DC91; RG96; TG04; AGG09; Bie+08], but are limited in terms of high-quality distributed rendering.

3.2.3 *Engineering in the WILD*

During the WILD project, we investigated three approaches to address these issues in prototyping and implementing custom applications:

↳*"I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail"* – A. MASLOW, *1966*

IMPLICIT DISTRIBUTION: From the developer perspective, this model is similar to standard standalone WIMP applications. Distribution and synchronization of data and the rendering process are transparent for the developer. *Hydrascope* [HMB13], implemented by Michel BEAUDOUIN-LAFON and Björn HARTMANN (Assistant Professor at UC Berkeley), relies on web technologies to leverage their intrinsic capabilities for distribution and sharing. It consists of a client-server architecture for web applications, based on content replication and synchronization. Clients are web browsers distributed in the multi-surface environment, and the application (the server) generates and distributes the interactive content. Distributed interaction is achieved by notifying the server whose will propagate changes to other clients when a client is updated. This approach as the advantage of relying on well-known and widely used web technologies e. g., HTML, CSS and JavaScript, running in standard browsers. This can be very useful for fast prototyping and for porting applications to different platforms, provided that synchronization mechanisms are transparent for the developer.

EXPLICIT DISTRIBUTION: With this model, sharing is transparent but distribution is explicit. The resulting toolkit, *SharedSubstance* [Gje+11], was developed by Tony GJERLUFSEN (former intern at in|situ|), Clemens NYLANDSTED KLOKMOSE and James EAGAN (former postdocs at in|situ|), Clément PILLIAS (former research engineer at in|situ|) and Michel BEAUDOUIN-LAFON. The developer defines processes (called "environments") that run on different machines and discover each other dynamically, thus composing a distributed application. Environments contains a hierarchical data structure that they can share in whole or in part with other environments. Through this structure, environments can share different kinds of resources: data of any type, rendering capabilities, input streams, etc. Sharing is thus transparent but distribution is explicit since the developer must specify the required environments for a given application. However, he does not have to care about "where" the environments are running and "how" they share the data (similar to a service-oriented architecture). The brain scans comparison application of Figure 3.4c was implemented with SharedSubstance. It defines several environments including: rendering environments on each computer of the graphic cluster, sharing their rendering capabilities; another rendering environment on the tabletop computer; a VICON process, sharing output data from the tracking system; a data provider process, sharing brain scans data; etc. SharedSubstance makes the description of distributed applications

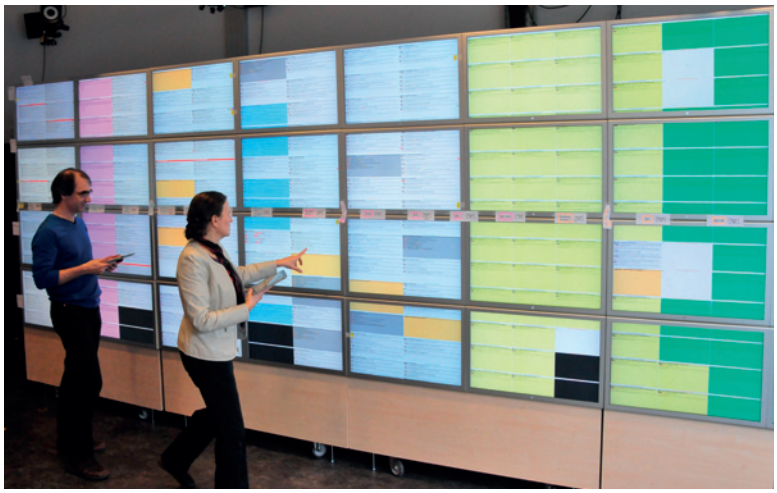


Figure 3.5: CHI'13 Program Going WILD. Wendy & Michel are scheduling sessions for the CHI'13 conference with an *Hydrascope* web application. Content of the 176 sessions is arranged in a 16×11 HTML table displayed on the wall-sized display. They can select and manipulate content with their hand-held multi-touch tablets.

clear and flexible, thanks to efficient mechanisms for dynamic sharing. However, it does not improve the implementation of lower-level capabilities, such as rendering, since they are encapsulated into environments.

MIXED APPROACH: The last approach, the *jBricks* framework, is based on a replication strategy for structured graphics in order to make the distribution of rendering transparent for the developer. However, the distribution of input and interaction is explicit to achieve a high-level of flexibility. A leading principle of *jBricks* is to dissociate the management of input/interaction from the functional and graphical components of applications with two modules: *ZVTM-Cluster* for graphics, developed by Emmanuel PIETRIGA (former Researcher at in|situ|) and Romain PRIMET (Research Engineer at in|situ|), and *WILDInputServer* for interaction, that I have developed. The following presents an overview of jBricks, details can be found in our EICS'11 paper **[Pie+11]** (see full article p. 153).

THE JBRICKS FRAMEWORK – ZVTM-CLUSTER *ZVTM-Cluster* is the module of jBricks implementing the graphical parts of jBrick's applications. It extends the *ZVTM* [Pie05] library with distributed rendering capabilities. ZVTM is an advanced structured graphics library for the Java language. It supports standard Java2D primitives, but also features high-level graphical primitives (e. g. polygons of arbitrary shape, splines, Swing widgets, bitmap images and high-quality text, advanced stroke and fill patterns) as well as advanced mechanisms to ease the implementation of advanced Post-WIMP interfaces and zoomable visualizations (e. g. virtual spaces, cameras, animations, built-in advanced interaction techniques such as lenses). ZVTM rendering is handled in retained mode: the toolkit retains a complete model of the objects to render. This feature was used to extend ZVTM into ZVTM-Cluster: distributed rendering consists of replicating a single virtual space among all cluster nodes, and setting one camera per display configured so that their juxtaposition forms an overall coherent image from the user's perspective. As shown in Figure 3.6, a single instance of the application runs on a *client node*, generating the geometry and distributing it to *render servers* running

on *cluster nodes*. When changes occur in the client node (the master application), it transmits only atomic changes to the appropriate objects on the render servers. Our tests showed that up to 200,000 objects could be rendered at interactive frame rates on the WILD platform. ZVTM-Cluster also enables interactive visualization of very large images, such as the 26 gigapixel panorama of Paris (354 048×75 520 pixels) and *Spitzer*'s Infrared Milky Way (see Figure 3.6), that can be freely panned and zoomed on a wall display. In terms of development, ZVTM-based desktop applications can be adapted to run on a cluster-driven large display by changing as few as four lines of code, mainly to load the platform's geometry and to replace the standard view component with the distributed rendering one. This is the only part of the distributed behavior that a developer has to deal with. Render servers are instances of a generic display program distributed with jBricks that have to be run on cluster nodes. Overall, this enables quick prototyping, development and deployment, as well as the ability to develop outside of the platform.

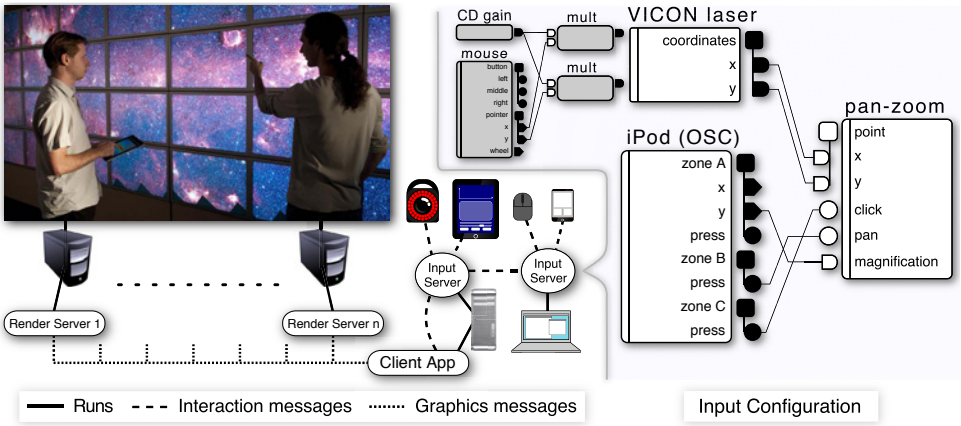


Figure 3.6: Example jBricks configuration: wall's graphics client and input server for motion tracker and tablet run on client node; input server for mouse, keyboard and smartphone run on user's laptop.

THE JBRICKS FRAMEWORK – WILDINPUTSERVER While ZVTM provides extensive support for the graphical part of advanced interaction techniques, it relies on standard Java input and interaction management, meaning that it only handles basic input events from standard devices. This raises a similar issue as previously mentioned tools, such as e. g. Equalizer [Nam+09], where designing and programming advanced interaction techniques for complex interactive environments requires ad-hoc support at the application level, through multiple external libraries and APIs. I addressed this issue in the second module of jBricks, which is the input and interaction manager of the framework, WILDInputServer (WIS) (see Figure 3.7). WIS is a standalone application based on the FlowStates toolkit presented in section 3.1 of this chapter. It makes it possible to use both the data-flow editor of ICON and the state machines programming model of SwingStates in a distributed multi-surface environment.

Thanks to ICON's built-in support for many input devices, WIS handles many kind of distributed input and allows their distribution to several applications. We extended the ICON library to further support generic devices through the most widely used protocols, with specific data-flow devices that can receive and send Open Sound Control (OSC) [WF97], *Ivy* [Bui+02], TUIO [Kal+05] or Virtual Reality Peripheral Network (VRPN) [Tay+01] messages. This approach

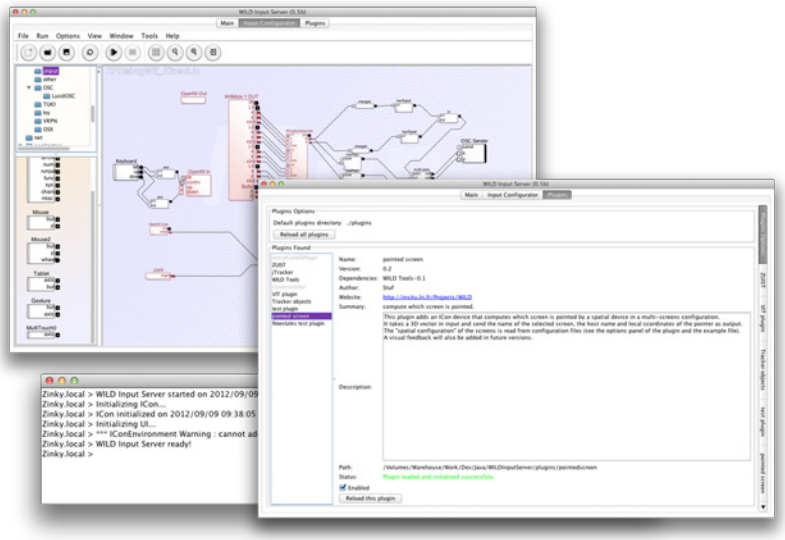


Figure 3.7: The WILDInputServer.

provides a straightforward and implicit way of performing automatic device registration thanks to the addressing mechanism of these protocols: each input source that sends a message addressed to a specific receiving device in a running configuration is implicitly considered. For instance, a WIS' OSC receiver device can listen to messages addressed to `/WIS/position`[4] with two arguments, `x` and `y`. This data-flow device will then externalize the corresponding output slots which will be updated each time a new `/WIS/position` message is received, wherever it comes from: a smartphone running an application that sends OSC messages from touchscreen events, the tracking software of an interactive table, mouse movements from a laptop running another instance of the WIS, etc.

↳*pluggable interactions*

The input manager is easily extensible, thanks to the simple programing principles of FlowStates' state machines. This makes it possible to support new input devices, but also to implement re-usable processing functions or even built-in interaction techniques as data-flow processors. However, FlowStates was originally developed as a library for developing applications, meaning that one should recompile the application after adding a new feature (a new state machine). This is not a viable approach in the jBricks framework, since it would require to change and recompile the input manager all the time. I addressed this issue by adding a dynamic plugin management system to WIS that supports unloading and reloading functionalities at run-time. Plugins are developed with a simple API and can contain several FlowStates' state machines, which will be made available in the input editor without restarting it. They can be used to extend the input server in a persistent way, by simply copying a plugin's jar file in the application folder, or they can be loaded temporarily and installed over the network. For example, an application can implement domain-specific processing devices (e. g. pointing acceleration functions, simulation algorithms, etc.) and install them in WIS through a simple network protocol when it is launched. Combined with ICon capabilities to edit input and interaction configurations at runtime, this makes the overall system highly adaptable.

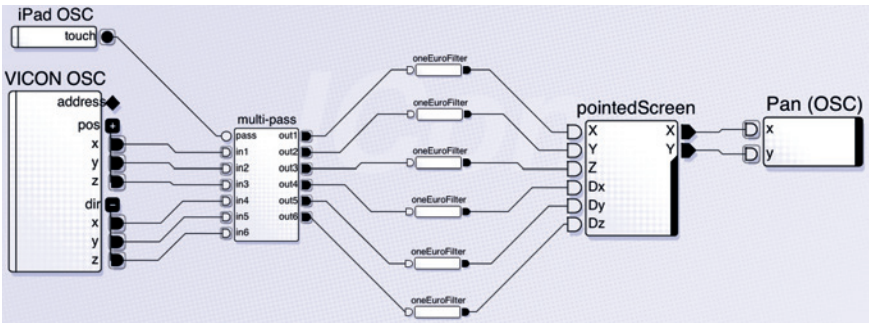
4 The OSC protocol consists of messages containing an "Address Pattern" (URL-like) and optional typed arguments. See the specification: `http://opensoundcontrol.org/spec-1_0`

Input configuration and part of the description of interaction techniques are specified with ICon's data-flow configurations, using the extensive library of adapter devices, e. g. math or logic operators, control structures, flow control, input filtering. These can be used to manipulate and transform the raw values of input channels into higher-level data structures (e. g. the `mult` device in Figure 3.6). The WIS library and default plugins also extend the basic processing devices of ICon with platform-specific ones, adapted to interactive platforms such as WILD. For instance, a "pointed tile" data-flow component returns the display tile that is intersected by a 3D vector received as input (typically modeling the user's arm). More than simple low-level processing components, these higher-level devices are close to the re-usable interaction techniques of MaggLite **[Huo+04a]**, offering several levels of granularity to the user when building an input configuration. They can for example implement the logical part of many advanced interaction techniques – e. g. Marking Menus [KB91], Toolglasses [Bie+93] – independently of their visual feedback and resulting actions.

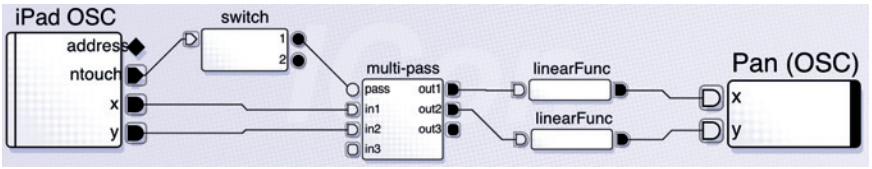
CONNECTING JBRICKS MODULES This higher-level part of the interaction (i. e. visual feedback, manipulation of objects and data) is coded into the client application developed with ZVTM-cluster, since it is the domain-dependent part of the application. It is therefore necessary to connect both parts together: the input configuration and the client application. In order to keep the framework modular and flexible, we chose to support several kinds of coupling, from a loose one to a more tight approach, in a way that is similar to FlowStates "moving frontier" between data-flow – lower-level system control – and state machines – higher-level interaction logic – (see section 3.1). The connection between the ZVTM application and WIS can be established in two ways: with high-level interaction events through network protocols or with plugins. For example, the image manipulation application of Figure 3.6 supports *pan* and *zoom* interactions in order to explore very large images. Pan and zoom operations can be performed for example with a multi-touch hand-held device, as in the figure, or with a 3D motion tracker (e. g. MS Kinect or VICON). In the latter case, applications typically implement a VRPN "tracker" listener, receive raw 3D data from the device, and implement the interaction technique from these raw values. Even if VRPN trackers could be changed at run-time, this raises the issue of controlling the application with another kind of input device. Since the application directly receives raw values from specific input channels (e. g. 3D locations and rotations), controlling it with other input channels (e. g. 2D positions on a touch surface) requires to change the core application code, or to implement a fake VRPN tracker.

We solve this problem by using higher-level interaction events, e. g. through OSC messages, that allow us to choose and adapt the level of abstraction for connecting the input/interaction configuration to the application. In our pan & zoom application example, the application will define its "interaction entry points" as a simple OSC protocol: `/spitzer/pan` with `x` and `y` parameters, and `/spitzer/zoom` with `x`, `y` and `z` parameters. Then, each time the application receives a message, it performs the corresponding action, which is somewhat similar to a Remote Procedure Call (RPC) protocol. On the WIS side, the designer/developer just has to connect built-in OSC data-flow devices as end points of an input configuration in order to create and send these messages to the application. Figure 3.8 gives two example configurations for the panning interaction. In Figure 3.8a, OSC messages are received from both a VICON tracker – the `VICON OSC` that delivers raw 3D positions – and a multi-touch tablet – the `iPad OSC` that delivers boolean touch events. The VICON's output slots are connected to a control device, `multi-pass`, which interrupts the data-flow when the tactile tablet is not touched (the iPad's

`touch` output slot is false). Then, outputs of `multi-pass` are filtered through Casiez et al.'s "*One Euro Filter*" [CRV12], before going into the `pointedScreen` device which is, in summary, a laser pointing method for the wall display. The resulting 2D coordinates are then sent to the application through the `Pan (OSC)` device. Thus, each time the tablet device is touched, the panning interaction is enabled and can be controlled by pointing toward the wall-sized display. Figure 3.8b shows an alternative configuration with the multi-touch tablet only used as a touchpad. It can be built in less than a minute and saved for future use, and can replace the previous one without relaunching the application.



(a) VICON + Tablet

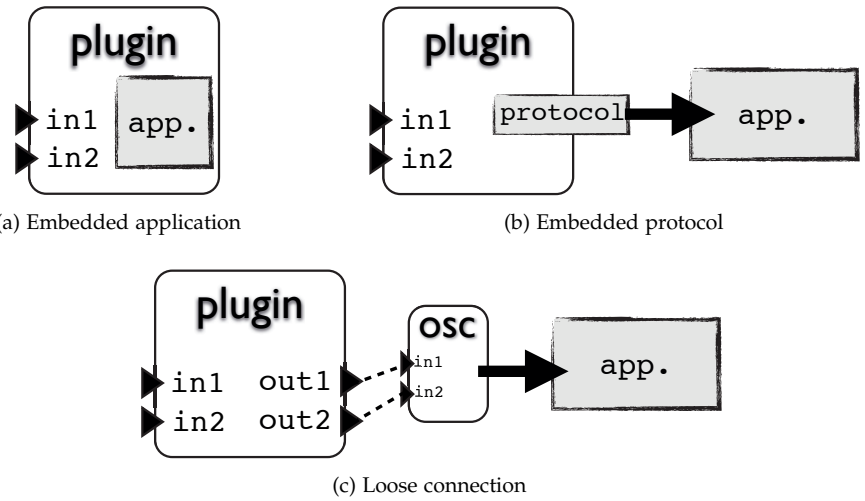


(b) Tablet

Figure 3.8: WILDInputServer Panning Configurations.

A deeper integration of the two modules can be achieved through the plugin architecture of WIS. As explained before, applications can install plugins in the input management system, and these plugins can declare data-flow processing devices implemented via FlowStates state machines. As shown in Figure 3.9, the developer can use this architecture to define several communication channels between the input configuration and the application. In the first case, the application is embedded into the plugin (see Figure 3.9a), so that the ZVTM client application will run in the same Java Virtual Machine (JVM) as WIS. This tight coupling gives direct access to data-flow processing devices and their values from the application code, enabling finer control of the application and avoiding the use of a network protocol. It is however more rigid and makes it harder to handle cases with a high-level of distribution, e. g. an application that should receive input data from several instances of the input management system. The second case, illustrated in Figure 3.9b, consists in embedding only a part of the application in the plugin, typically a dedicated communication protocol, e. g. for performance or security issues. While this is a more flexible approach, the drawback is that it makes the application closed to more generic protocols and thus to other input management systems. The last case consists of using the first connection method, through built-in protocols such as OSC, but to install processing devices that are specific to an application (see Figure 3.9c). While this does not directly connect the input server and the application, these devices can be required for data transformation before sending events to the application (e. g. range transformation, physics simulations, pointing acceleration functions, etc.).

↳*connection with plugins*



(a) Embedded application

(b) Embedded protocol

(c) Loose connection

Figure 3.9: WILDInputServer Plugins and Applications.

This separation of concerns, where the interaction management module is independent of the application code and the application is not aware of low-level input configuration, achieves a high-level of flexibility, which is of great value in environments with distributed multiple input and displays. Several instances of WIS can run on different machines – managing sub-parts of the input configuration – in a transparent way for the main application by creating "composite virtual input devices" that produce high-level input events: the combination of the iPad and VICON in Figure 3.8a can be seen as a unique device. Input and interaction configurations can be modified rapidly at run-time with the visual editor and by non-programmers, e. g. interaction designers, which is well-adapted to rapid prototyping. Furthermore, since the two modules of jBricks can also be run on standard computers (desktops, laptops), the framework makes it easy to prototype outside of the platform by replacing a complex and expensive device, e. g. the VICON system, by an accessible one in seconds (see the illustrating scenario in our paper **[Pie+11]**, included p. 153). At a higher level, the separation between input/interaction and graphics extends the "mixed-graphs" model that I introduced with the MaggLite toolkit **[Huo+04a]** to a distributed architecture. The different levels of coupling between the two jBricks' modules also resemble the FlowStates concept of "moving frontier" between the system and the application: more description of the interaction within the application, which implies lower re-usability but deeper integration, vs. more interaction within the input manager, which implies better re-usability, but requires to externalize more of the application functionalities. This supports the interaction design continuum within a single development framework, from several levels of "live" prototyping to a final implementation, thus "reducing viscosity", "empowering new design participants" and enabling "power in combination", according to Olsen's criteria [Ols07].

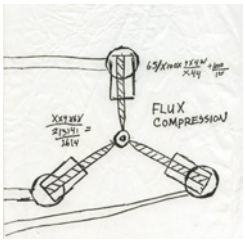
JBricks is widely used for developing applications in the WILD room, by the in|situ| and AVIZ research groups. In particular, Mathieu Nancel used it to implement the novel pointing techniques and controlled experiments reported in his Ph.D. thesis [Nan12]. These techniques are now re-usable in other projects, thanks to the modularity of WIS. Julie Wagner also used jBricks to implement several body-centric interaction techniques during her Ph.D. [Wag12], as well as controlled experiments. Their work highlight both the possibilities of the frame-

work for designing and implementing novel interaction techniques in distributed environments, but also its robustness and reliability for conducting experiments. Finally, WIS is also used independently of the ZVTM rendering architecture in most of the applications running on the platform. Outside of our WILD Room, several researchers have shown strong interest in using it, and it will be included soon in the software framework of the "Inria-VCoRE technological action".

3.2.4 *Going WILD-er*

The general lessons of the WILD project are summarized in our IEEE Computer paper [Bea+12]. In terms of engineering, we created a set of frameworks and applications that we use for demonstration purposes and for our own research. Yet we did not solved nor addressed all the issues summarized at the beginning of this section. For instance, reusing existing applications still has to be investigated further. We have already extended some applications for scientific visualization to be displayed on our wall-sized display, thanks to their scripting capabilities (i. e. *PyMol*[5] and *BrainVISA/Anatomist*[6] for the brains scans application of Figure 3.4c) and we have started to explore the reuse of web applications with Hydrascope. In addition, both the SharedSubstance and jBricks frameworks support the display of existing running application windows (with respectively the Scotty [EBM11] and Metisse [CR05] systems). However, this support is limited to pixels streaming and basic input redirection, and thus does not leverage the high-resolution and advanced input capabilities of the platform. Beyond multiple-device handling and configuration, we did not address specific collaboration issues for groupware, such as roles and conflicts policies, which should be implemented at the application level. However, our architectures already enable sharing, and WIS delivers customizable high-level events that could be used to distinguish easily among users. Collaborative aspects, in terms of both usage and engineering, will be the primary focus of our investigation in our recently launched projects *Digiscope*[7], coordinated by Michel Beaudouin-Lafon, and *Digipods*, which I am coordinating. These projects will create a unique platform for distant collaborative work with 9 heterogeneous advanced visualization platforms, including high-resolution wall displays, CAVEs, large touch and 3D displays, interconnected by a high-end audio-video telepresence system.

3.3 ENGINEERING UNLEASHES INTERACTION DESIGN



↳*"This is what makes time travel possible: the flux capacitor!"*

In this last section, I will discuss the matching between software tools and some of the requirements of interaction design, which relate to the necessity of using "the right tool for the right task" that I mentioned in the introduction of this chapter. Some steps of the interaction design process can require one specific and unique tool. But some other, and especially the implementation of precise prototypes, can require to combine different tools and components in order to test and improve the solutions. The question is then to identify which tools or toolkits to use and how to combine them. In the previous chapter, I already discussed the situation where the required tools – e. g. hardware or software architecture, API or toolkit – are incomplete or even not existing yet. Here, my concern is more to identify the possibilities that engineering of interactive systems offers to interaction design, thanks to the lessons learned during the projects I have summarized in this chapter, and to some relevant related work.

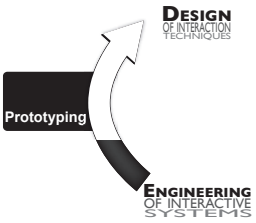
5 see PyMol website: http://www.pymol.org/
6 see BrainVISA website: http://brainvisa.info/
7 see Digiscope website: http://digiscope.fr/

I will identify and discuss three common situations, where "Technology Defines Possible Designs", "Technology Enables The Evaluation Of Design" and "Technology Integrates Design". I do not claim that these are the three only ways to relate interaction design and technology, since this close relationship has already been observed and discussed: by KAY and his early vision of interaction with computers that shaped technological breakthroughs such as *SmallTalk*; by BEAUDOUIN-LAFON, who is advocating the promotion of interactions as first-class objects in toolkits [Bea04]; by OLSEN, who is discussing the notion of *viscosity* [GP96] of UI tools in the interaction design process [Olso7]. My objective is to initiate discussion and to gain a first understanding of the relationships between the two research strands and their mutual evolution. This can open the way to new research perspectives, which will be the topic of the last chapter.

3.3.1 *When Technology Defines Possible Designs*

Interaction design, even with the now commonly adopted user-centered approaches, cannot escape to take technology into account from the very early stages, since the overall objective is to design for users interacting with technology. As discussed in the previous chapter, technology can even inspire interaction design. Considering that, we can now refine, from the technological point of view, the situations discussed in sections 2.3.1 – *When Interaction Design Is Driven by Technology* – and 2.3.2 – *When Interaction Design is Constrained by Technology* – by considering the following attributes for a system:

1. what technology *suggests* is what designers, and sometimes users when involved in participatory design [EK84], will be inspired to ideate for specific needs, based on existing or envisioned technologies.
2. what technology *enables* are the capabilities of hardware, models, architectures, etc. to support the implementation and the transition from an idea to a prototype.
3. what technology *provides* are the already existing technologies or components that could be reused, extended and assembled for the implementation. These could be some of those parts that also "suggested" a new design.

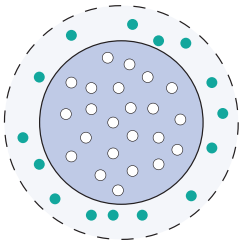


↳*Technology Enables Design*

There is a strong link between what technology *enables* and what it *provides*, which is what OLSEN defines as "power in combination" [Olso7]: the capacity of a UI system to provide a set of basic primitives and to enable their combination to build more complex systems. When the gap is too wide between what technology (and the design process in general) has suggested and what it effectively enables and provides, there is some overhead in prototyping and implementing design alternatives, trying to fix the system, and even sometimes discarding some good solutions, if not the original idea itself. This is what JOHNSON describes in his book *Where Good Ideas Come From: The Natural History of Innovation*, when he draws a parallel between innovation and KAUFFMAN's concept of "*Adjacent Possible*" in natural sciences [Joh10]. The adjacent possible is the set of possible first-order combinations of the atomic parts "in our possession", e. g. electronic parts, raw or factory-made materials, theories, methods, etc. Every new combination extends the boundaries of the adjacent possible, since they should be in turn used as what JOHNSON calls *spare parts*. However, as he illustrates with many examples in the history of innovation and science, even the best idea cannot be turned into a concrete innovation if it goes beyond the limit of the adjacent possible, if the spare parts are not (yet) available. For example, referring again to KAY's *Dynabook* concept, it is clear that it was not in the adjacent possible at the time. Despite many efforts to push the limits and to add more parts, ending

↳*Dynabook mock-up*

up along the way with many concrete innovations (e. g. SmallTalk and Object Oriented Programming, the MVC model, the *Xerox Alto* and its GUI), the necessary adjacent possible was not reached at that time and the Dynabook remained a non-functional mock-up. Conversely, our mobile AR technique for note-taking (see section 2.2.3.1) was obviously less visionary than Kay's Dynabook, and directly inspired by the adjacent possible of actual technology. We can have the same look at HCI technology in general and in particular UI toolkits: What is their intrinsic adjacent possible? How easily can they be extended or combined with other parts? Is there a (more or less) iterative path between their current adjacent possible and the one that will make a design possible? This is especially crucial if we want UI systems to be better *tools* that support prototyping and exploration along the design process.

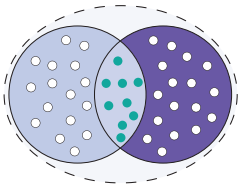


↳*Extending the adjacent possible of a tool*

Existing and widely used UI toolkits – e. g. Java *Swing*, *Qt* –, which are the technological foundation of our current interactive systems, are good tools for what they were designed for, implementing WIMP user interfaces [Bea04]. Some associated development environments, such as Interface Builders, are also well adapted to quick prototyping of WIMP interfaces. But they fail to account for and support what some current trends, models and technologies "suggest" in terms of advanced interaction: direct manipulation, multi-modal interaction, gestural interaction, etc. In fact, these toolkits rarely "provide" the necessary components that "enable" such innovations, which are outside of their "adjacent possible": their atomic primitives (*widgets*) do not capture direct manipulation [Bea04], their event model does not scale to complex interactive systems [Mye91], they have proven to be difficult to extend beyond their initial scope [Cha94]. The past 20 years have seen very active research in UI toolkits, focusing on completely new tools tailored for a particular problem [Mye+90; Mye+97; GF01; DF04; Huo+04a; HMS05; Har+07; Law+09; Mar+11; KMC12; Kin+12b], or on extending existing tools [CR05; AB08; AZ09] for new purposes. The FlowStates toolkit presented in this chapter lies in this second category, since it extends the Java language and event-model to provide designers and developers with relevant new tools: state-based control structures to describe the interaction logic, and a data-flow model and visual language for input and interaction configuration. FlowStates is relevant to be discussed here for two reasons. First, it is a combination of two existing "spare parts" (SwingStates and ICon) which are extending the adjacent possible of each other at the engineering level (combination of models). But at a higher level, as a tool for interaction design, FlowStates' model facilitates and fosters extensibility by considering interaction as a first-class object [Bea04], the toolkit itself following the principles of instrumental interaction [BM00]: data-flow devices *reify* state-machines (or interactions), thus enabling their direct manipulation by the developer, the designer or even the end user through the graphical editor. Interaction techniques can be *re-used* in a *polymorphic* manner (with other objects of interest, etc.). Using the toolkit in this way extends its own adjacent possible: once implemented with FlowStates, an interaction technique or a part of an interaction technique can be reused and combined to create new techniques, and so on. The OpenInterface framework [Law+09] is also a good illustration of this approach, since it provides developers and designers with both low-level tools for extensibility (component model) and high-level tools for their composition (visual editor).

But we also observe that most of these advanced UI toolkits are rarely used by researchers or designers when prototyping or implementing novel interaction techniques, despite they have an interesting "adjacent possible". The first reason is that the adjacent possible is not an "absolute" measure, and depends of

course on the knowledge we have of the tools that exist. In his discussion about innovation, JOHNSON observes that media, and the Internet in particular, have changed the boundaries of the adjacent possible, making existing spare parts easier to find [Joh10]. In fact, searching the web for existing solutions to a problem is nowadays an essential activity in every domain. In our case, finding a UI toolkit that addresses the problem we are facing is most of the time a partial solution. The adjacent possible of a toolkit is also absolute: it depends on the knowledge that the user has of the toolkit, which is likely to be non-existent in that situation, but also on how the toolkit "discloses" its capabilities, as I already mentioned in the conclusion of the previous chapter. For instance, for designing and prototyping the *Gliimpse* technique (see section 2.3.2), we used the standard Java API and Swing toolkit mostly because we did not rapidly find an obvious solution to the first problems we have identified, thus blurring the choice of a potentially better technical solution. We thus started to implement our prototype with a toolkit we are already mastering. Gliimpse was clearly on the edge of the adjacent possible of the technology we used, and its implementation was a challenge. The positive effect is that it helped us better understand how the technique could be implemented, the requirements at the toolkit and application levels, and gave insights into how it could be replicated or help the design of future development tools for similar techniques. On the other hand, this limited our investigations of more advanced designs, such as local animations of only parts of the code. In the worst case, this could even have made the prototype impossible to implement. Overall, a possible explanation of the problem we encountered is the limited "power in combination" [Ols07] of the tools we used, part of what I called the *enable* and *provide* properties: the main issue in Gliimpse was to construct mappings and combine things together, whether they were provided by a single tool or from several ones, i.e. editor and document rendering components and markup language interpreters and animation engine.



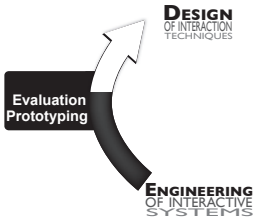
↳*Combining tools for a new adjacent possible*

Extending the overall adjacent possible of a whole "ecosystem" (i.e. an ensemble of tools) requires power in combination between the different tools that are making it. This is what we tried to achieve with the jBricks toolkit for multi-surface environments (see section 3.2), by decoupling input and interaction management from the graphical application, while providing efficient mechanisms to compose them in a generic way, with several levels of integration. As a result, the two tools benefit from each other, but can also be used independently, with other tools. This might support several phases of interaction design: from several prototypes, where multiple alternatives can be tested with different combinations of tools, to a more robust implementation where the components can be integrated deeper.

3.3.2 *When Technology Enables The Evaluation Of Designs*

The HCI community has a long history of discussing methods and tools for research and practices in HCI, especially on the hot topic of *evaluation* of novel interaction techniques and interactive systems in general. GREENBERG and BUXTON give an overview of the situation in their 2008 position paper [GB08], advocating for a more grounded and better informed choice of evaluation methods according to the design and research problems, instead of blindingly running usability studies (i.e. controlled experiments). Related to this debate on the role and relevance of evaluation in HCI, the community is currently trying to promote "replication" (of studies, systems, methodologies) as one of the ways to ground the field (e.g. two *RepliCHI* panels were held in 2011 and 2012 [Wil+11; Wil+11], and a workshop will be organized at CHI 2013[8]). My goal here is not to take side

8 See the CHI'13 RepliCHI workshop website: http://www.cs.nott.ac.uk/~mlw/replichi.php



↳*Technology Enables Evaluation of Design*

in these debates, but to emphasize how engineering, and better technology, can support evaluation when necessary. Greenberg and Buxton [GB08] also promote the use of the right tool for the right task, but at the methodological level. Some of these tools for the evaluation of interaction design, such as controlled experiments or field studies, need robust functional prototypes and systems, with additional functionalities related to the evaluation protocol itself [MHP00]. The issue is then, again, to have access to and to use the right technological tool(s).

This implies that the underlying technology satisfies several requirements. An evaluation is always evaluating the "whole thing": the apparatus and the user. To ensure that measurements, and thus the results, are not corrupted, we need to control the behavior of the system and prevent potential problems such as performance issues, crashes, participant loss of concentration due to system failures, etc. Performance, robustness, reliability and predictability are standard objectives of software development in general, but they are harder to achieve when prototyping advanced interaction techniques that challenge technology and are rarely based on well-tried software engineering methods and tools. Our jBricks framework has proven to be reliable and efficient enough to conduct highly-demanding experiments with the WILD platform, as mentioned before. One of the benefits of jBricks in this context is again its modularity, which helps to identify and fix problems (e. g. live data monitoring with WIS, on-the-fly disconnection and reconnection of modules to identify flaws) as well as the ability to alternate quickly between experimental conditions and to re-use existing blocks from one experiment to another, thus favoring replication.

Beyond implementing the prototype of the interactive system, another requirement is to ease to conduct the experiment, with appropriate mechanisms for instrumenting the prototype and running the experimental protocol: displaying instructions, moving to the next trial, performing measurements and logging, etc. Except for some low-level logging mechanisms, these features are not present in standard interactive systems, since they would be of little use in normal use, so they need to be implemented for every new experiment. Some frameworks were developed to ease this process, in particular the *Touchstone* platform developed at in|situ| [Mac+07]. Touchstone was, however, designed to conduct controlled experiments on standard computers, not on mobile devices or distributed multi-surfaces platforms like WILD. The evaluation of the mobile techniques presented in sections 2.2.1 and 2.2.2, that I have conducted before the release of these frameworks, required to implement all these mechanisms from scratch (taking into account the platform limitations) and to test their reliability before conducting the actual experiments. I extended Touchstone with a simple but efficient network protocol, enabling communication between the mobile device or distributed application running the prototype, and the Touchstone platform running the experimental protocol and the logger on a standard computer. We can thus focus on the implementation of the techniques to evaluate on the mobile or distributed environment. We conducted the evaluations of BiPad (see section 2.2.3.2) and mobile AR techniques (see section 2.2.3.1) with this platform, extended with a jBricks' WIS configuration for the mobile AR setup (to enable three-way communication between the hand-held device, the physical control board and Touchstone, see implementation details in Can Liu's thesis [Liu12]). This has proven to be reliable, and reduce development time in comparison to the implementation of an instrumented prototype on the mobile device.

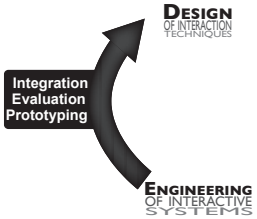
Finally, as stated by Greenberg and Buxton, "*usability evaluation, as practiced today, is appropriate for settings with well-known tasks and outcomes. Unfortunately, they*

fail to consider how novel engineering innovations and systems will evolve and be adopted by a culture over time." [GB08]. This stresses that evaluation of interactive systems should also capture their adoption by users on a more long-term use, but also how they are effectively used, sometimes adapted in more convenient or unintended ways. An example is the field study we have conducted on left-over windows (see section 2.1.3), which helped us to identify patterns of use of a technology and corresponding users' behaviors. Such studies are long and difficult to design and conduct, but can also raise many technological issues: the studied techniques, and possible additional software (tools for the study), must be reliable enough for real use, with real users, and without supervision from an operator. In our case, we also faced problems because our study was investigating an existing system, not a design prototype, and implementing the event logger was the longest phase of the project. We used several APIs (e. g. Cocoa, AppleScript Access, Accessibility) and designed several algorithms to identify windows' and their states, to maintain lists of open/reduced/left-over windows, to access low-level events, etc.

3.3.3 *When Technology Integrates Designs (Or Not)*

Designing novel interaction techniques and interactive systems, understanding and explaining "interaction phenomena", defining and operationalizing theories and models are satisfying for us as scientists. Another reward is to see the results from our field spread into the "real world", as an ultimate assessment of their relevance and usefulness. This is however rarely the case, and even the small number of technologies that succeed in getting out of the research labs rarely do so in less than 20 years. The history of HCI technology is riddled with many well-known examples that I have already mentioned. KAY advocates that very novel ideas come from changing points of view [Kay89], and BUXTON points out the need of iterative augmentation and refinements before a good invention eventually turns out to become an innovation [Bux08]. We all have in mind the impressive number of inventions by XEROX PARC scientists that turned into real innovations in the 70's, and sometimes in about 10 years: the *Xerox Alto* (1973) inspired the design of the *Apple Lisa* released in 1983, followed by the *Macintosh* one year later.

Setting aside marketing and business concerns and considering only technology, the context of HCI in the 70's was very different from now: everything was still open to design and these pioneers almost completely shaped modern interactive technologies. The "adjacent possible", as discussed before, was small and many good ideas and breakthroughs were needed to push its limits. But we are now in a different situation, were the adjacent possible has been extended and is fortunately still growing and opening the way to many new designs. The counterpart is that it might also make new ideas and major advances more difficult to envision. In addition, and probably more important, they are more difficult to transfer into the "real world", since they must be integrated with operating systems and higher-level paradigms – e. g. WIMP, event-based architectures, UI toolkits – that have not really changed since they were conceived. This "installed base" prevents a radical change of interactive systems, and also limits the integration of novel interaction methods. In fact, the adjacent possible not only comes from the available parts, but also from the possibilities of assembling those parts, what I called "what technology *enables*", echoing OLSEN's "power in combination" [Ols07], BUXTON's "refinements and augmentations" [Bux08] and BEAUDOUIN-LAFON's "reinterpretability" [Bea04].



↳*Technology Enables Integration of Design*

So, how could we offer better integration of our "inventions" into the current ecosystem of interactive technology? The first solution is when integration is straightforward, by the means of the Operating System (OS) and/or adapted toolkits. But the further we get from the WIMP paradigm, the more unlikely it is to occur. The second solution is to account for the properties of the target system from the early stages of design, as illustrated with the TorusDesktop pointing facilitation technique in section 2.3.1. This trade-off might drastically reduce the design space, requiring to balance between the benefits of possible designs and their feasibility given the constraints of the technology. The drawback is obviously to discard better alternatives from the start, but this risk should also be assessed in terms of related work. In the case of TorusDesktop, target-aware techniques are supposed to be better, were already explored in depth, but none of them has been incorporated into an existing interactive system. The third solution, illustrated by Gliimpse, is to target an ad-hoc but advanced prototype, half-way to the "final" version, but comprehensive enough to convey the innovation. As discussed in section 2.3.2, this *high level of viscosity* [Olso7] implies to hack and tweak, as well as to reinvent parts of interactive systems. While enabling the evaluation and revision of underlying technology, it only provides "surface"-level integration (in a standalone application) where a deeper integration could durably extend the adjacent possible: For instance, a built-in animation engine between heterogeneous data and objects at a system- or toolkit-level[9] would have been of great help to implement Gliimpse, but also to explore a similar solution in other situations. Again, such an implementation can in turn inform the design of a more general approach, and might help implement it in a toolkit, but it is unlikely to scale to the system level.

As a fourth solution, research has produced several systems or toolkits that facilitate the integration of advanced interaction into real systems. The most complete and advanced one is probably *Metisse*, developed in the in|situ| group by Olivier Chapuis and Nicolas Roussel. Metisse is a redesign of parts of the X Window system in order to support advanced window management techniques in a standard Linux graphical environment [CR05]. Metisse "intercepts" and "redirects" the rendering and input events of application windows in order to recompose windows and modify interaction in various ways. It has been used to implement existing advanced window manipulation techniques [Bea01], to create and experiment with many novel techniques in a real environment, with legacy applications [Stu+06; CR07; CLP09; FCR09; CR10]. Metisse has even been included in a Linux distribution (Mandriva Linux in 2007), and used for game development. Higher level tools help achieve the integration of novel post-WIMP techniques by "hijacking" and interpreting pixels rendered on the screen [DF10], or by leveraging introspection mechanisms of some UI toolkits and "injecting" new code into a legacy application at run-time [EBM11]. The integration is weaker than with Metisse, but powerful enough to allow testing advanced techniques which otherwise would have been impossible or cumbersome to implement in a real environment.

These examples show that improving the support for integrating novel techniques into existing systems requires an increased level of "openness" in terms of access to resources (e. g. input, pixels, etc.) and objects (applications, widgets, etc.), as well as their level of extensibility. All the approaches described above consist of intercepting system resources more or less deeply in order to make these resources more accessible: in short, they plug into "open closed systems" wherever they can.

9 For instance, Apple APIs provide such built-in animation mechanisms, but between predefined objects attributes of the same type.

ICon [DF04] and WILDInputServer (see section 3.2.3) also follow this principle for input resources, therefore facilitating the integration of multiple-device interaction techniques with limited effort and increased configuration capabilities. However, these are only "fixes" and should be part of low-level system architectures to exploit the full potential of advanced interaction in real conditions of use.

3.3.4 *A Missing Link Between Interaction Design and Engineering*

Now that we have studied both parts of the "iceberg" it should be clear that they both influences each other: On the one hand, designing advanced interaction both leverages and challenges interactive systems and tools; On the other hand, UI tools and technologies both support and inspire interaction design, but also limit what is possible. It is this strong coupling that I call the loop of "*Designeering Interaction*" and that I explore in the next chapter.

"And if the gap between creative design and computer tools was a result of the poor creative power of the tools for interaction design? Exploring, proposing and evaluating solutions, questioning conventions and established knowledge, this is how we defined creative activities. These are also the activities that should be supported by future tools for interaction design, so that the limitations of applications would not reflect the limitations encountered by their developers or their designers, but the limitations defined by their users."

Stéphane Huot – Ph.D. Dissertation (2005) [Huo05].

4

DESIGNEERING INTERACTION

The introduction of the previous chapter asked: "*can a designer/developer easily implement a novel zoomable interface with the* ZVTM *toolkit [Pie05], describe its interaction logic with state-machines through the* HSM *toolkit [BB06] and support mark-based [AZ09] and multi-touch [Kin+12a] input?*". One way to answer this question is the extent to which these tools put the system to be designed within the adjacent possible. Chapter 2 illustrated various situations: Gliimpse was on the edge of the adjacent possible; BiPad was in the adjacent possible; FlowStates was intentionally designed to extend the adjacent possible. However, this is a retrospective analysis and the problem lies in the designer's a priori knowledge of the "adjacent possible" of the tools at hand. Exploring the adjacent possible often requires a trial-and-error strategy focusing on technical issues only, rather than design issues. While this makes sense when engineering new interactive systems or toolkits, it does not support interaction design as a creative activity very well.

I already addressed this tension between UI technology and interaction design during my Ph.D., pointing out the gap between the creative nature of interaction design and the creative power of the tools we use. Research in HCI has produced many toolkits, addressing many different issues in various ways. But as illustrated in the previous chapters, designing advanced interaction still remains too often a process full of technical pitfalls, requiring to hack systems, to master low-level technologies and to combine them together [OK05] in ways they were not expected to be combined. From a technological point of view, we are in a situation quite similar to the Do It Yourself (DIY)/*hackerspace* community a few years ago: Testing and implementing prototypes was required deep knowledge in electronics to assess the feasibility of the idea and to design sometimes complex circuit

diagrams before assembling low-level components. The *Arduino*[1] technology now makes it possible for anyone, with a minimum of knowledge, to invent, prototype and explore: Such tools make technology affordable and more accessible by encapsulating complexity and enabling rapid combination, thus better supporting creativity. Even though some ideas will always be on the edge, it makes it easier to assess and explore the adjacent possible of this technology. The only tools similar to Arduino that we have in HCI are those dedicated to WIMP interfaces, based on the combination of widgets. As I already discussed in section 3.3.1, they hardly support new forms of interaction. On the other hand, the profusion of specialized Post-WIMP toolkits with different abstractions does not ease their combination. These challenges were already pointed out in 2000 by MYERS et al. in their paper "*Past, Present, and Future of User Interface Software Tools*" [MHP00]. Most of them are still on today's research agenda.

Through the lens of JOHNSON's adaptation of KAUFFMAN's "Adjacent Possible" theory [Joh10], I identified some attributes of UI toolkits as some of the "spare parts" for interaction design: what they *suggest*, *enable* and *provide*. The challenge is now to design a framework that captures these attributes in order to inform designers on the adjacent possible of existing toolkits, and to identify the requirements for designing future ones. This chapter gives a first sketch of what this "*Designeering Interaction*" conceptual framework could be. It also discusses several future directions for research on the operationalization of the framework, as well as the design of tools that support the concept at several levels: system and programming level, prototyping level, and end-user level.

4.1 THE CYCLE OF DESIGNEERING INTERACTION

↳*"We shape our tools and thereafter our tools shape us"* – M. MCLUHAN, *1964*

The goal of the concept of "Designeering Interaction" is to get a better understanding and a definition of the relationships between interaction design and the technology that it uses, mainly software tools as a first step, in order to inform both the design process (especially prototyping) and the engineering of new tools. Improving existing tools and shaping new ones is an inherent part of many, if not all, Human activities (e. g. crafting/manufacturing, cooking, hunting). As a first step towards the Designeering Interaction framework, I take a closer look at the relationships between the two.

In chapters 2 and 3, I discussed these relationships separately, from the interaction design and the engineering points of views. From a more general perspective, they constitute a cycle, as shown in Figure 4.1. This is well illustrated with multi-touch technologies (section 2.3.4), if we set aside the emergence of the initial technology: multi-touch features and limitations suggested new designs that technology made possible to *prototype* and to *evaluate*; in turn, these new designs led to evaluate and to *revise* and *improve* technology. We can expect that in turn, these improvements to the technology will make it possible to *prototype*, *evaluate* and *integrate* new designs, which will again inform on necessary *revision(s)* or possible *extension(s)* of the technology.

Figure 4.1 presents the cycle of designeering, linking the relationships between interaction design and engineering. Some of the relationships may not be present in all situations, depending on where a prototype lies with respect to the Adjacent Possible boundaries:

1 See the Arduino web site: http://www.arduino.cc/.

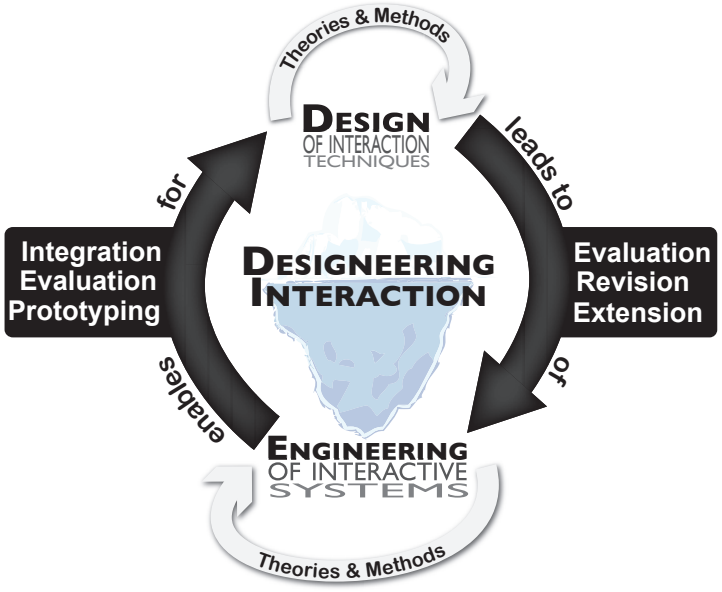


Figure 4.1: The infinite loop of "Designeering Interaction".

INSIDE If the prototype is inside of the adjacent possible, we are likely to observe only "engineering ⇒ design" relationships: TorusDesktop or BiPad (see chapter 2), for example, were inside the adjacent possible and did not really challenged technology. Designing prototypes inside the adjacent possible means that we have the right tools, and this might not lead to the revision of existing tools or the engineering of new ones.

BORDERLINE For prototypes that are near the boundary, some "engineering ⇒ design" relationships may be missing but most or all of the " design ⇒ engineering" relationships will be present. These borderline problems are the most interesting, since they make the loop more efficient and are likely to significantly extend the Adjacent Possible, thus paving the way to systems outside of it. However, as mentioned in the introduction of this chapter, engineering should not be the single focus of attention. Some examples include Gliimpse, which I already discussed in this document, and the *CPN2000* project which, by studying principles on the edge of the adjacent possible, has resulted in many findings in terms of technology [BL00], interaction techniques [Bea+01] or theory [BM00].

OUTSIDE If the prototype is beyond the adjacent possible, the "engineering ⇒ design" section is likely to be nonexistent and is unlikely to engage the "design ⇒ engineering" part of the cycle. In this case, intermediate goals that lie within the edge of the Adjacent Possible are required to make progress, but with no guarantee that they will lead to the right result (e. g. Dynabook).

It is unlikely that we can define the exact "path" for designs that lie outside the Adjacent Possible. However, estimating whether the system to be designed is inside or not, and using tools that drive the designeering loop faster could ease the exploration of design spaces and the improvement of technologies, thus helping to extend the boundaries of the Adjacent Possible faster.

This discussion has addressed the concept of an overall Adjacent Possible in UI technologies. Its atomic parts are the theories, models, methods and technologies that we assemble together to create new designs. We can recursively apply this

concept to these parts, and especially to UI technologies and toolkits. If we could operationalize the concept of the Adjacent Possible of a toolkit, as well as their possible combinations, we could in turn better estimate the feasibility of a new design.

4.2 TOWARDS A CONCEPTUAL FRAMEWORK

Defining the Adjacent Possible of a toolkit requires to think in terms of "combining atomic parts". These atomic parts constitute the *provide* attribute of a toolkit that I defined in the previous chapter. Generally speaking, toolkits provide developers with abstractions and a basic set of concrete implementations of these abstractions. For example, all WIMP toolkits are based on two abstractions, the *widget* [MA88] and the *events/callback* mechanism (e. g. `Listeners` in Java), and they all provide a set of default widgets.

The atomic parts of Post-WIMP toolkits highly depend on the level of abstraction and generalization of the concepts that the toolkit implements. For instance, toolkits for stroke-based interaction, such as *SATIN* [HL00] or *SST* [AZ09], provide developers at least with *gestures recognizer(s)* and *strokes interpreter(s)* as their atomic parts. These components are also likely to depend on the data they manipulate, e. g. *strokes* and *gestures*, that should be considered atomic parts as well. In this example, the atomic parts are components dedicated to a particular task or interaction method. Conversely, some toolkits provide lower-level and generic atomic parts that do not encapsulate a specific interaction but instead an abstraction to describe interaction. This is the case of *FlowStates*, whose atomic parts are *state machines*, *data-flow processing devices* and the *abstract events* they exchange. Another example is *Proton++* [Kin+12a], which provides a way to describe multi-touch gestures with regular expressions.

These atomic parts must then be assembled. The attribute that I called *enable* in the previous chapter represents the set of operations that the developer can do with the atomic parts, mainly their *creation*, *extension*, and *combination*.

EXTENSION Extension or implementation in toolkits is often achieved by inheritance from the abstract concept or from an existing concrete atomic part. The level of extensibility is hard to assess, as it depends on the level of abstraction of the provided atomic parts. For example, in the Java Swing toolkit, slightly changing the behavior of a widget (e. g. adding a roll-over effect on a button) is fairly easy to achieve with inheritance, but implementing a widget from scratch is difficult (as in *Motif* [MHP00] and most WIMP toolkits). Conversely, with FlowStates, extending an existing state machine is not easy and can even be problematic at a conceptual level (e. g. when redefining an existing transition), but creating a new one is straightforward.

COMBINATION For the combination of atomic parts, I propose to distinguish between two cases, depending on whether the combination is "*structural*" or "*logical*". Structural combination consists in assembling atomic parts in a way that do not create a *synergy*[2], a new interaction technique. For example, WIMP interfaces are purely structural combinations of widgets, guided by the structure of the resulting interface. Even though some logic will be defined as responses to user's actions on the widgets (e. g. changing the content of a widget when clicking another one), this is the logic driven by the application and does not define a new

2 In systems theory, synergy refers to interactions between elements that produce an effect greater than the sum of their individual effects.

form of interaction. Similarly, in the case of stroke-based interaction, chaining a shape interpreter with a recognizer is a structural combination. Conversely, a logical combination creates new "spare parts" with different or extended capabilities. For instance, a Marking Menu [KB91] is a combination of a Pie Menu [Cal+88] and a gesture recognizer. This type of combinations is relatively easy to achieve with FlowStates, sometimes without even writing code: provided that the two blocks have already been implemented, they can be connected in the visual editor. Logical combinations are for example the main principle of the *Ubit* toolkit [Lec03] whose "*brickgets*" extend the widget concept to extensible and combinable bricks of lower granularity (e. g. behavior, graphics). Logical combinations are more powerful than structural ones, especially for prototyping, since they define new interaction techniques and not only a set of co-existing techniques. In other words, structural combinations remains inside the Adjacent Possible of a toolkit, whereas we can expect logical combinations to extend it. However, enabling logical combinations is more difficult to achieve since it requires an adapted model that accounts for modularity and interoperability of the atomic parts.

REUSE An important property to consider for both the extension and combination of atomic parts is how the toolkit allows to *reuse* these newly created parts. This is related to the "design ⇒ engineering" phase of the designeering cycle, when interaction design extends the adjacent possible of technology, making future explorations easier and faster. Most toolkits let developers reuse extensions of atomic parts, typically through the features of the underlying programing language (e. g. Object Oriented Programming). Reusing combinations is more complex since it requires combinations to be also considered as atomic parts, and therefore to be compliant with the abstractions and the model of the toolkit. In WIMP toolkits, this is usually achieved by encapsulating several widgets into a parent widget. Reusing combinations in toolkits for advanced interaction is in general more difficult to achieve because of their specialization. For example, a combination of custom gesture recognizers designed with a toolkit for pen-based interaction is unlikely to be reused with a 3D motion capture system, even though it is a very similar problem. Overall, reusing extensions and combinations is often limited to the sole purpose they were originally made for, which is of limited interest for the exploration of new designs. In fact, most UI toolkits have poor support for *polymorphism* [BM00], since the abstractions they provide are most of the time highly dependent on their application domain and the underlying data.

INTEROPERABILITY Finally, combination and reuse is even more difficult to achieve when combining elements from different toolkits. This lack of interoperability between toolkits is one of the main limitations for rapid and iterative prototyping of advanced interaction. As discussed before, we now have many different tools that address specific problems, but they are difficult or impossible to use together. Different toolkits are based on different abstractions and models, requiring to deal with several programing paradigms, abstractions, data structures and even sometimes programming languages within the same prototype. For example, combining the four toolkits listed in the introduction (*ZVTM*, *HSM*, *SST* and *Proton++*) would require to: (i) implement Java bindings for the HSM and Proton++ toolkits (written in C++), which is not straightforward and could seriously limit the accessible functionalities of the toolkit; (ii) handle multiple and different events mechanisms from SST (marks gestures) and Proton++ (multi-touch gestures), and connect them to HSM state machines; (iii) link HSM state machines to ZVTM functionalities. This requires specialized programing skills and is time consuming: it might be easier to develop an ad-hoc solution from

↳*toolkits are good to do what they were made for, and to make us do what they were made for.*

scratch, even though it is unlikely to be reused. However, being able to leverage the capabilities of these toolkits together for a new purpose would enrich the adjacent possible of both interaction design and technology.

Current solutions to overcome this lack of interoperability mainly consists in external mechanisms that "glue" toolkits together. Distributed components such as Java Beans or Service Oriented Architecture (SOA) [VCT12] have been used as mechanism to externalize some toolkits components and functionalities. This is however constraining for prototyping since it imposes to follow specific patterns and conventions. Simpler and lighter protocols, such as *Ivy* [Bui+02] and OSC, come from the domain of computer music and are more and more used in interaction design as tools for connecting heterogeneous modules of an interactive system (as we did in our distributed environments for WILD). The problem is that only a few toolkits, if any, embed these types of mechanisms for interoperability, and so they have to be added at the application level. Finally, another unresolved issue for interoperability is to deal with the different concepts and abstractions manipulated by toolkits, which always require an adaptation layer when combining them. A good example of interoperable tools are web technologies such as HTML, JavaScript and Cascading Style Sheets (CSS), since they were designed to manipulate the same abstraction: the Document Object Model (DOM). Beaudouin-Lafon's proposition to promote *interaction* as first-class objects in toolkits [Bea04] is a promising way towards better interoperability, provided that they have a similar abstraction of what interaction is.

4.2.1 *Extension and Operationalization of The Framework*

The *provide* and *enable* attributes are a first attempt to assess the intrinsic adjacent possible of a toolkit in order to estimate the feasibility of a prototype. They refine Olsen's "power in combination" [Ols07] criterion, and could help choose the right tool for a given problem. The four dimensions of the *enable* attribute, i. e. *extension*, *combination*, *reuse* and *interoperability*, also define important features for toolkits to support the designeering cycle in both directions by promoting the facilitation of prototyping and the extension of the technology. I believe that these attributes can help rethink the notion of a toolkit since a common abstraction for Post-WIMP interaction, similar to the widget for WIMP interfaces, is unlikely to exist. Following these principles might help create better tools for interaction design, not just a set of predefined building blocks, but interoperable frameworks to create, reuse and combine the blocks needed for the task at hand.

However, these attributes and their dimensions need to be refined and investigated deeper in order to operationalize the framework, compare existing tools, and develop new ones. A first step is to study and classify existing tools for interaction design with respect to these attributes to assess their relevance. Table 4.1 shows a preliminary coarse-level assessment of these attributes for the *Java Swing*, *Proton++* and *FlowStates* toolkits.

Additionally, as discussed in section 3.3.1, what the toolkit *suggests* has to be in line with what it actually *provides* and *enables*. This is however a complex problem, since what the toolkit suggests is subjective. It might vary from one developer to the next, depending on their knowledge and experience of a particular toolkit and of what Myers et al. call "*threshold and ceiling*" of a toolkit: "*The 'threshold' is how difficult it is to learn how to use the system, and the 'ceiling' is how much can be done using the system.*" [MHP00]. Finally, it should also depend on the visibility of what the toolkit *provides* and *enables*, and the possibility to assess these attributes

↳*the guy with an infinite adjacent possible*

easily. For example, we can hypothesis that *Visual Languages* better exhibit these attributes, by making the atomic parts visible in a library of components and manipulable directly in an editor. An interesting research direction would be to define a measure for the "level of disclosure" of a toolkit, inspired by GAVER's technology affordances [Gav91], in relation to its *provide* and *enable* attributes.

Provide		**Java Swing**	**Proton++**	**FlowStates**
Atomic parts:	abstract	yes	yes	yes
	concrete	yes	no	yes
Enable				
Extension:	inheritance	good	poor	poor
	creation	poor	good	good
Combination:	structural	good	good	medium
	logical	poor	poor	good
Reuse:	extensions	good	medium	good
	combinations	medium	medium	good
Interoperability:	internal	medium	good	good
	external	poor	medium	good

Table 4.1: The *Provide* and *Enable* attributes of a UI toolkit, illustrated with three examples: the *Java Swing* WIMP toolkit, *Proton++* for multi-touch gestures and *FlowStates*.

4.3 TOOLS FOR DESIGNEERING INTERACTION

While not fully operationalized yet, the "designeering" framework already suggests some interesting directions for the engineering of new tools that better support the "designeering cycle": tools that enable *prototyping*, to *evaluating* and to *integrating* novel interaction techniques, while being modular and flexible enough to be *revised* and *extended* with these new designs. While some research trends are adopting an "horizontal" approach, considering the whole life cycle of interactive software development at multiple levels (programming languages, libraries and toolkits, software architectures, prototyping) [Cha08a; Cha08b; Let+10], my research strategy is to follow instead a "vertical" approach, focusing on interaction design and prototyping. By developing and studying tools that support the concepts of "designeering interaction", my goal is to enrich the framework which in turn will help understand the necessary improvements to the tools in an iterative co-evolution process. In this section, I briefly discuss what this tools could be at three levels: "system and programming", "prototyping", and "end-user level". These are the three main research directions that I have started to explore in my current projects or that I plan to initiate in the near future.

4.3.1 *System & Programming Languages*

Interactive systems, e. g. desktop computers, mobile devices or distributed platforms, are built on very similar architectures and operating systems. They mainly consist of a stack with five layers: Hardware & Peripherals, Kernel (CPU and memory management, file system, drivers, etc.), System libraries, Application Frameworks (toolkits) and Applications. Each layer accesses and encapsulates

the resources from the layer underneath and provides the upper layer with the higher-level functionalities it is supposed to need. This encapsulation and abstraction of low-level resources is partly the result of the standardization of development processes, which, in terms of interaction, is based on the wide acceptance of the WIMP paradigm. At the application framework level, WIMP toolkits provide developers with widgets and high-level input events (mouse and keyboard) so that prototyping or implementing advanced interaction is hard to achieve if the required functionalities are not already implemented at the toolkit level. The only solution is to go down the levels until the desired features can be accessed, which most of the time requires shifting to another programing paradigm and/or language. Common examples that are still issues today include accessing advanced channels from input devices (e. g. pressure from a pen tablet), which requires going down to system-level libraries and manufacturer's API, or managing multiple mouse cursors, which requires implementing custom drivers [BL00] or fake cursors accessible only at the toolkit level [HB99; Lec03; DF04; TG04]. The same applies to programming languages, which were designed to describe computation, not interaction [Cha08a; Bea08].

While the HCI community argues that systems architectures and programming languages must be revised to better account for interaction [MHP00; Cha08a; Bea08], we will probably not have the opportunity to revolutionize or even just reshape the core of operating systems to better account for interaction as the central issue of computing. First, it would require a lot more time, efforts and resources than the HCI community is willing to spend. Second, except for experimental purposes, and even in the case of technical success, it is unlikely to be adopted by industry if we consider the current installed base of "standard" systems. Renouncing to design an "Interaction Machine", we are thus "hacking" systems, in order to experiment with new interactive technologies and to promote their integration in real setups. We can distinguish between three levels of extension: system and libraries, programming languages, and application frameworks. For example, Metisse [CR05] illustrates low-level system modifications that enable advanced interaction in real window managers. SwingStates [AB08] or I* [Cha08b] are libraries that extend programming languages and their concepts to better handle interaction. At the application framework level, ICon [DF04] gives high-level access to low-level resources (input devices), thus avoiding developers to manage this level of complexity.

As discussed in the designeering framework, the problem is again the interoperability between these tools at several levels. An interesting research direction is to define a common low-level language or library that accounts for the combination and interoperability properties of the designeering framework. It could be used as a transversal and universal access point to the different layers of systems. I have started to explore such a low-level interaction library based on the *erlang* programming language [AVW93]. Its built-in management of concurrency, message passing paradigm and hot swapping of code are powerful properties for combination and interoperability. In fine, the goal is not to make such a low-level library a multi-purpose development tool for advanced interaction, but a core library that would support combination and interoperability of higher-level and more focused frameworks and toolkits.

4.3.2 *Creative Prototyping: Sketching Interaction, not Interfaces*

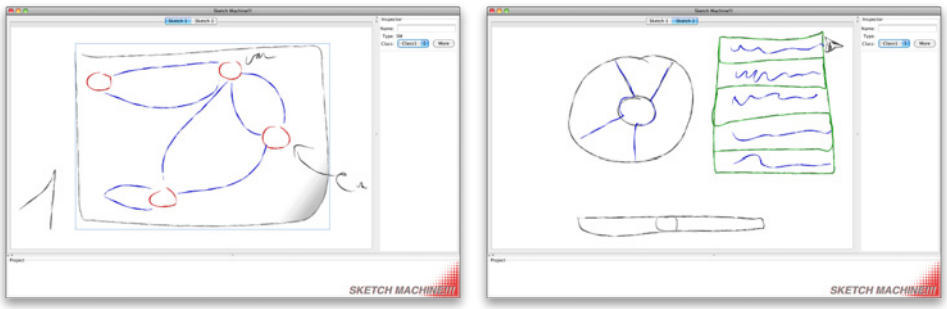
Tools for rapid design of user interfaces appeared with the rise of the WIMP paradigm. Noticeable Interface Builders include the pioneering *HyperCard* [Goo88],

and the widely used *VisualBasic*. However, even the more recent Interface Builders are limited for specifying behaviors of an interface beyond simple predefined actions on widgets, and rely on standard event-based programming for the logic of interaction. Tools based on hand-drawn sketches have also been studied as a way to bridge the gap between early design phases and functional prototypes, such as *SILK* [LM95] for WIMP interfaces or *DENIM* [New+03] for web sites. But the definition of interaction is also limited to simple behaviors (e. g. transition to another form when pressing a button). For novel forms of interaction, the diversity of input and interaction methods has led to various domain-specific authoring and prototyping tools such as: the *Phidgets* API [GF01], which pioneered easy development of physical interfaces by encapsulating low-level communication with electronic devices into high-level reusable blocks; *d.tools* [Har+06], which extends and generalizes this principle with an integrated environment based on visual programming for the design, test and analysis of applications for "information appliances" and physical interfaces; *Exemplar* [Har+07], which is based on Programming by Demonstration (PbD) for programming sensor-based interactions; The *OpenInterface* framework and its *SKEMMI* editor [Law+09] to reuse and connect components for advanced interfaces in a data-flow editor; *DejaVu* [KMC12], an Integrated Development Environment (IDE) for camera-based interactions, that combines standard programming with video stream capture and processing in a visual editor.

Overall, current tools for prototyping interaction reflect the situation that I already discussed for UI toolkits in general: Interface Builders are powerful tools for prototyping standard UIs by "structural combination", even with sketch-based methods, but require to switch to a lower-level paradigm for describing behaviors (standard event-based programming) and fail to support the requirements for advanced interaction. Conversely, tools for advanced interaction have better support for prototyping interactive behavior and novel interaction techniques in a creative ways, by "logical combination", but are often domain-specific.

SKETCHING INTERACTION Following the principles of "designeering interaction", I propose to explore new tools for designing advanced interaction, with the overall goal of supporting *extension*, *combination*, *reuse* and *interoperability* along the whole design process, from early design stages to functional prototyping. Sketching is the tool of choice in early stages of interaction design [Bux07b; Gre+12] but is rarely exploited further. Most sketch-based tools for interaction design, even the more recent ones [KCV10; SV12], only make it possible to draw "interfaces" (i. e. graphics and structural combinations), in a similar way to SILK [LM95]. "Sketching interaction" (i. e. creating logical combinations) raises some new challenges. It requires to abstract "interaction" as an atomic part of the system and to reify this abstraction into a visual language that matches the user's representation. The sketch-based Post-WIMP Interface Builder that I developed during my Ph.D. was a first step **[Huo+04a]**, mixing sketching for graphics and ICon's visual language for describing interaction. I started to investigate a more integrated approach, with an interactive sketch-based environment that could be appropriate for describing both the interface and the interaction. The first step is to explore the use of the state-transition and data-flow models, leveraging the properties of the FlowStates toolkit as an underlying runtime library for iterative and real-time sketch-based prototyping (see Figure 4.2).

The next steps will be to investigate how sketching can be used as a high-level "glue" for extending and combining different design tools inspired by previous work, such as programming by demonstration, and how to seamlessly integrate



(a) Sketching a state machine.

(b) Sketching graphical components.

Figure 4.2: SketchMachine!!! First prototype of a real-time sketch-based prototyping environment for advanced interaction.

standard programming when necessary (e. g. scripting or usual programming language). Finally, "sketching interaction" is likely to be user-dependent, especially in the context of interaction design, and imposing a rigid and fixed drawing language could constrain design opportunities and iterations. This raises the issue of the "resilience" of the underlying system, defined by Beaudouin-Lafon as "*the ability of a system to resist to change* [e. g. web browsers] *are very tolerant of incorrect syntax in the HTML documents they display*" [Bea04]. An interesting direction is to study personalization of the drawing language for interaction design, as done in *MusInk* [TLM09] for music composers, as a way to extend the system and its language.

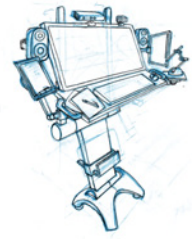
4.3.3 *Adaptability for End-Users*

At a higher level, supporting *combination* and *interoperability* as defined in the designeering framework could also benefit the end-user by making systems and applications more flexible and open to customization and adaptability. End-user programming and UI customization have been advocated for a while as a way for users to better appropriate and reinvent their interactive systems according to their needs, sometimes in unexpected ways [Mac91; MHP00; Lie+06]. This "*co-adaptation*" phenomenon [Mac91], where users not only adapt to technology but adapt the technology to their needs, is even more crucial in the emerging context of distributed and multi-devices environments [Cou06]. However, current systems still offer limited customization power [MHP00], mostly accessible to skilled users via application dependent scripting, and rarely at a system-level[3].

As mentioned previously, we will hardly revise the already established architecture of systems to account for interaction and its specificities. But we can promote intermediate tools (libraries or application frameworks) that help support customization by considering interaction as a first-class object. For example, I already discussed how ICon [DF04] and its extensions (MaggLite, FlowStates and WIS) enable a high level of adaptability and customization of interaction. Its fine level of granularity and visual language make it a powerful tool for developers and interaction designers, but they require practice and skills that are unlikely to transfer to end-users. However, the underlying reactive and data-flow models of ICon may be a good starting point for creating advanced customization tools for end-users.

3 Mac OS is an exception, with its AppleScript system-level scripting language. However, it has to be explicitly supported by applications.

I started to address these issues, in collaboration with James EAGAN during Quentin ROY's internship at in|situ|, in a project that called "Interaction Transformation". We studied several scenarios in the context of UI teleportation with the Scotty system [EBM11]: parts of a standard interface, e. g. an image viewer, running on a laptop computer were deported on a tabletop surface, and the user was able to dynamically associate touch interactions from the host device with the original scrolling controls of the application. The two main challenges we identified were: (i) to provide end-users with adapted interaction techniques to manipulate and configure interaction techniques (e. g. programing by demonstration, system suggestions); (ii) to define the underlying model that enables *combination* and *interoperability* of interaction techniques, and more precisely *reinterpretability* [Bea04]: Can we define a *level of compatibility* to ensure that two interaction techniques can replace each other for a given task? Can we define some generic *adapters* to compensate for the lack of compatibility between two interaction techniques?



↳*DigiCart blueprint (drawing by Mathieu* NANCEL*)*

These challenges will be at the heart of the recently started *Digipods* project that I am coordinating. The objective is to design new mobile and multi-device interactive environments, the *DigiCarts*, that offer customizable control of heterogeneous visualization platforms (e. g. Wall-Sized Displays, CAVEs). End-users should be able to adapt interaction methods according to their needs, to the capabilities of the platform and to the task at hand, but also to enable remote collaborative interaction with users working in other platforms(s). I expect this extreme use case of end-user customization and system adaptability to enrich the *designeering cycle* and to inform the engineering of future interactive systems – another step in climbing the Mount Improbable of HCI to extend the adjacent possible we created with the WILD Room (see Figure 4.3).

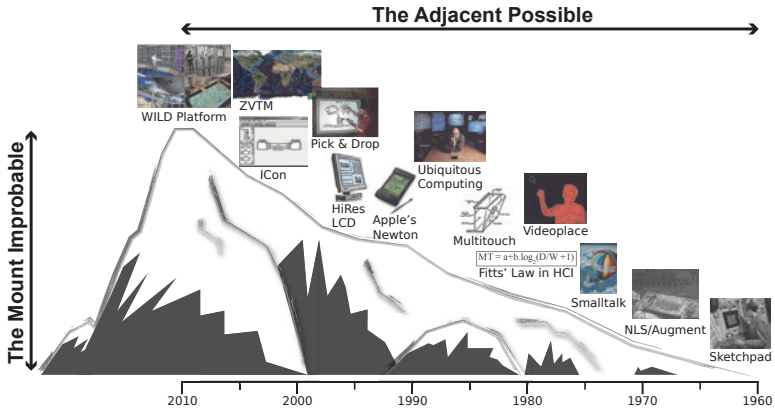


Figure 4.3: The in|situ|'s WILD Room is the result of gradual evolutions in HCI technology, made of a succession of *Adjacent Possibles* [Joh10], rather than a single jump up the steep cliff of the *Mount Improbable* [Daw97].

BIBLIOGRAPHY

[92] "A metamodel for the runtime architecture of an interactive system: the UIMS tool developers workshop." In: *SIGCHI Bull.* 24.1 (1992). Pp. 32–37. (Cit. on p. 44).

[Ada92] Douglas N. Adams. *Mostly Harmless*. Arthur Dent series. Random House, 1992. (Cit. on p. 41).

[ABo06] Anand Agarawala and Ravin Balakrishnan. "Keepin' it real: pushing the desktop metaphor with physics, piles and the pen." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '06. Montréal, Québec, Canada: ACM, 2006, pp. 1283–1292. (Cit. on p. 9).

[AGG09] Brian de Alwis, Carl Gutwin, and Saul Greenberg. "GT/SD: performance and simplicity in a groupware toolkit." In: *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*. EICS '09. Pittsburgh, PA, USA: ACM, 2009, pp. 265–274. (Cit. on p. 51).

[ACB08] Caroline Appert, Olivier Chapuis, and Michel Beaudouin-Lafon. "Evaluation of pointing performance on screen edges." In: *Proceedings of the working conference on Advanced visual interfaces*. AVI '08. Napoli, Italy: ACM, 2008, pp. 119–126. (Cit. on p. 11).

[ABo08] Caroline Appert and Michel Beaudouin-Lafon. "SwingStates: Adding State Machines to Java and the Swing Toolkit." In: *Software: Practice and Experience* 38.11 (2008). Pp. 1149–1182. (Cit. on pp. 2, 41, 43, 51, 60, 74).

[AZ09] Caroline Appert and Shumin Zhai. "Using strokes as command shortcuts: cognitive benefits and toolkit support." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 2289–2298. (Cit. on pp. 16, 41, 42, 60, 67, 70).

[AVW93] Joe Armstrong, Robert Virding, and Mike Williams. *Concurrent programming in ERLANG*. 1993. (Cit. on p. 74).

[Asa+05] Takeshi Asano, Ehud Sharlin, Yoshifumi Kitamura, Kazuki Takashima, and Fumio Kishino. "Predictive interaction using the delphian desktop." In: *Proceedings of the 18th annual ACM symposium on User interface software and technology*. UIST '05. Seattle, WA, USA: ACM, 2005, pp. 133–141. (Cit. on p. 34).

[Bai09] Gilles Bailly. "Techniques de menus : Caractérisation, Conception et Evaluation." Ph.D. Thesis. Grenoble: Université Joseph-Fourier - Grenoble I, 2009. (Cit. on p. 13).

[Bal04] Ravin Balakrishnan. ""Beating" Fitts' law: virtual enhancements for pointing facilitation." In: *Int. J. Hum.-Comput. Stud.* 61.6 (Dec. 2004). Pp. 857–874. (Cit. on p. 10).

[BB95] Jakob Bardram and Olav W. Bertelsen. "Supporting the Development of Transparent Interaction." In: *Selected papers from the 5th International Conference on Human-Computer Interaction*. EWCHI '95. London, UK, UK: Springer-Verlag, 1995, pp. 79–90. (Cit. on p. 33).

[BP99] Rémi Bastide and Philippe Palanque. "A Visual and Formal Glue between Application and Interaction." In: *Journal of Visual Languages and Computing* 10 (1999). Pp. 481–507. (Cit. on p. 43).

[Bau+03] Patrick Baudisch, Edward Cutrell, Mary Czerwinski, Daniel C. Robbins, Peter Tandler, Benjamin B. Bederson, and A. Zierlinger. "Drag-and-Pop and Drag-and-Pick: Techniques for Accessing Remote Screen Content on Touch- and Pen-Operated Systems." In: *IFIP TC13 International Conference on Human-Computer Interaction*. INTERACT '03'. IOS Press, 2003. (Cit. on p. 34).

[BR03] Patrick Baudisch and Ruth Rosenholtz. "Halo: a technique for visualizing off-screen objects." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '03. Ft. Lauderdale, Florida, USA: ACM, 2003, pp. 481–488. (Cit. on p. 11).

[Bau+04] Patrick Baudisch, Xing Xie, Chong Wang, and Wei-Ying Ma. "Collapse-to-zoom: viewing web pages on small screen devices by interactively removing irrelevant content." In: *Proceedings of the 17th annual ACM symposium on User interface software and technology*. UIST '04. Santa Fe, NM, USA: ACM, 2004, pp. 91–94. (Cit. on p. 20).

[Bea97] Michel Beaudouin-Lafon. "Interaction instrumentale : de la manipulation directe à la réalité augmentée." In: *"Actes Neuvièmes journées francophones sur l'Interaction Homme Machine*. IHM '97. Sept. 1997. (Cit. on p. 13).

[Bea99] Michel Beaudouin-Lafon, ed. *Computer Supported Co-operative Work*. Vol. 7. Trends in Software. John Wiley & Sons, 1999. (Cit. on p. 50).

[Bea00] Michel Beaudouin-Lafon. "Instrumental interaction: an interaction model for designing post-WIMP user interfaces." In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. Vol. 2(1). CHI '00. "The Hague: ACM, 2000, pp. 446–453. (Cit. on p. 43).

[BM00] Michel Beaudouin-Lafon and Wendy E. Mackay. "Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces." In: *Proceedings of the International Conference on Advanced Visual Interfaces*. AVI '08. Palermo: ACM, May 2000, pp. 102–109. (Cit. on pp. 60, 69, 71).

[BL00] Michel Beaudouin-Lafon and Michael Lassen. ""The architecture and implementation of CPN2000." In: *Proceedings of the ACM Symposium on User Interface Software and Technology*. Vol. 2(2). CHI Letters. San Diego: ACM, Nov. 2000, pp. 181–190. (Cit. on pp. 69, 74).

[Bea+01] Michel Beaudouin-Lafon, Wendy E. Mackay, Peter Andersen, Paul Janecek, Mads Jensen, Michael Lassen, Kasper Lund, Kjeld Mortensen, Stephanie Munck, Anne Ratzer, Katrine Ravn, Søren Christensen, and Kurt Jensen. "CPN/Tools: A Post-WIMP Interface for Editing and Simulating Coloured Petri Nets." In: *Proceedings of the 22nd International Conference on Application and Theory of Petri Nets (ICATPN'2001)*. Ed. by J-M Colom and M. Koutny. Lecture Notes in Computer Science. Springer-Verlag, June 2001, pp. 71–80. (Cit. on p. 69).

[Bea01] Michel Beaudouin-Lafon. "Novel interaction techniques for overlapping windows." In: *Proceedings of the 14th annual ACM symposium on User interface software and technology*. UIST '01. Orlando, Florida: ACM, 2001, pp. 153–154. (Cit. on p. 64).

[Bea04] Michel Beaudouin-Lafon. "Designing interaction, not interfaces." In: *Proceedings of the Conference on Advanced Visual Interfaces*. AVI '04. Gallipoli (Italy): ACM, May 2004, pp. 15–22. Invited keynote address. (Cit. on pp. 43, 59, 60, 63, 72, 76, 77).

[BM07] Michel Beaudouin-Lafon and Wendy E. Mackay. "Prototyping Tools and Techniques." In: *"Human Computer Interaction Handbook: Fundamentals*. Ed. by Andrew Sears and Julie A. Jacko. CRC Press, Sept. 2007. (Cit. on pp. 33, 37, 39).

[Bea08] Michel Beaudouin-Lafon. "Interaction is the Future of Computing." In: *HCI Remixed, Reflections on Works That Have Influenced the HCI Community*. Ed. by Thomas Erickson and David McDonald. MIT Press, 2008, pp. 263–266. (Cit. on p. 74).

[BH94] Benjamin B. Bederson and James D. Hollan. "Pad++: a zooming graphical interface for exploring alternate interface physics." In: *Proceedings of the 7th annual ACM symposium on User interface software and technology*. UIST '94. Marina del Rey, California, USA: ACM, 1994, pp. 17–26. (Cit. on p. 41).

[BGM04] Benjamin B. Bederson, Jesse Grosjean, and Jon Meyer. "Toolkit Design for Interactive Structured Graphics." In: *IEEE Trans. Softw. Eng.* 30.8 (Aug. 2004). Pp. 535–546. (Cit. on p. 51).

[BWB06] Hrvoje Benko, Andrew D. Wilson, and Patrick Baudisch. "Precise selection techniques for multi-touch screens." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '06. Montréal, Québec, Canada: ACM, 2006, pp. 1263–1272. (Cit. on pp. 30, 40).

[Bie+08] Jacob T. Biehl, William T. Baker, Brian P. Bailey, Desney S. Tan, Kori M. Inkpen, and Mary Czerwinski. "Impromptu: a new interaction framework for supporting collaboration in multiple display environments and its field evaluation for co-located software development." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: ACM, 2008, pp. 939–948. (Cit. on p. 51).

[Bie+93] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. "Toolglass and magic lenses: the see-through interface." In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '93. Anaheim, CA: ACM, 1993, pp. 73–80. (Cit. on pp. 28, 55).

[BGB04] 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. Vienna, Austria: ACM, 2004, pp. 519–526. (Cit. on p. 34).

[BB06] Renaud Blanch and Michel Beaudouin-Lafon. "Programming rich interactions using the hierarchical state machine toolkit." In: *Proceedings of the working conference on Advanced visual interfaces*. AVI '06. Venezia, Italy: ACM, 2006, pp. 51–58. (Cit. on pp. 41–43, 67).

[BO11] Renaud Blanch and Michael Ortega. "Benchmarking pointing techniques with distractors: adding a density factor to Fitts' pointing paradigm." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '11. Vancouver, BC, Canada: ACM, 2011, pp. 1629–1638. (Cit. on p. 10).

[BK11] Susanne Boedker and Clemens Nylandsted Klokmose. "The Human-Artifact Model: An Activity Theoretical Approach to Artifact Ecologies." In: *Human-Computer Interaction* 26.4 (Dec. 2011). Pp. 315–371. (Cit. on p. 33).

[BNG04] Jullien Bouchet, Laurence Nigay, and Thierry Ganille. "ICARE software components for rapidly developing multimodal interfaces." In: *Proceedings of the 6th international conference on Multimodal interfaces*. ACM. 2004, pp. 251–258. (Cit. on p. 41).

[Bui+02] Marcellin Buisson, Alexandre Bustico, Stéphane Chatty, Francois-Régis Colin, Yannick Jestin, Sébastien Maury, Christophe Mertz, and Philippe Truillet. "Ivy: un bus logiciel au service du développement de prototypes de systèmes interactifs." In: *Proceedings of the 14th French-speaking conference on Human-computer interaction (Conférence Francophone sur l'Interaction Homme-Machine)*. IHM '02. Poitiers, France: ACM, 2002, pp. 223–226. (Cit. on pp. 53, 72).

[Bux07a] Bill Buxton. *Multi-Touch Systems that I Have Known and Loved*. Jan. 2007. Updated by the author on Aug 30, 2012. Retrieved from: http://www.billbuxton.com/multitouchOverview.html (Accessed February 3, 2013). (Cit. on p. 2).

[Bux07b] Bill Buxton. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann, Mar. 2007. (Cit. on pp. 33, 75).

[Bux08] Bill Buxton. *The Long Nose of Innovation*. Businessweek. Jan. 2008. Retrieved from: http://www.businessweek.com/innovate/content/jan2008/id2008012_297369.htm (Accessed February 02, 2013). (Cit. on pp. 3, 33, 63).

[Bux90] William Buxton. "A three-state model of graphical input." In: *Proceedings of the IFIP TC13 Third International Conference on Human-Computer Interaction*. INTERACT '90. Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., 1990, pp. 449–456. (Cit. on p. 42).

[Cal+88] John R. Callahan, Don Hopkins, Mark D. Weiser, and Ben Shneiderman. "An empirical comparison of pie vs. linear menus." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '88. Washington, D.C., USA: ACM, 1988, pp. 95–100. (Cit. on p. 71).

[CEB78] Stuart K. Card, William K. English, and Betty J. Burr. "Evaluation of Mouse, Rate-Controlled Isometric Joystick, Step Keys, and Text Keys for Text Selection on a CRT." In: *Ergonomics* 21.8 (1978). Pp. 601–613. (Cit. on p. 4).

[CMN83] Stuart K. Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1983. (Cit. on pp. 4, 9).

[CMR91] Stuart K. Card, Jock D. Mackinlay, and George G. Robertson. "A morphological analysis of the design space of input devices." In: *ACM Trans. Inf. Syst.* 9.2 (Apr. 1991). Pp. 99–122. (Cit. on p. 4).

[CMS99] Stuart K. Card, Josh D. Mackinlay, and Ben Shneiderman. *Readings in Information Visualization: Using Vision to Think*. London: Academic Press, 1999. (Cit. on pp. 20, 23).

[CRV12] Géry Casiez, Nicolas Roussel, and Daniel Vogel. "1€ filter: a simple speed-based low-pass filter for noisy input in interactive systems." In: *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*. CHI '12. Austin, Texas, USA: ACM, 2012, pp. 2527–2530. (Cit. on p. 56).

[CG04] Matthew Chalmers and Areti Galani. "Seamful interweaving: heterogeneity in the theory and design of interactive systems." In: *Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques*. DIS '04. Cambridge, MA, USA: ACM, 2004, pp. 243–252. (Cit. on p. 33).

[Cha05] Olivier Chapuis. "Gestion des fenêtres: enregistrement et visualisation de l'interaction." In: *IHM '05: Proceedings of the 17th international conference of the Association Francophone d'Interaction Homme-Machine*. IHM 2005. Toulouse, France: ACM, 2005, pp. 255–258. (Cit. on p. 17).

[CR05] Olivier Chapuis and Nicolas Roussel. "Metisse is not a 3D desktop!" In: *Proceedings of the 18th annual ACM symposium on User interface software and technology*. UIST '05. Seattle, WA, USA: ACM, 2005, pp. 13–22. (Cit. on pp. 58, 60, 64, 74).

[CR07] Olivier Chapuis and Nicolas Roussel. "Copy-and-paste between overlapping windows." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '07. San Jose, California, USA: ACM, 2007, pp. 201–210. (Cit. on p. 64).

[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. Boston, MA, USA: ACM, 2009, pp. 1391–1400. (Cit. on pp. 10, 34, 64).

[CR10] Olivier Chapuis and Nicolas Roussel. "UIMarks: quick graphical interaction with specific targets." In: *Proceedings of the 23nd annual ACM symposium on User interface software and technology*. UIST '10. New York, New York, USA: ACM, 2010, pp. 173–182. (Cit. on p. 64).

[CLV07] S. Chatty, A. Lemort, and S. Vales. "Multiple Input Support in a Model-Based Interaction Framework." In: *Proc. TABLETOP '07*. Oct. 2007, pp. 179–186. (Cit. on pp. 41, 45).

[Cha94] Stéphane Chatty. "Extending a graphical toolkit for two-handed interaction." In: *Proceedings of the 7th annual ACM symposium on User interface software and technology*. UIST '94. Marina del Rey, California, USA: ACM, 1994, pp. 195–204. (Cit. on p. 60).

[Cha08a] Stéphane Chatty. "Programs = Data + Algorithms + Architecture: Consequences for Interactive Software Engineering." In: *Engineering Interactive Systems*. Ed. by Jan Gulliksen, MortonBorup Harning, Philippe Palanque, GerritC. Veer, and Janet Wesson. Vol. 4940. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 356–373. (Cit. on pp. 73, 74).

[Cha08b] Stéphane Chatty. "Supporting Multidisciplinary Software Composition for Interactive Applications." In: *Software Composition*. Ed. by Cesare Pautasso and Éric Tanter. Vol. 4954. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 173–189. (Cit. on pp. 73, 74).

[Cou06] Joëlle Coutaz. "Meta-user interfaces for ambient spaces." In: *Proceedings of the 5th international conference on Task models and diagrams for users interface design*. TAMODIA'06. Hasselt, Belgium: Springer-Verlag, 2006, pp. 1–15. (Cit. on pp. 51, 76).

[Cou10] Joëlle Coutaz. "User interface plasticity: model driven engineering to the limit!" In: *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering interactive Computing Systems*. ACM. 2010, pp. 1–8. (Cit. on p. 51).

[CM06] Andrew Crossan and Roderick Murray-Smith. "Rhythmic interaction for song filtering on a mobile device." In: *Haptic and Audio Interaction Design*. HAID '06. Springer, 2006, pp. 45–55. (Cit. on p. 14).

[Cru+92] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, and John C. Hart. "The CAVE: audio visual experience automatic virtual environment." In: *Commun. ACM* 35.6 (June 1992). Pp. 64–72. (Cit. on p. 98).

[Cyc] Cycling '74. *max/msp/jitter*. http://www.cycling74.com (Accessed February 10, 2013). (Cit. on p. 43).

[Das] Dassault Systèmes. *Virtools Dev*. http://www.virtools.com/ (Accessed February 10, 2013). (Cit. on p. 43).

[Daw97] Richard Dawkins. *Climbing mount improbable*. Penguin science. Penguin, 1997. Kindle edition. (Cit. on pp. 5, 77).

[Dem+08] Alexandre Demeure, Jean-Sébastien Sottet, Gaëlle Calvary, Joëlle Coutaz, Vincent Ganneau, and Jean Vanderdonckt. "The 4C Reference Model for Distributed User Interfaces." In: *Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems*. ICAS '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 61–69. (Cit. on p. 51).

[DC91] Prasun Dewan and Rajiv Choudhard. "Flexible user interface coupling in a collaborative system." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '91. New Orleans, Louisiana, USA: ACM, 1991, pp. 41–48. (Cit. on p. 51).

[DF10] Morgan Dixon and James Fogarty. "Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure." In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. CHI '10. Atlanta, Georgia, USA: ACM, 2010, pp. 1525–1534. (Cit. on p. 64).

[DF04] Pierre Dragicevic and Jean-Daniel Fekete. "Support for input adaptability in the ICON toolkit." In: *Proceedings of the 6th international conference on Multimodal interfaces*. ICMI '04. State College, PA, USA: ACM, 2004, pp. 212–219. (Cit. on pp. 41–43, 60, 65, 74, 76).

[EBM11] James R. Eagan, Michel Beaudouin-Lafon, and Wendy E. Mackay. "Cracking the cocoa nut: user interface programming at runtime." In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. UIST '11. Santa Barbara, California, USA: ACM, 2011, pp. 225–234. (Cit. on pp. 58, 64, 77).

[EK84] Pelle Ehn and Morten Kyng. "A tool perspective on design of interactive computer support for skilled workers." In: *Proceedings of the Seventh Scandinavian Research Seminar on Systemeering*. 1984. (Cit. on pp. 33, 59).

[EMP09] Stefan Eilemann, Maxim Makhinya, and Renato Pajarola. "Equalizer: A Scalable Parallel Rendering Framework." In: *IEEE Transactions on Visualization and Computer Graphics* 15.3 (2009). Pp. 436–452. (Cit. on p. 50).

[EBM05] Christoph Endres, Andreas Butz, and Asa MacWilliams. "A survey of software infrastructures and frameworks for ubiquitous computing." In: *Mob. Inf. Syst.* 1.1 (Jan. 2005). Pp. 41–80. (Cit. on p. 50).

[Eng62] Douglas C. Engelbart. *Augmenting Human Intellect: A Conceptual Framework*. Air Force Office of Scientific Research, AFOSR-3233. 1962. (Cit. on p. 1).

[Eng88] Douglas C. Engelbart. "The Augmented Knowledge Workshop." In: *History of Personal Workstations*. Ed. by Adele Goldberg. New York, NY: ACM Press, Aug. 1988, pp. 187–236. (Cit. on p. 1).

[FCR09] Guillaume Faure, Olivier Chapuis, and Nicolas Roussel. "Power tools for copying and moving: useful stuff for your desktop." In: *Proceedings of the 27th international conference on Human factors in computing systems*. CHI '09. Boston, MA, USA: ACM, Apr. 2009, pp. 1675–1678. (Cit. on pp. 14, 64).

[FMS93] Steven Feiner, Blair Macintyre, and Dorée Seligmann. "Knowledge-based augmented reality." In: *Commun. ACM* 36.7 (July 1993). Pp. 53–62. (Cit. on p. 26).

[FEG09] Jean-Daniel Fekete, Niklas Elmqvist, and Yves Guiard. "Motion-pointing: target selection using elliptical motions." In: *Proceedings of the 27th international conference on Human factors in computing systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 289–298. (Cit. on p. 14).

[Fit+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. Vancouver, Canada: ACM, 2003, pp. 71–79. (Cit. on pp. 44, 46).

[Fon+10] Amanda Fonville, Eun Kyoung Choe, Susan Oldham, and Julie A. Kientz. "Exploring the use of technology in healthcare spaces and its impact on empathic communication." In: *Proceedings of the 1st ACM International Health Informatics Symposium*. IHI '10. Arlington, Virginia, USA: ACM, 2010, pp. 497–501. (Cit. on p. 30).

[FVB06] Clifton Forlines, Daniel Vogel, and Ravin Balakrishnan. "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. Montreux, Switzerland: ACM, 2006, pp. 211–220. (Cit. on p. 45).

[Gav91] William W. Gaver. "Technology affordances." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '91. New Orleans, Louisiana, USA: ACM, 1991, pp. 79–84. (Cit. on p. 73).

[Gei+01] Christian Geiger, Bernd Kleinjohann, Christian Reimann, and Dirk Stichling. "Mobile AR4ALL." In: *4th International Symposium on Augmented Reality*. ISAR 2001. IEEE Computer Society, 2001, pp. 181–182. (Cit. on p. 26).

[Gje+11] Tony Gjerlufsen, Clemens Nylandsted Klokmose, James Eagan, Clément Pillias, and Michel Beaudouin-Lafon. "Shared substance: developing flexible multi-surface applications." In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. CHI '11. Vancouver, BC, Canada: ACM, 2011, pp. 3383–3392. (Cit. on pp. 47, 51).

[Gla01] Leon Glass. "Synchronization and rhythmic processes in physiology." In: *Nature* 410.6825 (Mar. 2001). Pp. 277–284. (Cit. on p. 14).

[Goo88] Danny Goodman. *The complete HyperCard handbook*. Macintosh performance library. Bantam Books, 1988. (Cit. on p. 74).

[GP96] Thomas R. G. Green and Marian Petre. "Usability Analysis of Visual Programming Environments: a 'cognitive dimensions' framework." In: *Journal of Visual Languages and Computing* 7 (1996). Pp. 131–174. (Cit. on p. 59).

[GF01] Saul Greenberg and Chester Fitchett. "Phidgets: easy development of physical interfaces through physical widgets." In: *Proceedings of the 14th annual ACM symposium on User interface software and technology*. UIST '01. Orlando, Florida: ACM, 2001, pp. 209–218. (Cit. on pp. 60, 75).

[Gre07] Saul Greenberg. "Toolkits and interface creativity." In: *Multimedia Tools Appl.* 32.2 (Feb. 2007). Pp. 139–159. (Cit. on pp. 39, 41).

[GB08] Saul Greenberg and Bill Buxton. "Usability evaluation considered harmful (some of the time)." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: ACM, 2008, pp. 111–120. (Cit. on pp. 61–63).

[Gre+12] Saul Greenberg, Sheelagh Carpendale, Nicolai Marquardt, and Bill Buxton. *Sketching User Experiences - The Workbook*. Academic Press, 2012. (Cit. on p. 75).

[GB05] Tovi Grossman and Ravin Balakrishnan. "The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '05. Portland, Oregon, USA: ACM, 2005, pp. 281–290. (Cit. on p. 34).

[Gui87] Yves Guiard. "Asymmetric Division of Labor in Human Skilled Bimanual Action: The Kinematic Chain as a Model." In: *J. Motor. Behav.* 19 (1987). Pp. 486–517. (Cit. on p. 31).

[Gus+08] Sean Gustafson, Patrick Baudisch, Carl Gutwin, and Pourang Irani. "Wedge: clutter-free visualization of off-screen locations." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: ACM, 2008, pp. 787–796. (Cit. on p. 11).

[HP09] Gerwin de Haan and Frits H. Post. "StateStream: a developer-centric approach towards unifying interaction models and architecture." In: *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*. EICS '09. Pittsburgh, PA, USA: ACM, 2009, pp. 13–22. (Cit. on p. 45).

[Har+06] Björn Hartmann, Scott R. Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. "Reflective physical prototyping through integrated design, test, and analysis." In: *Proceedings of the 19th annual ACM symposium on User interface software and technology*. UIST '06. Montreux, Switzerland: ACM, 2006, pp. 299–308. (Cit. on p. 75).

[Har+07] Björn Hartmann, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. "Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition." In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. CHI '07. San Jose, California, USA: ACM, 2007, pp. 145–154. (Cit. on pp. 60, 75).

[HMB13] Björn Hartmann, Wendy E. Mackay, and Michel Beaudouin-Lafon. "HydraScope: Creating Multi-Surface Meta-Applications Through View Synchronization and Input Multiplexing." In: *Proceedings of the International Symposium on Pervasive Displays*. PerDis '13. 2013, in press. (Cit. on p. 51).

[HF11] Steven J. Henderson and Steven K. Feiner. "Augmented reality in the psychomotor phase of a procedural task." In: *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*. ISMAR '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 191–200. (Cit. on p. 26).

[HCS98] Ken Hinckley, Mary Czerwinski, and Mike Sinclair. "Interaction and modeling techniques for desktop two-handed input." In: *Proceedings of the 11th annual ACM symposium on User interface software and technology*. UIST '98. San Francisco, California, USA: ACM, 1998, pp. 49–58. (Cit. on p. 42).

[Hin+05] Ken Hinckley, Patrick Baudisch, Gonzalo Ramos, and Francois Guimbretiere. "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. Portland, Oregon, USA: ACM, 2005, pp. 451–460. (Cit. on p. 14).

[HB10] Christian Holz and Patrick Baudisch. "The generalized perceived input point model and how to double touch accuracy by extracting fingerprints." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '10. Atlanta, Georgia, USA: ACM, 2010, pp. 581–590. (Cit. on pp. 20, 40).

[HB11] Christian Holz and Patrick Baudisch. "Understanding touch." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '11. Vancouver, BC, Canada: ACM, 2011, pp. 2501–2510. (Cit. on p. 40).

[HL00] Jason I. Hong and James A. Landay. "SATIN: a toolkit for informal ink-based applications." In: *Proceedings of the 13th annual ACM symposium on User interface software and technology*. UIST '00. San Diego, California, USA: ACM, 2000, pp. 63–72. (Cit. on pp. 41, 70).

[HDS11] Lode Hoste, Bruno Dumas, and Beat Signer. "Mudra: a unified multimodal interaction framework." In: *Proceedings of the 13th international conference on multimodal interfaces*. ICMI '11. Alicante, Spain: ACM, 2011, pp. 97–104. (Cit. on p. 41).

[HB99] Juan Pablo Hourcade and Benjamin B. Bederson. *Architecture and Implementation of a Java Package for Multiple Input Devices (MID)*. Tech. rep. HCIL Technical Report No. 9908, 1999. (Cit. on p. 74).

[HMS05] Scott E. Hudson, Jennifer Mankoff, and Ian Smith. "Extensible input handling in the subArctic toolkit." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '05. Portland, Oregon, USA: ACM, 2005, pp. 381–390. (Cit. on pp. 42, 60).

[Hum+02] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. "Chromium: a stream-processing framework for interactive rendering on clusters." In: *ACM Trans. Graph.* 21.3 (2002). Pp. 693–702. (Cit. on p. 50).

[Iza+03] Shahram Izadi, Harry Brignull, Tom Rodden, Yvonne Rogers, and Mia Underwood. "Dynamo: a public interactive surface supporting the cooperative sharing and exchange of media." In: *Proceedings of the 16th annual ACM symposium on User interface software and technology*. UIST '03. Vancouver, Canada: ACM, 2003, pp. 159–168. (Cit. on p. 47).

[Jac85] R. J. K. Jacob. "A State Transition Diagram Language for Visual Programming." In: *Computer* 18.8 (Aug. 1985). Pp. 51–59. (Cit. on p. 43).

[JDM99] Robert J. K. Jacob, Leonidas Deligiannidis, and Stephen Morrison. "A software model and specification language for non-WIMP user interfaces." In: *ACM Trans. Comput.-Hum. Interact.* 6.1 (Mar. 1999). Pp. 1–46. (Cit. on p. 45).

[Jeo+06] Byungil Jeong, Luc Renambot, Ratko Jagodic, Rajvikram Singh, Julieta Aguilera, Andrew Johnson, and Jason Leigh. "High-performance dynamic graphics streaming for scalable adaptive graphics environment." In: *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. Tampa, Florida: ACM, 2006. (Cit. on p. 50).

[Joh10] Steven Johnson. *Where Good Ideas Come From: The Natural History of Innovation*. Penguin Group US, 2010. (Cit. on pp. 5, 59, 61, 68, 77).

[Kal+05] Martin Kaltenbrunner, Till Bovermann, Ross Bencina, and Enrico Costanza. "Tuio: A Protokol for Table-Top Tangible User Interfaces." In: *Proceedings of Gesture Workshop 2005*. Gesture Workshop, 2005. (Cit. on pp. 38, 53, 99).

[Kar+06] Amy K. Karlson, George G. Robertson, Daniel C. Robbins, Mary P. Czerwinski, and Greg R. Smith. "FaThumb: a facet-based interface for mobile search." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '06. Montréal, Québec, Canada: ACM, 2006, pp. 711–720. (Cit. on p. 20).

[KB07] Amy K. Karlson and Benjamin B. Bederson. "ThumbSpace: generalized one-handed input for touchscreen-based mobile devices." In: *Proceedings of the 11th IFIP TC 13 international conference on Human-computer interaction*. INTERACT'07. Rio de Janeiro, Brazil: Springer-Verlag, 2007, pp. 324–338. (Cit. on pp. 20–22).

[KBC08] Amy K. Karlson, Benjamin B. Bederson, and Jose L. Contreras-Vidal. "Understanding One Handed Use of Mobile Devices." In: *Handbook of Research on User Interface Design and Evaluation for Mobile Technology*. Information Science Reference, 2008. (Cit. on pp. 3, 19).

[KMC12] Jun Kato, Sean McDirmid, and Xiang Cao. "DejaVu: integrated support for developing interactive camera-based programs." In: *Proceedings of the 25th annual ACM symposium on User interface software and technology*. UIST '12. Cambridge, Massachusetts, USA: ACM, 2012, pp. 189–196. (Cit. on pp. 60, 75).

[Kay89] Alan C. Kay. *Predicting The Future*. 1989. Retrieved from: http://www.ecotopia.com/webpress/futures.htm (Accessed March 8, 2013). (Cit. on p. 63).

[KCV10] Suzanne Kieffer, Adrien Coyette, and Jean Vanderdonckt. "User interface design by sketching: a complexity analysis of widget representations." In: *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*. EICS '10. Berlin, Germany: ACM, 2010, pp. 57–66. (Cit. on p. 75).

[Kin+12a] Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. "Proton++: a customizable declarative multitouch framework." In: *Proceedings of the 25th annual ACM symposium on User interface software and technology*. UIST '12. Cambridge, Massachusetts, USA: ACM, 2012, pp. 477–486. (Cit. on pp. 38, 41, 42, 67, 70).

[Kin+12b] Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. "Proton: multitouch gestures as regular expressions." In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. CHI '12. Austin, Texas, USA: ACM, 2012, pp. 2885–2894. (Cit. on pp. 38, 60).

[KRR10] Werner A. König, Roman Rädle, and Harald Reiterer. "Interactive Design of Multimodal User Interfaces - Reducing technical and visual complexity." In: *Journal on Multimodal User Interfaces* 3.3 (Feb. 2010). Pp. 197–213. (Cit. on pp. 41, 43).

[KB91] Gordon Kurtenbach and William Buxton. "Issues in combining marking and direct manipulation techniques." In: *Proceedings of the 4th annual ACM symposium on User interface software and technology*. UIST '91. Hilton Head, South Carolina, United States: ACM, 1991, pp. 137–144. (Cit. on pp. 13, 35, 55, 71).

[KB93] Gordon Kurtenbach and William Buxton. "The limits of expert performance using hierarchic marking menus." In: *Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*. CHI '93. Amsterdam, The Netherlands: ACM, 1993, pp. 482–487. (Cit. on p. 13).

[LM95] James A. Landay and Brad A. Myers. "Interactive sketching for the early stages of user interface design." In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. CHI '95. Denver, Colorado, United States: ACM Press/Addison-Wesley Publishing Co., 1995, pp. 43–50. (Cit. on p. 75).

[Lau+04] Laura Ann Laura Ann Petitto, Siobhan Holowka, Lauren E. Sergio, Bronna Levy, and David J. Ostry. "Baby hands that move to the rhythm of language: hearing babies acquiring sign languages babble silently on the hands." In: *Cognition* 93.1 (2004). Pp. 43–73. (Cit. on p. 14).

[Law+09] Jean-Yves Lionel Lawson, Ahmad-Amr Al-Akkad, Jean Vanderdonckt, and Benoit Macq. "An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components." In: *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*. EICS '09. Pittsburgh, PA, USA: ACM, 2009, pp. 245–254. (Cit. on pp. 60, 75).

[Leco3] Eric Lecolinet. "A molecular architecture for creating advanced GUIs." In: *Proceedings of the 16th annual ACM symposium on User interface software and technology*. UIST '03. Vancouver, Canada: ACM, 2003, pp. 135–144. (Cit. on pp. 71, 74).

[Lee08] Johnny Chung Lee. "Hacking the Nintendo Wii Remote." In: *IEEE Pervasive Computing* 7.3 (July 2008). Pp. 39–45. (Cit. on p. 5).

[LZB98] Andrea Leganchuk, Shumin Zhai, and William Buxton. "Manual and cognitive benefits of two-handed input: an experimental study." In: *ACM Trans. Comput.-Hum. Interact.* 5.4 (Dec. 1998). Pp. 326–359. (Cit. on p. 30).

[Let+10] Catherine Letondal, Stephane Chatty, Greg Phillips, Fabien André, and Stephane Conversy. "Usability requirements for interaction-oriented development tools." In: *Proceedings of the 22nd Annual Workshop of the Psychology of Programming Interest Group PPIG 2010, Sep 2010*. Ed. by Joey Lawrance and Rachel Bellamy. Maria Paloma Díaz Pérez and Mary Beth Rosson, Sept. 2010, pp. 12–26. (Cit. on p. 73).

[Lie+06] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. "End-User Development: An Emerging Paradigm." In: *End User Development*. Ed. by Henry Lieberman, Fabio Paternò, and Volker Wulf. Vol. 9. Human-Computer Interaction Series. Dordrecht: Springer Netherlands, 2006. Chap. 1, pp. 1–8. (Cit. on p. 76).

[LST08] Youn-Kyung Lim, Erik Stolterman, and Josh Tenenberg. "The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas." In: *ACM Trans. Comput.-Hum. Interact.* 15.2 (July 2008). 7:1–7:27. (Cit. on p. 33).

[Liu12] Can Liu. "Exploring Mobile Augmented Reality Instructions to Assist Operating Physical Interfaces." Master Thesis. RWTH Aachen University, Jan. 2012. (Cit. on pp. 26, 27, 62).

[MM05] Hamish G. MacDougall and Steven T. Moore. "Marching to the beat of the same drummer: the spontaneous tempo of human locomotion." In: *Journal of Applied Physiology* 99.3 (2005). Pp. 1164–1173. (Cit. on p. 14).

[Mac91] Wendy E. Mackay. "Triggers and barriers to customizing software." In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. CHI '91. New Orleans, Louisiana, United States: ACM, 1991, pp. 153–160. (Cit. on p. 76).

[MF97] Wendy E. Mackay and Anne-Laure Fayard. "HCI, natural science and design: a framework for triangulation across disciplines." In: *Proceedings of the 2nd conference on Designing interactive systems: processes, practices, methods, and techniques*. DIS '97. Amsterdam, The Netherlands: ACM, 1997, pp. 223–234. (Cit. on pp. 4, 33).

[MF99] Wendy E. Mackay and Anne Laure Fayard. "Video brainstorming and prototyping: techniques for participatory design." In: *CHI '99 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '99. Pittsburgh, Pennsylvania: ACM, 1999, pp. 118–119. (Cit. on p. 33).

[Mac+07] Wendy E. Mackay, Caroline Appert, Michel Beaudouin-Lafon, Olivier Chapuis, Yangzhou Du, Jean-Daniel Fekete, and Yves Guiard. "TouchStone: Exploratory Design of Experiments." In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. CHI '07'. ACM, Apr. 2007, pp. 1425–1434. (Cit. on pp. 39, 62).

[Mac08] Wendy E. Mackay. "From Gaia to HCI: On Multi-disciplinary Design and Co-adaptation." In: *HCI Remixed, Reflections on Works That Have Influenced the HCI Community*. Ed. by Thomas Erickson and David McDonald. MIT Press, 2008, pp. 247–251. (Cit. on p. 33).

[Mac92] I. Scott MacKenzie. "Fitts' law as a research and design tool in human-computer interaction." In: *Hum.-Comput. Interact.* 7.1 (Mar. 1992). Pp. 91–139. (Cit. on pp. 4, 34).

[MLG10] Sylvain Malacria, Eric Lecolinet, and Yves Guiard. "Clutch-free panning and integrated pan-zoom control on touch-sensitive surfaces: the cyclostar approach." In: *Proceedings of the 28th international conference on Human factors in computing systems*. CHI '10. Atlanta, Georgia, USA: ACM, 2010, pp. 2615–2624. (Cit. on p. 14).

[Mar+11] Nicolai Marquardt, Robert Diaz-Marino, Sebastian Boring, and Saul Greenberg. "The proximity toolkit: prototyping proxemic interactions in ubiquitous computing ecologies." In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. UIST '11. Santa Barbara, California, USA: ACM, 2011, pp. 315–326. (Cit. on p. 60).

[MAC99] Sébastien Maury, Sylvie Athénes, and Stéphane Chatty. "Rhythmic menus: toward interaction based on rhythm." In: *ACM SIGCHI Conference on Human Factors in Computing Systems Extended Abstracts*. CHI EA '99. Pittsburgh, Pennsylvania: ACM, 1999, pp. 254–255. (Cit. on p. 14).

[MA88] Joel McCormack and Paul Asente. "An overview of the X toolkit." In: *Proceedings of the 1st annual ACM SIGGRAPH symposium on User Interface Software*. UIST '88. Alberta, Canada: ACM, 1988, pp. 46–55. (Cit. on pp. 41, 70).

[McC+06] Michael McCurdy, Christopher Connors, Guy Pyrzak, Bob Kanefsky, and Alonso Vera. "Breaking the fidelity barrier: an examination of our current characterization of prototypes and an example of a mixed-fidelity success." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '06. Montréal, Québec, Canada: ACM, 2006, pp. 1233–1242. (Cit. on p. 33).

[Meh82] Nimish Mehta. "A Flexible Machine Interface." M.A.Sc. Thesis. Toronto: University of Toronto, Department of Electrical Engineering, 1982. (Cit. on p. 2).

[MVV11] Jérémie Melchior, Jean Vanderdonckt, and Peter Van Roy. "A model-based approach for distributed user interfaces." In: *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*. EICS '11. Pisa, Italy: ACM, 2011, pp. 11–20. (Cit. on p. 51).

[Mes] Meso Group. *vvvv : a multipurpose toolkit*. http://vvvv.org/ (Accessed February 10, 2013). (Cit. on p. 43).

[Moe02] Dirk Moelants. "Preferred tempo reconsidered." In: *Proceedings of the 7th International Conference on Music Perception and Cognition*. Ed. by C Stevens, D Burnham, G McPherson, E Schubert, and J. A. Renwick. Sydney: AMPS, 2002, pp. 580–583. (Cit. on p. 14).

[MH08] Tomer Moscovich and John F. Hughes. "Indirect mappings of multi-touch input using one and two hands." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: ACM, 2008, pp. 1275–1284. (Cit. on p. 30).

[MHP00] Brad Myers, Scott E. Hudson, and Randy Pausch. "Past, present, and future of user interface software tools." In: *ACM Trans. Comput.-Hum. Interact.* 7.1 (Mar. 2000). Pp. 3–28. (Cit. on pp. 33, 62, 68, 70, 72, 74, 76).

[Mye+90] Brad A. Myers, Dario A. Giuse, Roger B. Dannenberg, David S. Kosbie, Edward Pervin, Andrew Mickish, Brad Vander Zanden, and Philippe Marchal. "Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces." In: *Computer* 23.11 (Nov. 1990). Pp. 71–85. (Cit. on pp. 42, 60).

[Mye91] Brad A. Myers. "Separating application code from toolkits: eliminating the spaghetti of call-backs." In: *Proceedings of the 4th annual ACM symposium on User interface software and technology*. UIST '91. Hilton Head, South Carolina, USA: ACM, 1991, pp. 211–220. (Cit. on pp. 2, 42, 60).

[Mye+97] Brad A. Myers, Richard G. McDaniel, Robert C. Miller, Alan S. Ferrency, Andrew Faulring, Bruce D. Kyle, Andrew Mickish, Alex Klimovitski, and Patrick Doane. "The Amulet Environment: New Models for Effective User Interface Software Development." In: *IEEE Trans. Softw. Eng.* 23.6 (June 1997). Pp. 347–365. (Cit. on pp. 42, 60).

[Mye98] Brad A. Myers. "A Brief History of Human Computer Interaction Technology." In: *ACM Interactions* 5.2 (Mar. 1998). Pp. 44–54. (Cit. on p. 2).

[Nac+06] Miguel A. Nacenta, Samer Sallam, Bernard Champoux, Sriram Subramanian, and Carl Gutwin. "Perspective cursor: perspective-based interaction for multi-display environments." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '06. Montréal, Québec, Canada: ACM, 2006, pp. 289–298. (Cit. on p. 47).

[Nac+07] Miguel A. Nacenta, Satoshi Sakurai, Tokuo Yamaguchi, Yohei Miki, Yuichi Itoh, Yoshifumi Kitamura, Sriram Subramanian, and Carl Gutwin. "E-conic: a perspective-aware interface for multi-display environments." In: *Proceedings of the 20th annual ACM symposium on User interface software and technology*. UIST '07. Newport, Rhode Island, USA: ACM, 2007, pp. 279–288. (Cit. on p. 47).

[Nam+09] Sungwon Nam, Byungil Jeong, Luc Renambot, Andrew Johnson, Kelly Gaither, and Jason Leigh. "Remote visualization of large scale data for ultra-high resolution display environments." In: *Proceedings of the 2009 Workshop on Ultrascale Visualization*. UltraVis '09. Portland, Oregon: ACM, 2009, pp. 42–44. (Cit. on p. 53).

[Nan12] Mathieu Nancel. "Designing and Combining Interaction Techniques in Large Display Environments." Ph.D. Thesis. Université Paris-Sud, Dec. 2012. (Cit. on pp. 47, 57).

[Nav+06] David Navarre, Philippe Palanque, Pierre Dragicevic, and Rémi Bastide. "An approach integrating two complementary model-based environments for the construction of multimodal interactive applications." In: *Interact. Comput.* 18.5 (Sept. 2006). Pp. 910–941. (Cit. on p. 45).

[New+03] Mark W. Newman, James Lin, Jason I. Hong, and James A. Landay. "DENIM: an informal web site design tool inspired by observations of practice." In: *Hum.-Comput. Interact.* 18.3 (Sept. 2003). Pp. 259–324. (Cit. on p. 75).

[Ni+06] Tao Ni, Greg S. Schmidt, Oliver G. Staadt, Mark A. Livingston, Robert Ball, and Richard May. "A Survey of Large High-Resolution Display Technologies, Techniques, and Applications." In: *Proceedings of the IEEE conference on Virtual Reality*. VR '06. IEEE, 2006, pp. 223–236. (Cit. on p. 50).

[NR12] Donald A. Norman and Verganti Roberto. "Incremental and Radical Innovation: Design Research versus Technology and Meaning Change." Mar. 2012. Based on a talk presented at the Designing Pleasurable Products and Interfaces conference in Milan, 2011. Retrieved from: http://www.jnd.org/dn.mss/incremental_and_radi.html (Accessed February 1, 2013). (Cit. on p. 5).

[OK05] Dan R. Olsen Jr. and Scott R. Klemmer. "The future of user interface design tools." In: *ACM SIGCHI Conference on Human Factors in Computing Systems Extended Abstracts*. CHI EA '05. Portland, OR, USA: ACM, 2005, pp. 2134–2135. (Cit. on p. 67).

[Ols07] Dan R. Olsen Jr. "Evaluating user interface systems research." In: *Proceedings of the 20th annual ACM symposium on User interface software and technology*. UIST '07. Newport, Rhode Island, USA: ACM, 2007, pp. 251–258. (Cit. on pp. 34, 39, 40, 42, 57, 59, 61, 63, 64, 72).

[PRM00] Jason Pascoe, Nick Ryan, and David Morse. "Using while moving: HCI issues in fieldwork environments." In: *ACM Trans. Comput.-Hum. Interact.* 7.3 (Sept. 2000). Pp. 417–437. (Cit. on p. 20).

[PF93] Ken Perlin and David Fox. "Pad: an alternative approach to the computer interface." In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '93. Anaheim, CA: ACM, 1993, pp. 57–64. (Cit. on p. 23).

[PH08] Keith B. Perry and Juan Pablo Hourcade. "Evaluating one handed thumb tapping on mobile touchscreen devices." In: *Proceedings of graphics interface 2008*. GI '08. Windsor, Ontario, Canada: Canadian Information Processing Society, 2008, pp. 57–64. (Cit. on p. 20).

[Pie05] Emmanuel Pietriga. "A Toolkit for Addressing HCI Issues in Visual Language Environments." In: *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*. VLHCC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 145–152. (Cit. on pp. 34, 41, 42, 51, 52, 67).

[PWS88] R. L. Potter, L. J. Weldon, and B. Shneiderman. "Improving the accuracy of touch screens: an experimental evaluation of three strategies." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '88. Washington, D.C., USA: ACM, 1988, pp. 27–32. (Cit. on pp. 20–22, 40).

[QCD12] Philip Quinn, Andy Cockburn, and Jérôme Delamarche. "Examining the costs of multiple trajectory pointing techniques." In: *International Journal of Human-Computer Studies* (2012). to appear. (Cit. on p. 12).

[RBB04] Gonzalo Ramos, Matthew Boulos, and Ravin Balakrishnan. "Pressure widgets." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '04. Vienna, Austria: ACM, 2004, pp. 487–494. (Cit. on p. 35).

[RB07] Gonzalo A. Ramos and Ravin Balakrishnan. "Pressure marks." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '07. San Jose, California, USA: ACM, 2007, pp. 1375–1384. (Cit. on p. 35).

[RC94] Ramana Rao and Stuart K. Card. "The table lens: merging graphical and symbolic representations in an interactive focus + context visualization for tabular information." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '94. Boston, Massachusetts, USA: ACM, 1994, pp. 318–322. (Cit. on p. 20).

[Rek97] Jun Rekimoto. "Pick-and-drop: a direct manipulation technique for multiple computer environments." In: *Proceedings of the 10th annual ACM symposium on User interface software and technology*. UIST '97. Banff, Alberta, Canada: ACM, 1997, pp. 31–39. (Cit. on p. 47).

[RS99] Jun Rekimoto and Masanori Saitoh. "Augmented surfaces: a spatially continuous work space for hybrid computing environments." In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. CHI '99. Pittsburgh, Pennsylvania, USA: ACM, 1999, pp. 378–385. (Cit. on p. 47).

[RG96] Mark Roseman and Saul Greenberg. "Building real-time groupware with GroupKit, a groupware toolkit." In: *ACM Trans. Comput.-Hum. Interact.* 3.1 (Mar. 1996). Pp. 66–106. (Cit. on p. 51).

[RSI96] Jim Rudd, Ken Stern, and Scott Isensee. "Low vs. high-fidelity prototyping debate." In: *interactions* 3.1 (Jan. 1996). Pp. 76–85. (Cit. on p. 33).

[Sac08] Oliver Sacks. *Musicophilia: Tales of Music and the Brain*. 2nd ed. Vol. 1. New York, USA: Vintage Books, 2008. (Cit. on p. 14).

[SV12] Ugo Sangiorgi and Jean Vanderdonckt. "GAMBIT: Addressing multi-platform collaborative sketching with html5." In: *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*. EICS '12. Copenhagen, Denmark: ACM, 2012, pp. 257–262. (Cit. on p. 75).

[SPM04] Carsten Schwesig, Ivan Poupyrev, and Eijiro Mori. "Gummi: a bendable computer." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '04. Vienna, Austria: ACM, 2004, pp. 263–270. (Cit. on p. 30).

[Sco+10] James Scott, Shahram Izadi, Leila Sadat Rezai, Dominika Ruszkowski, Xiaojun Bi, and Ravin Balakrishnan. "RearType: text entry using keys on the back of a device." In: *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*. MobileHCI '10. Lisbon, Portugal: ACM, 2010, pp. 171–180. (Cit. on p. 30).

[SM89] McKay Moore Sohlberg and Catherine A. Mateer. *Introduction to Cognitive Rehabilitation: Theory and Practice*. Guilford Press, 1989. (Cit. on p. 25).

[Stu+06] Wolfgang Stuerzlinger, Olivier Chapuis, Dusty Phillips, and Nicolas Roussel. "User interface façades: towards fully adaptable user interfaces." In: *Proceedings of the 19th annual ACM symposium on User interface software and technology*. UIST '06. Montreux, Switzerland: ACM, 2006, pp. 309–318. (Cit. on p. 64).

[Sut63] Ivan Edward Sutherland. "Sketchpad: A man-machine graphical communication system." Ph.D. Dissertation. Cambridge, Massachusetts: Massachusetts Institute of Technology, Electrical Engineering Department, Jan. 1963. (Cit. on p. 1).

[SL07] Christine Szentgyorgyi and Edward Lank. "Five-key text input using rhythmic mappings." In: *Proceedings of the 9th international conference on Multimodal interfaces*. ICMI '07. Nagoya, Aichi, Japan: ACM, 2007, pp. 118–121. (Cit. on p. 14).

[Tan+03] Arthur Tang, Charles Owen, Frank Biocca, and Weimin Mou. "Comparative effectiveness of augmented reality in object assembly." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '03. Ft. Lauderdale, Florida, USA: ACM, 2003, pp. 73–80. (Cit. on p. 26).

[Tay+01] Russell M. Taylor II, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, and Aron T. Helser. "VRPN: a device-independent, network-transparent VR peripheral system." In: *Proceedings of the ACM symposium on Virtual reality software and technology*. VRST '01. Baniff, Alberta, Canada: ACM, 2001, pp. 55–61. (Cit. on pp. 53, 99).

[TLM09] Theophanis Tsandilas, Catherine Letondal, and Wendy E. Mackay. "Musink: composing music through augmented drawing." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 819–828. (Cit. on p. 76).

[TG04] Edward Tse and Saul Greenberg. "Rapidly prototyping Single Display Groupware through the SDGToolkit." In: *Proceedings of the fifth conference on Australasian user interface - Volume 28*. AUIC '04. Dunedin, New Zealand: Australian Computer Society, Inc., 2004, pp. 101–110. (Cit. on pp. 41, 51, 74).

[TP92] Sherry Turkle and Seymour Papert. "Epistemological pluralism and the revaluation of the concrete." In: *Journal of Mathematical Behavior* 11.1 (1992). Pp. 3–33. (Cit. on p. 33).

[VCT12] Radu-Daniel Vatavu, Catalin-Marian Chera, and Wei-Tek Tsai. "Gesture Profile for Web Services: An Event-Driven Architecture to Support Gestural Interfaces for Smart Environments." In: *Ambient Intelligence*. Ed. by Fabio Paternò, Boris Ruyter, Panos Markopoulos, Carmen Santoro, Evert Loenen, and Kris Luyten. Vol. 7683. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 161–176. (Cit. on p. 72).

[VB07] Daniel Vogel and Patrick Baudisch. "Shift: a technique for operating pen-based interfaces using touch." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '07. San Jose, California, USA: ACM, 2007, pp. 657–666. (Cit. on pp. 20–22).

[WS03] Daniel Wagner and Dieter Schmalstieg. "First Steps Towards Handheld Augmented Reality." In: *Proceedings of the 7th IEEE International Symposium on Wearable Computers*. ISWC '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 127–. (Cit. on p. 26).

[Wag12] Julie Wagner. "A Body-centric Framework for Generating and Evaluating Novel Interaction Techniques." Ph.D. Thesis. Université Paris-Sud, Dec. 2012. (Cit. on pp. 30, 32, 47, 57).

[Wei91] Mark Weiser. "The Computer for the 21st Century." In: *Scientific American* 265.3 (Jan. 1991). Pp. 66–75. (Cit. on pp. 2, 46).

[Wig+07] Daniel Wigdor, Clifton Forlines, Patrick Baudisch, John Barnwell, and Chia Shen. "Lucid touch: a see-through mobile device." In: *Proceedings of the 20th annual ACM symposium on User interface software and technology*. UIST '07. Newport, Rhode Island, USA: ACM, 2007, pp. 269–278. (Cit. on p. 30).

[Wil+11] Max L. Wilson, Wendy E. Mackay, Ed Chi, Michael Bernstein, Dan Russell, and Harold Thimbleby. "RepliCHI - CHI should be replicating and validating results more: discuss." In: *CHI '11 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '11. Vancouver, BC, Canada: ACM, 2011, pp. 463–466. (Cit. on p. 61).

[Wob09] Jacob O. Wobbrock. "TapSongs: tapping rhythm-based passwords on a single binary sensor." In: *Proceedings of the 22nd annual ACM symposium on User interface software and technology*. UIST '09. Victoria, BC, Canada: ACM, 2009, pp. 93–96. (Cit. on p. 14).

[Wob+09] Jacob O. Wobbrock, James Fogarty, Shih-Yen (Sean) Liu, Shunichi Kimuro, and Susumu Harada. "The angle mouse: target-agnostic dynamic gain adjustment based on angular deviation." In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 1401–1410. (Cit. on pp. 10, 34).

[WF97] Matthew Wright and Adrian Freed. "Open Sound Control: A New Protocol for Communicating with Sound Synthesizers." In: *International Computer Music Conference*. Thessaloniki, Hellas: International Computer Music Association, 1997, pp. 101–104. (Cit. on pp. 53, 98).

[WB03] Mike Wu and Ravin Balakrishnan. "Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays." In: *Proceedings of the 16th annual ACM symposium on User interface software and technology*. UIST '03. Vancouver, Canada: ACM, 2003, pp. 193–202. (Cit. on p. 30).

[Yat+08] Koji Yatani, Kurt Partridge, Marshall Bern, and Mark W. Newman. "Escape: a target selection technique using visually-cued gestures." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: ACM, 2008, pp. 285–294. (Cit. on p. 21).

[ZDB12] Brian Ziebart, Anind Dey, and J. Andrew Bagnell. "Probabilistic pointing target prediction via inverse optimal control." In: *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*. IUI '12. Lisbon, Portugal: ACM, 2012, pp. 1–10. (Cit. on p. 10).

LIST OF FIGURES

ACRONYMS

ACM	Association for Computing Machinery (see http://www.acm.org/)
ACM SIGCHI	ACM Special Interest Group on Computer-Human Interaction (see http://www.sigchi.org/)
API	Application Programming Interface
AR	Augmented Reality
CAVE	Cave Automatic Virtual Environment (see [Cru+92])
CHI	ACM SIGCHI Conference on Human Factors in Computing Systems
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CSCW	Computer-Supported Cooperative Work
DOI	Degree Of Interest
DOM	Document Object Model
DIY	Do It Yourself
F+C	Focus + Context
FTIR	Frustrated Total Internal Reflection
GUI	Graphical User Interface
GPU	Graphics Processing Unit
HCI	Human Computer Interaction
HMD	Head-Mounted Display
HTML	HyperText Markup Language
IDE	Integrated Development Environment
JVM	Java Virtual Machine
LED	Light-Emitting Diode
MIDI	Musical Instrument Digital Interface
MVC	Model-View-Controller
OOP	Object Oriented Programming
OS	Operating System
OSC	Open Sound Control (see [WF97])
PbD	Programming by Demonstration
PDA	Personal Digital Assistant
RFID	Radio-Frequency Identification

RTF	Rich Text Format
RPC	Remote Procedure Call
SOA	Service Oriented Architecture
TUIO	Tangible User Interface Objects (see [Kal+05])
UI	User Interface
UIST	Annual ACM Symposium on User Interface Software and Technology
VR	Virtual Reality
VRPN	Virtual Reality Peripheral Network (see [Tay+01])
WILD	Wall-Sized Interaction with Large Datasets (in | situ | high-resolution visualization platform project, see http://www.lri.fr/~mbl/WILD/)
WIS	WILDInputServer (the input and interaction management module of jBricks, see section 3.2.3)
WIMP	Windows Icons Menus Pointer, the standard computer GUI (term attributed to Merzouga Wilberts in 1980)

Appendices

PUBLICATIONS

A.1 FULL LIST OF PUBLICATIONS

In the field of Human-Computer Interaction, conferences are considered the primary method of publication. Some of these conferences (ACM CHI, ACM UIST) are particularly selective and are considered journal level work. In the following list, the names of major conferences are in bold. Distinguished publications (best paper awards or nominations) are in gray boxes.

REFEREED INTERNATIONAL JOURNALS AND MAGAZINES

[Bea+12] Michel Beaudouin-Lafon, **Stéphane Huot**, Mathieu Nancel, Wendy Mackay, Emmanuel Pietriga, Romain Primet, Julie Wagner, Olivier Chapuis, Clément Pillias, James R. Eagan, Tony Gjerlufsen, and Clemens Klokmose. "Multi-surface Interaction in the WILD Room." In: *IEEE Computer* 45.4 (2012). Pp. 48–56. http://hal.inria.fr/docs/00/68/78/25/PDF/WILD-IEEEComputer-authorversion.pdf. (Cit. on pp. 47, 48, 58).

REFEREED INTERNATIONAL CONFERENCES

[Wag+13] Julie Wagner, Mathieu Nancel, Sean Gustafson, **Stéphane Huot**, and Wendy E. Mackay. "A Body-centric Design Space for Multi-surface Interaction." In: *Proceedings of the **31st international conference on Human factors in computing systems***. **CHI '13**. Paris, France: ACM, 2013, 10 pages, to appear. *Honorable Mention*. http://hal.inria.fr/hal-00789169/PDF/BodyScape-Hal.pdf. (Cit. on pp. 4, 109).

[Gho+12] Emilien Ghomi, Guillaume Faure, **Stéphane Huot**, Olivier Chapuis, and Michel Beaudouin-Lafon. "Using Rhythmic Patterns as an Input Method." In: *Proceedings of the **30th international conference on Human factors in computing systems***. **CHI '12**. Austin, Texas, USA: ACM, 2012, pp. 1253–1262. *Best Paper*. http://hal.archives-ouvertes.fr/docs/00/66/39/73/PDF/CHI12-ewe-halv1.pdf. (Cit. on pp. 14, 16, 109).

[Liu+12] Can Liu, **Stéphane Huot**, Jonathan Diehl, Wendy E. Mackay, and Michel Beaudouin-Lafon. "Evaluating the Benefits of Real-time Feedback in Mobile Augmented Reality with Hand-held Devices." In: *Proceedings of the **30th international conference on Human factors in computing systems***. **CHI '12**. Austin, Texas, USA: ACM, 2012, pp. 2973–2976. *Honorable Mention*. http://hal.inria.fr/hal-00663974/PDF/ARFeedbackA.pdf. (Cit. on p. 29).

[WHM12] Julie Wagner, **Stéphane Huot**, and Wendy E. Mackay. "BiTouch and BiPad: Designing Bimanual Interaction for Hand-held Tablets." In: *Proceedings of the **30th international conference on Human factors in computing systems***. **CHI '12**. Austin, Texas, USA: ACM, 2012, pp. 2317–2326. http://hal.inria.fr/hal-00663972/PDF/bipadA.pdf. (Cit. on pp. xiii, 30, 32, 109).

[DHC11] Pierre Dragicevic, **Stéphane Huot**, and Fanny Chevalier. "Glimpse: Animating from Markup Code to Rendered Documents and Vice-Versa." In: *Proceedings of the **24th ACM Symposium on User Interface Software and Technology***. **UIST 2011**. Santa-Barbara, CA, USA: ACM, 2011, pp. 257–262. http://hal.inria.fr/docs/00/62/62/59/PDF/GlimpseA.pdf. (Cit. on pp. 18, 37, 109).

[HCD11] **Stéphane Huot**, Olivier Chapuis, and Pierre Dragicevic. "TorusDesktop: Pointing via the Backdoor is Sometimes Shorter." In: *Proceedings of the **29th international conference on Human factors in computing systems***. **CHI '11**. Vancouver, CA: ACM, 2011, pp. 829–838. http://hal.archives-ouvertes.fr/docs/00/59/12/95/PDF/CHI11-torus-avf.pdf. (Cit. on pp. 12, 35, 109).

[Pie+11] Emmanuel Pietriga, **Stéphane Huot**, Mathieu Nancel, and Romain Primet. "Rapid Development of User Interfaces on Cluster-Driven Wall Displays with jBricks." In: *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*. EICS '11. Pisa, Italy: ACM, 2011, pp. 185–190. http://hal.inria.fr/docs/00/58/54/79/PDF/jbricks-eics11.pdf. (Cit. on pp. 47, 51, 52, 57, 109).

[CHF10a] Fanny Chevalier, **Stéphane Huot**, and Jean-Daniel Fekete. "Visualisation de mesures agrégées pour l'estimation de la qualité des articles Wikipedia." In: *Proceedings of EGC 2010: Conférence Internationale Francophone sur l'Extraction et la Gestion des Connaissances, Revue des Nouvelles Technologies de l'Information RNTI-E-19*. EGC 2010. Hammamet, Tunisia: Cépaduès-Éditions, Jan. 2010, pp. 351–362. http://hal.inria.fr/docs/00/55/06/97/PDF/WikipediaVIZ-EGC2010.pdf.

[CHF10b] Fanny Chevalier, **Stéphane Huot**, and Jean-Daniel Fekete. "WikipediaViz: Conveying Article Quality for Casual Wikipedia Readers." In: *Proceedings of IEEE Pacific Visualization Symposium*. PacificVis '10. Taipei, Taïwan: IEEE, Mar. 2010, pp. 215–222. http://hal.inria.fr/docs/00/55/06/98/PDF/WikipediaViz-PacificVis2010.pdf.

[RHL08] Anne Roudaut, **Stéphane Huot**, and Eric Lecolinet. "TapTap and MagStick: Improving One-Handed Target Acquisition on Small Touch-screens." In: *Proceedings of **the 9th International Working Conference on Advanced Visual Interfaces***. **AVI '08**. ACM, May 2008, pp. 146–153. http://hal.inria.fr/docs/00/55/06/94/PDF/TapTap-AVI08.pdf. (Cit. on pp. 21, 22, 40, 108).

[HL07b] **Stéphane Huot** and Eric Lecolinet. "Focus+Context Visualization Techniques for Displaying Large Lists with Multiple Points of Interest on Small Tactile Screens." In: *Proceedings of **11th IFIP TC13 International Conference on Human-Computer Interaction***. **Interact 2007**. Springer Verlag, Lecture Notes in Computer Science, Sept. 2007, pp. 219–233. http://hal.inria.fr/docs/00/55/05/98/PDF/SnaiList-Interact2007.pdf. (Cit. on pp. 24, 25, 108).

[HL06] **Stéphane Huot** and Eric Lecolinet. "SpiraList: A Compact Visualization Technique for One-Handed Interaction with Large Lists on Mobile Devices." In: *Proceedings of the 4th Nordic Conference on Human-Computer Interaction, NordiCHI 2006*. NordiCHI 2006. Nordic HCI organizations. Oslo, Norway: ACM, Oct. 2006, pp. 445–448. http://hal.inria.fr/docs/00/55/06/00/PDF/SpiraList-NordiCHI2006.pdf. (Cit. on p. 24).

[Huo+04a] **Stéphane Huot**, Cédric Dumas, Pierre Dragicevic, Jean-Daniel Fekete, and Gérard Hégron. "The MaggLite Post-WIMP Toolkit: Draw It, Connect It and Run It." In: *Proceedings of **ACM Symposium on User Interface Software and Technology*, **UIST 2004***. Vol. 5(2). CHI Letters. Santa Fe, NM: ACM, Oct. 2004, pp. 257–266. http://hal.inria.fr/docs/00/55/05/95/PDF/MaggLite-UIST2004.pdf. (Cit. on pp. 41, 43, 51, 55, 57, 60, 75).

[HDH03] **Stéphane Huot**, Cédric Dumas, and Gérard Hégron. "Toward Creative 3D Modeling: an Architects' Sketches Study." In: *Proceedings of **9th IFIP TC13 International Conference on Human-Computer Interaction***. **Interact '03**. Zurich, Switzerland: IOS Press, Sept. 2003, pp. 785–788. http://hal.inria.fr/docs/00/55/05/91/PDF/Architects-Sketches-Interact2003.pdf.

[Bou+02] Didier Boucard, **Stéphane Huot**, Christian Colin, Daniel Siret, and Gérard Hégron. "An Image-Based and Knowledge-Based system for efficient architectural and urban modeling." In: *Proceedings of the international conference ACADIA*. Los Angeles, CA, USA, Oct. 2002, pp. 231–240.

[DH02] Pierre Dragicevic and **Stéphane Huot**. "SpiraClock: a continuous and non-intrusive display for upcoming events." In: ***extended abstracts on Human factors in computer systems***. **CHI '02**. Minneapolis, MN, USA: ACM Press, 2002, pp. 604–605. http://hal.inria.fr/docs/00/55/05/99/PDF/SpiraClock-CHI2002.pdf.

[Sos+02] Alexey Sosnov, **Stéphane Huot**, Pierre Macé, and Gérard Hégron. "Rapid Incremental Architectural Modeling from Imprecise Perspective Sketches and Geometric Constraints." In: *Proceedings of the international conference Graphicon*. Linz, Autriche, 2002.

Refereed Domestic Conferences

[WMH12] Julie Wagner, Wendy E. Mackay, and **Stéphane Huot**. "Left-over Windows Cause Window Clutter... But What Causes Left-over Windows?" In: *Ergo IHM 2012: Proceedings of the 24th French Speaking Conference on Human-Computer Interaction*. Biarritz, France: ACM, International Conference Proceedings Series, Oct. 2012, pp. 47–50. http://hal.inria.fr/docs/00/77/63/01/PDF/WMLisa-hal.pdf. (Cit. on p. 17).

[App+09] Caroline Appert, **Stéphane Huot**, Pierre Dragicevic, and Michel Beaudouin-Lafon. "FlowStates: Prototypage d'applications interactives avec des flots de données et des machines à états." In: *Proceedings of IHM 2009, 21ème conférence francophone sur l'Interaction Homme-Machine*. ACM, International Conference Proceedings Series, Oct. 2009, pp. 119–128. *Best Paper*. http://hal.inria.fr/docs/00/53/85/98/PDF/FlowStates.pdf. (Cit. on pp. 43, 44, 109).

[NBH09] Mathieu Nancel, Michel Beaudouin-Lafon, and **Stéphane Huot**. "Un espace de conception fondé sur une analyse morphologique des techniques de menus." In: *Proceedings of IHM 2009, 21ème conférence francophone sur l'Interaction Homme-Machine*. ACM, International Conference Proceedings Series, Oct. 2009, pp. 13–22. http://hal.inria.fr/docs/00/55/05/94/PDF/Design_space_menus-IHM2009.pdf. (Cit. on pp. 4, 13).

[HL07a] **Stéphane Huot** and Eric Lecolinet. "ArchMenu et ThumbMenu : Contrôler son dispositif mobile "sur le pouce"." In: *Proceedings of IHM 2007, 19ème conférence francophone sur l'Interaction Homme-Machine*. ACM, International Conference Proceedings Series, Nov. 2007, pp. 107–110. http://hal.inria.fr/docs/00/55/05/93/PDF/ArchThumbMenu-IHM2007.pdf.

[HDD06] **Stéphane Huot**, Pierre Dragicevic, and Cédric Dumas. "Flexibilité et modularité pour la conception d'interactions: Le modèle d'architecture logicielle des Graphes Combinés." In: *Actes de la 18ème conférence francophone sur l'Interaction Homme-Machine, IHM'06*. AFIHM. Montreal, Canada: ACM, Apr. 2006, pp. 43–50. http://hal.inria.fr/docs/00/55/05/96/PDF/MixedGraphs-IHM2006.pdf. (Cit. on p. 41).

[Huo+04b] **Stéphane Huot**, Cédric Dumas, Pierre Dragicevic, and Gérard Hégron. "Conception et utilisation d'interactions avancées avec la boîte à outils MaggLite." In: *Actes de la 16ème conférence francophone sur l'Interaction Homme-Machine, IHM 2004*. AFIHM. Namur, Belgique: ACM Press, Aug. 2004, pp. 177–178. Demonstration.

[HDH04] **Stéphane Huot**, Cédric Dumas, and Gérard Hégron. "Svalabard: Une table à dessin virtuelle pour la modélisation 3D." In: *Actes de la 16ème conférence francophone sur l'Interaction Homme-Machine, IHM 2004*. AFIHM. Namur, Belgique: ACM Press, Aug. 2004, pp. 85–92. http://hal.inria.fr/docs/00/55/06/91/PDF/Svalabard-IHM2004.pdf.

[HD02] **Stéphane Huot** and Cédric Dumas. "Vers des modeleurs 3D créatifs: Étude de dessins d'architectes." In: *Actes de la 14ème conférence Francophone sur l'Interaction Homme-Machine, IHM 2002*. AFIHM. Poitiers, France: ACM Press, Nov. 2002, pp. 267–270.

WORKSHOPS

[Liu+11] Can Liu, Jonathan Diehl, **Stéphane Huot**, and Jan Borchers. "Mobile Augmented Note-taking to Support Operating Physical Devices." In: *Mobile HCI 2011 – Workshop on Mobile Augmented Reality: Design Issues and Opportunities*. Aug. 2011. http://hal.inria.fr/inria-00626260/PDF/Final_submission.pdf. (Cit. on p. 26).

[Gho+10] Emilien Ghomi, Olivier Bau, Wendy Mackay, and **Stéphane Huot**. "Conception et apprentissage des interactions tactiles: le cas des postures multi-doigts." In: *FITG '10: French workshop on tactile and gestural interaction*. June 2010. http://fitg10.lille.inria.fr/workshop-data/proposals/ghomi-et-al.pdf.

THESIS

[Huo05] **Stéphane Huot**. "Une nouvelle approche pour la conception créative: De l'interprétation du dessin à main levée au prototypage d'interactions non-standard." Ph.D. Dissertation (thèse de doctorat). Université de Nantes, École Nationale Supérieure des Techniques Industrielles et des Mines de Nantes, July 2005. http://tel.archives-ouvertes.fr/docs/00/04/84/43/PDF/tel-00010210.pdf. (Cit. on pp. 41, 67).

[Huo00] **Stéphane Huot**. "Reconstruction de bâtiments 3D à partir d'images." Master Dissertation (mémoire de DEA). Université de Nantes, École Nationale Supérieure des Techniques Industrielles et des Mines de Nantes, Sept. 2000.

OTHER PUBLICATIONS

[Raf+12] Bruno Raffin, Hannah Carbonnier, Jérôme Esnault, Jean-Christophe Lombardo, Rémi Felix, Thierry Duval, Alain Chauffaut, Georges Dumont, Ronan Gaugne, Valérie Gouranton, François Faure, Jérémie Allard, Romain Primet, **Stéphane Huot**, Yvonne Jung, Ulrich Bockholt, Johannes Behr, Karsten Schwenk, and Gerrit Voss. "The VCoRE Project: First Steps Towards Building a Next-Generation Visual Computing Platform." In: $7^{èmes}$ *Journées de l'Association Française de Réalité Virtuelle (French Association for Virtual Reality)*. AFRV '12. 2012.

[Bea+11] Michel Beaudoin-Lafon, Emmanuel Pietriga, Wendy Mackay, **Stéphane Huot**, Clément Pillias, and Romain Primet. "131 millions de pixel qui font le mur." In: *Plein Sud Spécial Recherche 2010-2011*. 2011, pp. 124–131.

[HNB08] **Stéphane Huot**, Mathieu Nancel, and Michel Beaudouin-Lafon. *PushMenu: Extending Marking Menus for Pressure-Enabled Input Devices*. Research Report 1502. LRI, Université Paris-Sud, France, 2008. http://hal.inria.fr/docs/00/55/05/97/PDF/PushMenu-RR2008.pdf. (Cit. on p. 13).

[HC02] **Stéphane Huot** and Christian Colin. "MArINa : reconstruction de bâtiments 3D à partir d'images." In: *Colloque Modélisation Multimodale appliquée à la reconstruction d'environnements architecturaux et urbains*. Bordeaux, France, 2002.

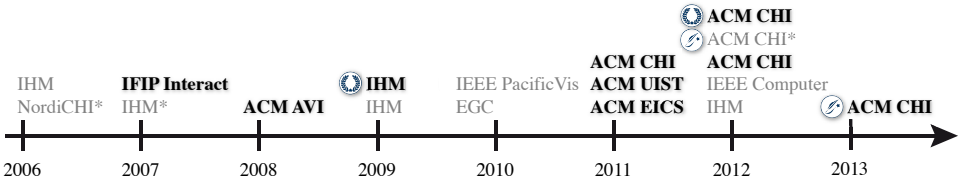
[CH01] Christian Colin and **Stéphane Huot**. "Photomodélisation à l'aide de la géométrie projective." In: *Actes des journées du Groupe de Travail Modélisation Géométrique*. Dijon, France, 2001.

[HC01] **Stéphane Huot** and Christian Colin. *MArINa: 3D reconstruction from images using formal projective geometry*. Research Report 01/1/INFO. École des Mines de Nantes, Jan. 2001.

[Hég+00] Gérard Hégron, Daniel Siret, Christian Colin, Pierre Macé, **Stéphane Huot**, Emmanuel Monin, and Didier Boucard. "Une nouvelle approche informatique de la modélisation architecturale : la restitution de la place Ludovise de Louis." In: *Colloque Victor Louis et son temps*. Paris, France, Dec. 2000.

A.2 SELECTED PUBLICATIONS (2005–2012)

This section contains 9 publications that give an overview of my work since I defended my Ph.D. in 2005. The timeline in Figure A.1 shows those publications in context.



Best paper

Honorable mention

ACM AVI	International Working Conference on Advanced Visual Interfaces
ACM CHI	SIGCHI conference on Human Factors in computing systems
ACM EICS	SIGCHI symposium on Engineering interactive computing systems
ACM UIST	Symposium on User Interface Software and Technology
IFIP Interact	IFIP TC13 Conference on Human-Computer Interaction
IEEE Computer	Special issue on *Beyond the Keyboard*
IEEE PacificVis	IEEE Pacific Visualization Symposium
NordiCHI	Nordic forum for Human-Computer Interaction research (ACM proceedings)
IHM	Francophone conference for Human-Computer Interaction (ACM proceedings)
EGC	Francophone conference for Knowledge Extraction and Management

Figure A.1: Selected publications (in bold) are available in this document. Other publications can be downloaded from the publisher's Digital Library or the HAL-INRIA open archive: http://hal.inria.fr/aut/Stéphane+Huot/. Publications marked with a * are short papers (refereed, and archived in proceedings).

p. 110 **Stéphane Huot** and Eric Lecolinet. "Focus+Context Visualization Techniques for Displaying Large Lists with Multiple Points of Interest on Small Tactile Screens." In: *Proceedings of **11th IFIP TC13 International Conference on Human-Computer Interaction***. **Interact 2007**. Springer Verlag, Lecture Notes in Computer Science, Sept. 2007, pp. 219–233. http://hal.inria.fr/docs/00/55/05/98/PDF/SnaiList-Interact2007.pdf

p. 124 Anne Roudaut, **Stéphane Huot**, and Eric Lecolinet. "TapTap and MagStick: Improving One-Handed Target Acquisition on Small Touch-screens." In: *Proceedings of **the 9th International Working Conference on Advanced Visual Interfaces***. **AVI '08**. ACM, May 2008, pp. 146–153. http://hal.inria.fr/docs/00/55/06/94/PDF/TapTap-AVI08.pdf

p. 132 Caroline APPERT, **Stéphane Huot**, Pierre DRAGICEVIC, and Michel BEAUDOUIN-LAFON. "FlowStates: Prototypage d'applications interactives avec des flots de données et des machines à états." In: *Proceedings of IHM 2009, 21ème conférence francophone sur l'Interaction Homme-Machine*. ACM, International Conference Proceedings Series, Oct. 2009, pp. 119–128. *Best Paper*. http://hal.inria.fr/docs/00/53/85/98/PDF/FlowStates.pdf

p. 143 **Stéphane Huot**, Olivier CHAPUIS, and Pierre DRAGICEVIC. "TorusDesktop: Pointing via the Backdoor is Sometimes Shorter." In: *Proceedings of the **29th international conference on Human factors in computing systems***. **CHI '11**. Vancouver, CA: ACM, 2011, pp. 829–838. http://hal.archives-ouvertes.fr/docs/00/59/12/95/PDF/CHI11-torus-avf.pdf

p. 153 Emmanuel PIETRIGA, **Stéphane Huot**, Mathieu NANCEL, and Romain PRIMET. "Rapid Development of User Interfaces on Cluster-Driven Wall Displays with jBricks." In: *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*. EICS '11. Pisa, Italy: ACM, 2011, pp. 185–190. http://hal.inria.fr/docs/00/58/54/79/PDF/jbricks-eics11.pdf

p. 159 Pierre DRAGICEVIC, **Stéphane Huot**, and Fanny CHEVALIER. "Gliimpse: Animating from Markup Code to Rendered Documents and Vice-Versa." In: *Proceedings of the **24th ACM Symposium on User Interface Software and Technology***. **UIST 2011**. Santa-Barbara, CA, USA: ACM, 2011, pp. 257–262. http://hal.inria.fr/docs/00/62/62/59/PDF/GlimpseA.pdf

p. 165 Julie WAGNER, **Stéphane Huot**, and Wendy E. MACKAY. "BiTouch and BiPad: Designing Bimanual Interaction for Hand-held Tablets." In: *Proceedings of the **30th international conference on Human factors in computing systems***. **CHI '12**. Austin, Texas, USA: ACM, 2012, pp. 2317–2326. http://hal.inria.fr/hal-00663972/PDF/bipadA.pdf

p. 175 Emilien GHOMI, Guillaume FAURE, **Stéphane Huot**, Olivier CHAPUIS, and Michel BEAUDOUIN-LAFON. "Using Rhythmic Patterns as an Input Method." In: *Proceedings of the **30th international conference on Human factors in computing systems***. **CHI '12**. Austin, Texas, USA: ACM, 2012, pp. 1253–1262. *Best Paper*. http://hal.archives-ouvertes.fr/docs/00/66/39/73/PDF/CHI12-ewe-halv1.pdf

p. 185 Julie WAGNER, Mathieu NANCEL, Sean GUSTAFSON, **Stéphane Huot**, and Wendy E. MACKAY. "A Body-centric Design Space for Multi-surface Interaction." In: *Proceedings of the **31st international conference on Human factors in computing systems***. **CHI '13**. Paris, France: ACM, 2013, 10 pages, to appear. *Honorable Mention*. http://hal.inria.fr/hal-00789169/PDF/BodyScape-Hal.pdf

Focus+Context Visualization Techniques for Displaying Large Lists with Multiple Points of Interest on Small Tactile Screens

Stéphane Huot[1,2], and Eric Lecolinet[1]

[1] GET/ENST – CNRS UMR 5141, 46 rue Barrault, 75013 Paris, France
[2] LRI (CNRS) & INRIA Futurs, Bât. 490, Univ. Paris-Sud, 91405 Orsay, France
Stephane.Huot@lri.fr, Eric.Lecolinet@enst.fr

Abstract. This paper presents a focus+context visualization and interaction technique for displaying large lists on handheld devices. This technique has been specifically designed to fit the constraints of small tactile screens. Thanks to its spiral layout, it provides a global view of large lists on a limited amount of screen real-estate. It has also been designed to allow direct interaction with fingers. This technique proposes an alternative to multi-focus visualization, called "augmented context", where several objects of interest can be pointed up simultaneously. We propose two implementations of this approach that either use spatial or temporal composition. We conducted a controlled experiment that compares our approach to standard scrollable lists for a search task on a PDA phone. Results show that our technique significantly reduces the error rate (about 3.7 times lower) without loss of performance.

Keywords: Mobile interfaces, focus+context visualization, spiral layout, finger interaction, one-handed interaction.

1 Introduction

During the last years, mobile devices have evolved from cell phones to handheld computers, introducing new usages, but also challenging new problems. Whereas hardware buttons were efficient to manage most of basic phones functionalities, modern handheld devices (PDA, PDA-Phones, etc.) can manage more and more data and include new functions that require more efficient interaction. However, most applications for handheld devices still rely on traditional GUI paradigms that have been developed for desktop computers. This model does not fit well on small devices because of their limited screen size and input capabilities. In fact, interacting with large quantities of data with small devices is still a challenging problem that is aggravated when several objects of interest must be highlighted simultaneously. This case is related to multi-focus visualization, but with the additional constraint that a very limited amount of screen real-estate can be used to represent multiple points of interest. Finally, the fact that handheld devices should be usable in mobility conditions is another key factor. Most applications for tactile screen devices require using a stylus, a way of interacting that is not very well suited for mobile interaction.

The next section describes some limitations of graphical representations and interaction techniques that have been proposed so far. Section 3 introduces some general principles we propose to solve these limitations. They have been implemented in two ways that are presented in the same section. The following sections include an experimental evaluation, related work and the conclusion and perspectives.

2 Challenges of Mobile Devices

2.1 Stylus Interaction

Stylus interaction is not very well suited for handheld devices for several reasons. It requires users to pull out the stylus, an operation that may be uneasy in mobility conditions (besides, users must be careful not to lose the stylus). Using a stylus also makes it impossible to interact with only one hand and it requires interaction to be performed very precisely by clicking on tiny widgets (thus reducing performance according to Fitts' Law [8]). This causes pointing errors, especially when the user is moving, and users may find this interaction style unpleasant because it requires too much attention [19]. Despite these limitations, most existing GUIs on mobile devices make use of tiny WIMP widgets and require using a stylus.

Ideally, the user should be able to interact directly with fingers by using only one hand while performing another activity [15]. But "thumb interaction" arouses several problems:

- The small size of touch screens limits tapping efficiency, especially when there are many UI objects on the screen. Conversely, the large contact area between the finger and the tactile screen makes tapping imprecise.
- Finger interaction causes screen occlusion.
- Some screen regions may be hard to reach, especially when the thumb is used.

To solve these problems, we propose an interaction technique based on a "Touch-Drag-Lift" strategy. Instead of directly tapping targets on the screen, the user manipulates a shifted cursor in a continuous manner. This technique, which is presented in section 4, limits screen occlusion while enhancing pointing precision.

2.2 Compact and Multi-focus Representations on Small Screens

The most common strategy for displaying a large amount of data on small screens consists in using scrollable viewports that reveal a subpart of the data. But viewports have several annoying drawbacks, especially on small screens. First, they are not well adapted for multi-focus representation as they rely on clipping. Clipping prevents highlighting a point of interest that is not currently located in the viewport (although some approaches, such as [3], show marks on scrollbars to indicate where interesting data could be found). As viewports hide most of the data, they provide very limited contextual information to users. This is especially true on small screens, where few items can be displayed (as shown in **Fig. 5d**). Focus+context (F+C) visualization techniques partially fit these constraints with a visual layout that combines:

- A *focus area*, where the data of most interest is displayed at full size or with full details,
- A *context area*, which is a peripheral zone where elements are displayed at reduced size or in a simplified way.

Depending on the techniques, the difference in representing elements in the focus and the context areas can be purely geometrical or make use of the semantics of the data [11], as semantic zooming [20].

F+C techniques enable to represent much more elements simultaneously than representations based on viewports. Elements that have a high degree of interest at a given time are displayed with a size that is large enough to make them more readable. Such techniques provide effective solutions to compensate the lack of screen real-estate on small displays. However, they generally do not suffice to solve the multi-focus problem when it is necessary to highlight elements that are located anywhere in the representation. This ability is crucial for applications that need to notify alerts or useful dynamical information about the elements they display. Some F+C techniques provide several focal points [23,24] but do not solve our problem efficiently because elements that must be highlighted may stay in the context zone. They are also likely to use too much space as the number of highlighted points increases.

3 New Techniques for Displaying Lists on Small Screens

3.1 Spiral Representation and Augmented Context

Fig. 1. Spiral sectors and DOI zones

The list visualization techniques we propose use a spiral layout divided into sectors to display items (**Fig. 1**). In contrast with linear clipped lists, this compact layout makes it possible to show a large number of items. However, all items can not be shown if their number exceeds a certain limit (which is the number of sectors in the spiral). Moreover, items located in the innermost revolutions cannot be displayed with full details because their text labels would be too large. We thus combined this spiral representation with a focus+context strategy to increase the length of displayable lists in a limited amount of screen real-estate. It is interesting to notice that a tabular layout could be even more space efficient than a spiral representation. However, its 2D nature would not make it very well suited for representing lists, especially when F+C

visualization techniques are used. Besides, a spiral layout is well adapted to thumb interaction because it is contained in a circular area that is easy to reach when holding the device in one hand.

F+C visualization techniques generally use geometrical distortions to display objects with a high degree of interest (DOI) at a larger scale in the focus area (like FishEye techniques [9]). In our case, such distortions would make textual data illegible, especially on low resolution handheld screens. We favored a semantic approach [20] where the graphical representation depends on data characteristics. The DOI of an item depends on its location in the spiral, on the spelling of its name and its neighbors' names, and on its intrinsic characteristics (as for instance, the requirement that it must remain visible). The visibility function does not involve geometrical distortions but controls if the label of this item is shown and how many letters are displayed (their number depends on the position of the item relatively to the focus and context zones). The label of an item located in the outermost revolution (the focus zone) is fully visible. Conversely, only the first letters of item labels are shown in the innermost revolutions (the context zone) where 3 letters are shown, then 2 and then only 1 (**Fig. 1**).

Item visibility in the context zone also depends on the spelling of labels. Items which labels start with the same letters are collapsed together in order to use less space. They can be expanded by moving them interactively in the focus zone. However, certain items conveying important information can remain always visible, whatever their location in the spiral. We call this principle "augmented context" as it allows to highlight objects of interest in the contextual view. Thanks to this property, there is no need to use intrusive or space consuming interaction techniques such as pop-ups or a dedicated label zone. Visual artifacts such as colors, blinking effects, special marks can also be used to provide various information about these items.

The next subsections describe two variants of the principles proposed so far. The first variant is founded on a purely spatial approach while the second makes use of a temporal approach. A preliminary version of the first variant was presented in [12].

3.2 A Spatial Focus+Context Strategy: SpiraList

This technique is based on a spatial strategy that follows a focus+context paradigm. Items are displayed in alphabetical order on the spiral layout in such a way that the first visible item and the last one are contiguous in the list (**Fig. 2a**). The *focus area* is located on the bottom of the outermost revolution so that the labels are fully visible on a handheld screen in portrait mode (in landscape mode, the focus appears at the right of the spiral). The selected item ("Piotr" on **Fig. 2a**) appears in the middle of the focus zone. The rest of the spiral contains the *context area*. According to the techniques presented in the previous subsection, item labels are progressively clipped and grouped while advancing inside the spiral. To avoid text overlapping, the labels are rotated according to their position in the spiral. Some visual cues are used to improve the representation: labels that start with a different letter than the previous label in the list are displayed in bold font. Similarly, colored sectors indicate items that are never collapsed because they convey some useful information (for instance a group call or an email if items represent persons).

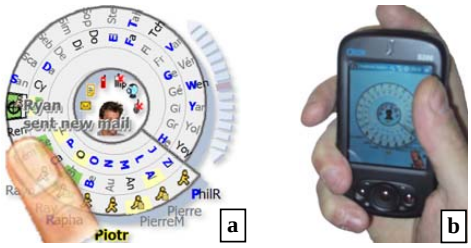


Fig. 2. SpiraList: a screenshot (a) and a picture of SpiraList in action (b).

Moving the cursor over items triggers a tooltip that shows their complete labels (**Fig. 2a**). Once the cursor reaches the item the user is looking for (or the group of items that contains it) he can move this item to the focus area just by releasing the finger. A fast animation is then performed to help the user to understand how the list is reorganized. If the desired item is grouped with other ones, the same action expands the group and moves corresponding items to the focus (or close to it if the group contains too many items). Another selection is then necessary to select the appropriate item. Alternatively, the list can be "scrolled" inside of the spiral by dragging the blue arrows located on its right hand side (**Fig. 2a**). This technique is useful to select items that are already located in the focus area, or to get information on their neighbors.

A noticeable characteristic with our technique lies in the way of changing the focus. Instead of most other focus+context techniques, the focus area always remains at the same place but the data moves to appear in the focus zone. This design tries to better take into account the form factor of handheld devices (**Fig. 2b**). Because of the limited available space and the rectangular shape of the screen, it is advantageous to display details in a fixed area below the spiral: this makes it possible to use the full screen width to display the spiral.

3.3 A Temporal Focus+Context Strategy: SnailList

A preliminary pilot study has shown that the SpiraList technique was well appreciated by users and considered to be time efficient in their subjective evaluations. However, actual measurements gave slightly disappointing results for lists containing more than 100 items. While this technique provides an efficient solution for displaying data at arbitrary locations in a global view, it is most suited for lists that do not exceed this size. These results lead us to analyze how to improve this initial design to obtain better performance for larger lists (we considered lists containing up to 500 items in the experiments).

The most obvious problem was that the automatic grouping algorithm used in the context zone was not efficient enough for large lists. This is because most items are likely to be collapsed in this case, thus reducing the chances of selecting the appropriate item with a single click. Moreover, the number of items that are collapsed together can be quite large if they start with a frequently used letter. The desired item may then appear quite far from the focus zone, thus requiring a second visual search to find it. It can even remain collapsed with other items, thus requiring further user

interaction. Besides, all the revolutions of the spiral are filled up with items when large lists are used. This may cause information overload and degrade visual search performance. Finally, we also identified text rotation as a factor that could reduce performance. Some previous work on tabletop displays [26] has shown that the effect of text orientation was somewhat overestimated in former work performed in the field of perception [16]. However, this effect may be significant in a searching task that requires scanning many elements with different orientations.

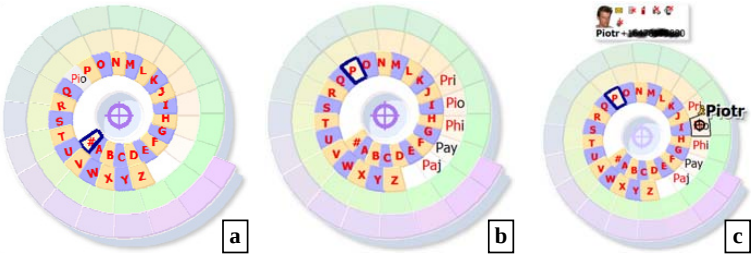


Fig. 3. SnailList: (a) context, (b) intermediate view and (c) focus.

The new proposed technique, called *SnailList*, is based on a so-called "temporal" strategy. In contrasts with the previous technique it does not provide a focus and a context zone with a variable level of detail according to the location in the spiral. Instead, the context zone appears first in the innermost part of the spiral and is always shown at the same location (**Fig. 3a**). This zone contains all alphabetical letters and the objects of attention that must remain visible, like the "Pio" item in **Fig. 3a** (according to our augmented context principle). The level of detail is the same for all items (only one letter appears) and none of them are collapsed. The user must select one of them to make the focus to appear. If the chosen item is an alphabetical shortcut the list of all items starting with this letter appears on the spiral (**Fig. 3b**). This new list is displayed at the end of the context zone and it can be seen as an intermediate level of detail between context and focus. The three first letters of the items labels are displayed without text rotation. The user must then find the desired item and click on it to select this element. This makes it to appear with full details in the focus area, at the top of the spiral as shown in **Fig. 3c**.

The main difference with the former technique is that it deals with three representations, corresponding to three different levels of detail (context, intermediate and focus) that appear successively in time. Hence, the level of detail depends on time (*i.e.* the number of successive selections) rather than spatial location and it does not vary progressively with the revolution depth. This reduces the number of items that are displayed simultaneously, and thus, the number and the difficulty of visual searches. Therefore, searching an item follows an incremental alphabetical search scheme. The first search consists in finding the first letter of the desired item in the context zone. As it is always displayed in the same way, users may learn the approximate positions of letters after some practice time. The second visual search takes more time because the user must find the appropriate element in a list that has variable length and content, and that may contain items with the same three first

letters (the full label must then be read in the tooltip). However, as the content of this list is filtered by the selected first letter, the number of items that the user must scan is noticeably smaller than with the first technique.

This intermediate view is not large enough if the list contains too many elements that start with the same letter (45 items by default). The number of visible items could be increased by augmenting the number of revolutions but this would decrease font size and make labels harder to read. Fortunately, this case seldom occurs except for very large lists or badly balanced ones. Alternate solutions could consist in using the same grouping strategy as in SpiraList or in improving filtering by adding supplementary alphabetical items in the context zone (e.g. “Aa” for accessing items that starts with “A” to “Al” and “Am” for remaining items).

4 Interacting with Fingers

The visualization techniques proposed so far have also been designed to fit the need of one-handed finger interaction, a very useful feature in mobile usage [15,19]. As explained in section 2, finger interaction (and more especially thumb interaction) brings two annoying problems: imprecise tapping and occlusion.

Imprecision. Imprecise tapping is due to the large contact area between the finger and the screen. It is thus difficult for the user to figure out which object is going to be selected. A solution can consist in using the touch screen as a relative input device. The position of a cursor in the screen space is then controlled by the movements of the finger in the motor space. Imprecision mostly affects the absolute location of press gestures. Drag gestures allow controlling the relative shifting of the cursor much more precisely. They are also less likely to be triggered accidentally than tapping [21].

Our interaction technique is based on this idea but still preserves some kind of absolute positioning. It follows a “touch-drag-lift” paradigm, which is related to the “take-off” technique [22]. *Touching* the screen anywhere with the thumb makes the cursor to appear above this location. The user can then *drag* this cursor all over the screen. Finally, *lifting* the thumb performs an interaction that depends of the last position of the cursor (over a list item, in an action triggering zone, etc.). Cursor moves are stabilized to withdraw the lack of precision of touch-screens: as continuous interaction with a finger involves multiple moving contact points and can lead to erratic and vibrating cursor jumps, the cursor trajectory is smoothed by means of a real-time adaptive low-pass filter that is optimized to avoid lag and does not slow down the cursor.

This technique makes it possible to achieve very good accuracy, even in the case of displaying dense information. It is also quite fast as it combines absolute (but imprecise) positioning with a relative (but precise) correction of this position.

Finger Occlusion. A fixed (but user adjustable) vertical offset is applied to the cursor as a simple but efficient way to avoid fingertip occlusion. An adaptive horizontal offset is also calculated to improve access to the left and right borders of the screen. It is computed according to the equation in **Fig. 4**:

$$\delta x = \frac{\left(x - W/2\right)^3}{W * \lambda}$$

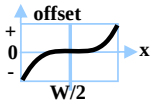


Fig. 4. Adaptive horizontal offset function.

In this equation, x is the non-corrected position along the horizontal axis, W the width of the screen and λ the strength of the offset. This offset grows as the cursor goes away from the center of the screen as shown on **Fig. 4** (right). The λ parameter controls the offset around the central position. An experimental value of 80 gave good results: the offset is then unnoticeable around the center of the screen and augments slightly when the cursor reaches the edges.

5 Experiment

We conducted an experiment to compare the efficiency of SnailList (SN) and standard Scrollable Lists (SC) for finding items in one-handed interaction conditions.

5.1 Experimental Setup

Subjects. 10 unpaid subjects, aged from 24 to 40 (7 males and 3 females) served in the experiment. 4 of them are daily users of handheld devices (PDA or PDA-Phones), 3 are occasional users and 3 had never used one. One of them is left-handed. At the beginning of each session, the two techniques were explained to the participant during 10 minutes and s/he performed 10 training tasks with each technique.

Apparatus. We used a QTEK S200 PDA (TI 200MHz CPU and 320x240 tactile screen). The program, written in C#, implements the SnailList widget and an instrumented version of the standard WM 5.0 Contacts application. It automatically logs actions performed by the subject (taps, lifts, items fly over or selection, etc.), errors and the time needed to complete a trial.

Task. The task performed by the participants consisted in retrieving contacts from an address book in a mobility situation. They had to operate with the thumb of the hand that carries the device. However, to limit subjects' weariness, they were seated and could lean their arm (but not their hand) on a table or on their legs.

At the beginning of each trial, the name of the item to find was presented to the subject in the same way as it appears in the contact list (Name then First name). When s/he was ready, the subject started the trial by tapping the "Start" button (**Fig. 5a**). The name remained displayed during the whole trial at the top of the list (**Fig. 5c&d**).

When using SC (**Fig. 5d**), the user can use the scrollbar and the shortcut buttons above the list. These buttons make it possible to scroll the list automatically at certain alphabetical positions. When s/he finds the correct contact, s/he just has to tap on it to validate the trial. With SN (**Fig. 5c**), the user can display different parts of the list by lifting the cursor over the appropriate alphabetical item in the context zone. When s/he finds the desired contact, s/he must lift the cursor over it to validate the trial.

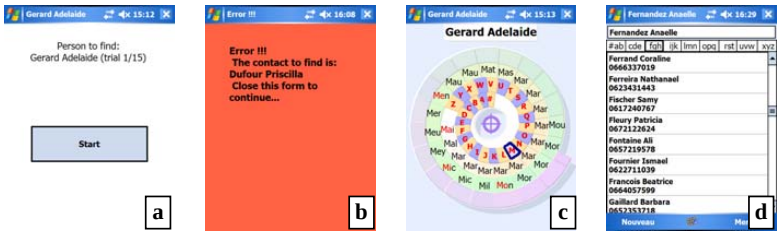


Fig. 5. Experiment: start of a trial (a), error form (b), SnailList (c) and Scrollable List (d).

When an erroneous contact is selected, an error notification appears (**Fig. 5b**) and the user is invited to close it to pursue the trial. When the right item is found, a new trial starts and the subject continues the experiment until all trials are performed.

5.2 Hypothesis and Experiment Model

Our hypotheses are:

Hypothesis 1: SnailList (SN) has equivalent or better performance than standard Scrollable Lists (SL) for retrieving items.

Hypothesis 2: SnailList (SN) generates fewer errors than Scrollable Lists (SC).

We chose three different sizes (100, 250 and 500 items) to compare the performances of the techniques. This choice was based on observations informally gathered by colleagues working on mobile phone usage. 100 items was observed to be an appropriate number for ordinary users, 250 for professionals that use mobile phones frequently and 500 as an upper limit for such usage. Those conditions involve 2 independent variables: *Techniques* (SC and SN) and *List Size* (100-250-500):

2 techniques (*SN* and *SC*) x 3 list sizes (*100, 250* and *500 items*) = 6 tasks

After a training of 10 trials with each technique (with 250 item lists), each subject performed 15 trials for every task. We grouped the tasks in 3 blocks of 30 trials, one block for each list size with the two technique conditions, in balanced order. Inside each block, subjects performed successively the 15 trials with the 2 techniques.

1 block of 30 trials x 3 list sizes = 90 trials per subject

We built several dummy contact lists using randomly chosen French names. As both techniques display lists in alphabetical order, we tried to minimize the effect of the alphabetical balance of the lists on experimental results. For each technique, the index of difficulty (ID) for searching a given item depends on N_l (the number of items that start with letter l) and on R_{it} (the rank of the item in the sub-list). When using SC to find an item that starts with letter l, a high value of N_l is likely to increase scanning and scrolling times (scrolling time depends on R_{it}). For SN, a high value of N_l augments the visual density and a high value of R_{it} augments the scanning time. To ensure that experiment results would not be biased by this balance effect, the same lists were used for each condition for a given subject and the 15 trials were organized in such a way that N_l and R_{it} were the same for each technique.

To summarize, our experimental design has 2 independent variables (*Technique* and *List Size*) and 2 dependent variables: *Trial Completion Time* and *Number of Errors*. Finally, we also asked the participants to answer a short questionnaire. We asked them which of the 2 techniques they preferred (and why) and which technique seemed to be more efficient and errorless.

5.3 Results and Discussion

We performed a repeated measures ANOVA on the *Completion Time* and found no significant main effects for *Technique* ($F_{(1,9)} = 0.69$, p=0.4164). As shown in **Fig. 6a**, mean performance time are close for the 2 techniques when considering global conditions (list sizes). This result validates our first hypothesis when considering the 3 sizes of lists together (SN performs equally well as SC).

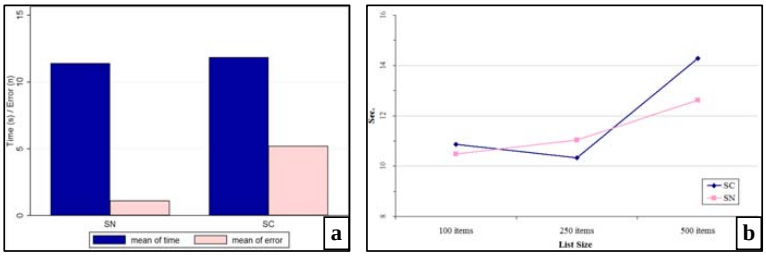


Fig. 6. Mean Time and Errors by Technique (a) and Mean Time by Technique and List Size (b)

We found a significant main effect on *Completion Time* for *list size* ($F_{(2,18)} = 12.43$, $p < 0.001$). Paired t-tests indicated that the *100 items* and *250 items* size conditions are faster than the *500 items* condition when considering both techniques ($t_{(9)} = 2.3892$, $p < 0.1$ and $t_{(9)} = 2.4886$, $p < 0.05$). When restricting to the *SC* technique, paired t-tests indicated that *100* and *250 items* conditions are faster than *500 items* condition ($t_{(9)} = 1.7485$, $p < 0.05$ and $t_{(9)} = 2.3169$, $p < 0.1$), whereas for the *SN* technique the difference is not significant ($t_{(9)} = 1.6292$, $p < 0.1$ and $t_{(9)} = 1.0828$, $p < 0.2$). This tendency is visible in the line graph of **Fig. 6b**.

Even if there is no significant interaction effect of *Technique*Size*, those results suggest that the completion time for retrieving an item with SnailList augments slightly when the list size increase whereas the completion time augments more abruptly with the standard Scrollable List. These results are not significant enough to formally validate the fact that our technique outperforms standard Scrollable Lists for very large lists, but they seem to indicate that SnailList is especially robust to scaling (increasing list size) for what concerns performance.

We performed a repeated measures ANOVA on the *Number of Errors* and found a significant main effect for *Technique* ($F_{(1,9)} = 18.74$, $p < 0.001$), without significant interaction effects for *Technique*Size*. T-test confirmed that the *SC* technique produced significantly more errors than the *SN* technique ($t_{(9)} = 4.1778$, $p = 0.0001$). This result validates our second hypothesis that SnailList produces fewer errors than standard Scrollable List in mobility situations. In fact, the mean of errors with SN

(when considering all list sizes) is 4 times less than with SC. The graph in **Fig. 7a** shows that this ratio is close to 7 in the case of 500 item lists.

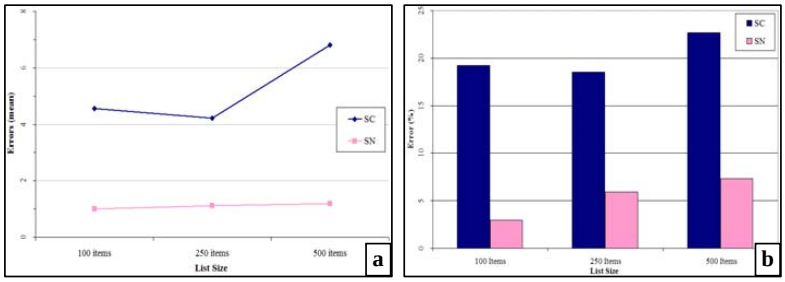


Fig. 7. Mean of errors (a) and Percentage of faulty trials (b) by Technique and by List Size

If we consider a trial to be erroneous if at least one error occurred during it, we obtain a percentage of 20.1% faulty trials for Scrollable List against 5.4% with SnailList. This makes SnailList 3.7 times more accurate than Scrollable List. **Fig. 7b** shows that this percentage is almost constant for Scrollable Lists when the size of the list increases, while it is almost proportional to the list size when SnailList is used. This result suggests that the design of SnailList make its index of difficulty mainly dependent on the list size. In fact, errors with SnailList are mainly due to confusion between adjacent items with close labels and this case is likely to occur more frequently for large lists than for small ones. Conversely, the index of difficulty of Scrollable Lists is almost constant and at a high value in mobility situation for all list sizes. We attribute this result to the imprecision of tapping when performed with fingers (we noticed that users accidentally validated items whereas they wanted to scroll or to click on a button).

In subjective evaluations, most of the participants were convinced that they completed tasks faster with SnailList and with fewer mistakes than with Scrollable List (9 of 10). These estimations correspond to reality for errors, but not for completion time (performances was mostly similar for both techniques). 9 of 10 subjects said that they preferred using SnailList over the scrollable list. The novelty of the technique aside, they argued that its design made it well-adapted to finger interaction on a small screen. They also highlighted the fact that SnailList was less frustrating to use because they made less errors than with the scrollable list.

6 Related Work

Designing interactions techniques for mobility requires taking into account devices form-factor constraints and users' tasks together. Recent studies that focused on this topic [15,19] showed that one-handed mobile interaction is preferable because it is less distracting for the user (a principle called "Minimal Attention User Interfaces" in [19]). These studies have also emphasized the use of the thumb as a good way for interacting in such conditions. Other approaches consist of augmenting handheld

devices with alternative input modes, like in physical embodied techniques [10]. However, thumb interaction is more feasible and well-adapted to currently available devices that provide a tactile screen.

In *AppLens* and *LaunchTile* [13], the authors introduced one-handed interaction techniques for controlling mobile device applications. These interactions rely on discrete zooming and gestures performed with the thumb. However, these techniques do not have the same purpose and have not been designed to display large quantities of data as the ones we propose in this paper. In *FaThumb* [14], the same authors considered this problem and proposed an efficient one-handed data seeking technique. But this approach, which can be seen as an alternative to textual queries, is not intended to solve the visualization problems we presented in this paper.

This spiral layout was first introduced in information visualization and extensively studied by Carlis and Konstan in [6]. They took advantage of the concentric layout of the Archimedean spiral for the visualization and interpretation of serial and periodic data. The same idea is used in *SpiraClock* [7], with a spiral that is augmented by an analog clock to provide a non intrusive display for upcoming temporal events. *RankSpiral* [25] also uses spiral layout to display large results of multiple search engines. This last approach is the most similar to our but it was designed for desktop PCs and does not take into account the display and input constraints of small devices.

In [6], Carlis and Konstan suggested the idea of extending the spiral layout with a focus+context scroll mechanism. We considered this idea as a good way for displaying more items in the spiral. Focus+context methods can be used to improve the overall visibility of large lists [4,17], but only a few items are readable because of their focus-dependant geometrical zoom, especially when a small screen is used. The compact display of our approach achieves the same level of visibility. But its non-geometrical strategy, which is somewhat related to semantic zooming, overcomes the legibility problem that arises with F+C techniques based on geometrical zooming. In [18], one of the problems addressed by the authors is close to ours but on standard displays: displaying large binary trees with several points of interest. They had shown that *Pan & Zoom* techniques outperform F+C techniques for navigating in this case. However their techniques rely on geometrical deformations that can have a strong impact on interaction performance and deals with 2D localization problems (as in [1]). Those conditions are quite different than ours that deal with 1D linear data on small screen, without geometrical deformations.

Various advanced new interaction techniques have been proposed for touch screens in recent studies (such as [2]) but they are not specifically designed for handhelds. Our work extends the “Take-off” technique [22] in the case of mobile devices. In particular, our adaptive horizontal cursor offset, that could be annoying on large touch screens [2], gives promising results on small screens.

7 Conclusion and Future Work

We have presented *SpiraList* and *SnailList*, two new visualization and interaction techniques for manipulating large lists on handheld devices. These techniques provide (a) one-handed operation using the thumb, a feature that is especially well suited for mobile interaction; (b) focus+context visualization and an augmented representation

of context that allows to display objects of interest permanently. An experimental comparison of standard scrollable lists and *SnailList* on a PDA phone has shown that this new technique significantly reduces the error rate (to about 3.7 times lower) without loss of performance when interacting with the thumb.

The proposed technique for finger and thumb interaction improves the "Take-off" method [22] and allows precise interaction even when many small objects are displayed on the screen. It is based on a differentiation between the visual space and the motor space of the touch screen device. An interesting extension would consist in studying how methods that perform advanced control/display ratio adaptation, such as semantic pointing [5], could be integrated in our technique to improve selection.

We are now aiming at improving the performance time of the proposed technique. Items could for instance be filtered according to their second and third letters by selecting the alphabetical items located in the center of the spiral multiple times. This successive dichotomy approach would reduce the number of items displayed in the peripheral zone and could improve search time. Finally, we also plan to perform further experiments with more participants to obtain a more detailed comparison of the considered techniques for large lists.

Acknowledgments. This work has been done in collaboration with Guillaume Dorbes and Bruno Legat from Alcatel-Lucent R&D. We want to thank them for their useful advices and their help in evaluating SpiraList and SnailList. Many thanks to the people that accepted to participate in our experiment and also to Gilles Bailly and Anne Roudaut for their useful help while conducting it.

References

1. Baudisch, P., Rosenholtz, R.: Halo: A Technique for Visualizing Off-Screen Locations. In: Proc. Of CHI'2003 (Fort Lauderdale, FL, USA), ACM, New York (2003) 481–488
2. Benko, H., Wilson, D. A., Baudisch, P.: Precise Selection Techniques for Multi-Touch Screens. In: Proc. of CHI'2006 (Montréal, CA), ACM, New York (2006) 1263–1272
3. Bederson, B.B., Clamage, A., Czerwinski, M. P., Robertson, G. G.: DateLens: A fisheye calendar interface for PDAs. In: ACM Transactions on Computer-Human Interaction (TOCHI), ACM, New-York (2004), 11(1):90–119
4. Bederson, B. B.: Fisheye Menus. In: Proc. of UIST 2000 (San Diego, USA), ACM, New York (2000) 217–226
5. Blanch, R., Guiard, Y., Beaudouin-Lafon, M.: Semantic pointing: improving target acquisition with control-display ratio adaptation. In: Proc. of CHI'2004 (Vienna, Austria), ACM, New York (2004) 519–526
6. Carlis, J.V., Konstan, J.A.: Interactive Visualization of Serial Periodic Data. In: Proc. of UIST 98 (San Francisco, USA), ACM, New York (1998) 29–38
7. Dragicevic, P., Huot, S.: SpiraClock: A Continuous and Non-Intrusive Display for Upcoming Events. In: Extended Abstracts of CHI'02 (Minneapolis, USA), ACM, New York (2002) 604–605
8. Fitts, P.M., The information capacity of the human motor system in controlling the amplitude of movement. In: Journal of Experimental Psychology, APA, Washington (1954), 47(6):381–391
9. Furnas, G.: Generalized Fisheye Views. In: Proc. of CHI'86 (Boston, USA), ACM, New York (1986) 16–23

10. Harrison, B.L., Fishkin, K.P., Gujar, A., Mochon, C., Want, R.: Squeeze me, hold me, tilt me! An exploration of manipulative user interfaces. In: Proc. of CHI'98 (Los Angeles, USA), ACM, New York (1986) 17–24
11. Herman, I., Melançon, G., Marshall, M.S.: Graph Visualization and Navigation in Information Visualisation: a Survey. In: IEEE Transactions on Visualization and Computer Graphics, IEEE (2000), 6(1):24–43
12. Huot, S., Lecolinet, E.: SpiraList: A Compact Visualization Technique for One-Handed Interaction with Large Lists on Mobile Devices. In: Proc. of NordiCHI'06 (Oslo, Norway), ACM, New York (2006) 445–448
13. Karlson, A. K., Bederson B. B., SanGiovanni, J.: AppLens and LaunchTile: Two Designs for One-Handed Thumb Use on Small Devices. In: Proc. of CHI'2005 (Portland, USA), ACM, New York (2005) 201–210
14. Karlson, A.K., Robertson, G.G., Robbins, D.C., Czerwinski, M.P., Smith, G.R.: FaThumb: a facet-based interface for mobile search. In: Proc. of CHI'2006 (Montréal, Canada), ACM, New York (2006) 711–720
15. Karlson, A., Bederson B. B., Contreras-Vidal, J.: Studies in One-Handed Mobile Design: Habit, Desire and Agility, Tech. Report 2006-02, HCIL, Washington (2006)
16. Koriat, A., Norman, J.: Reading Rotated Words. In : Journal of Experimental Psychology: Human Perception and Performance, APA, Washington (1985), 11(4):490–508
17. Lecolinet, E., Nguyen, D.: Focus+context visualization of zoomable hierarchical lists. In: Proc. of French-Speaking Conference on Human-Computer Interaction (IHM 2006) (Montréal, CA), ACM, New York (2006) 195–198
18. Nekrasovski, D., Bodnar, A., McGrenere, J., Guimbretière, F., Munzner, T.: An evaluation of pan & zoom and rubber sheet navigation with and without an overview. In: Proc. of CHI'06 (Montréal Canada), ACM, New York (2006) 11–20
19. Pascoe, J., Ryan, N., Morse, D.: Using while moving: HCI issues in fieldwork environments. In: ACM Transactions on Computer-Human Interaction (TOCHI), ACM, New York (2000) , 7(3):417–437
20. Perlin, K., Fox, D.: Pad: an alternative approach to the computer interface. In: Proc. Of SIGGRAPH '93, ACM, New York (1993) 57–64
21. Pirhonen, A., Brewster, S., and Holguin, C.: Gestural and audio metaphors as a means of control for mobile devices. In: Proc. of CHI'02 (Minneapolis, USA), ACM, New York (2002) 291–298
22. Potter, R.L., Weldon, L.J., Shneiderman, B.: Improving the Accuracy of Touchscreens: An Experimental Evaluation of Three Strategies. In: Proc. of CHI'88 (Washington, USA), ACM, New York (1988) 27–32
23. Sarkar, M., Snibbe, S. S., Tversky, O. J., Reiss, S. P.: Stretching the rubber sheet: a metaphor for viewing large layouts on small screens. In: Proc. of UIST 93 (Atlanta, USA), ACM, New York (1993) 81–91
24. Shipman, F. M., Marshall, C. C., LeMere, M.: Beyond Location: Hypertext Workspaces and Non-Linear Views. In: Proc. of Hypertext'99 (Darmstadt, Germany), ACM, New York (1999) 121–130
25. Spoerri, A.: RankSpiral: Toward Enhancing Search Results Visualizations. In: Proc. Of InfoVis 2004 (Austin, USA), IEEE (2004) 18–19
26. Wigdor, D., Balakrishnan, R.: Empirical investigation into the effect of orientation on text readability in tabletop displays. In: Proc. of ECSCW 2005 (Paris, France), Springer (2005) 205–224

TapTap and MagStick: Improving One-Handed Target Acquisition on Small Touch-screens

Anne Roudaut[1] Stéphane Huot[2] Eric Lecolinet[1]

Anne.Roudaut@enst.fr, Stephane.Huot@lri.fr, Eric.Lecolinet@enst.fr

[1]TELECOM ParisTech, CNRS LTCI
46 rue Barrault
75013, Paris, France

[2]LRI – Univ. Paris-Sud & CNRS, INRIA
F-91405 Orsay
France

ABSTRACT

We present the design and evaluation of TapTap and MagStick, two thumb interaction techniques for target acquisition on mobile devices with small touch-screens. These two techniques address all the issues raised by the selection of targets with the thumb on small tactile screens: screen accessibility, visual occlusion and accuracy. A controlled experiment shows that TapTap and MagStick allow the selection of targets in all areas of the screen in a fast and accurate way. They were found to be faster than four previous techniques except *Direct Touch* which, although faster, is too error prone. They also provided the best error rate of all tested techniques. Finally the paper also provides a comprehensive study of various techniques for thumb based touch-screen target selection.

Categories and Subject Descriptors

H.5.2. User Interfaces: Input Devices and Strategies, Interaction Styles, Screen Design; D.2.2 User Interfaces

General Terms

Design, Human Factors

Keywords

Mobile devices, one-handed interaction, thumb interaction, touch-screens, interaction techniques.

1. INTRODUCTION

Many mobile devices are now fitted with touch-screens that enable us to interact directly with our fingers. However, most graphical interfaces still require users to click on small widgets by using a stylus. As highlighted in [7, 10], this interaction style is not the best way to interact with small devices in a mobile context: it requires too much attention (especially if the user is moving) and forces users to use both hands (one hand holding the device while the other manipulates the stylus). Ideally, mobile interaction should just require one hand, with the thumb being used for selecting objects. In fact, direct selection on the screen is intuitive and fast, and using only one hand is central as users may perform several simultaneous tasks.

However, direct thumb interaction on small touch-screens raises several issues: a) hand and thumb morphology makes it difficult to reach the corners of the screen; b) the thumb may occlude large parts of the screen that can contain the desired target; c) the relatively large contact zone between the fingertip and the tactile screen makes selection ambiguous, especially in applications that require users to click on tiny widgets for triggering actions. Despite these issues, this *Direct Touch* technique is still the most widely used.

Figure 1. TapTap and MagStick

Alternate techniques have been proposed to improve accuracy and eliminate visual occlusion, but they make the interaction slower and more complex. In this paper, we first present a thorough analysis of the properties of the techniques published so far. We then introduce two novel interaction techniques, **TapTap** and **MagStick** (Fig. 1) that solve the problems raised by our analysis (screen accessibility, visual occlusion and accuracy). Finally, we performed a controlled experiment which proved that our two techniques outperform the ones proposed previously (*Direct Touch*, *Offset Cursor* [11], *Shift* [14] and *Thumbspace* [6]).

2. RELATED WORK

Research on thumb interaction with mobile devices is a relatively recent field. The state of the art thus still largely relies upon research on interaction with regular touch-screens. In spite of recent innovations, the issues of reaching far targets, visual occlusion and accuracy are not yet completely solved.

Target accessibility. The borders of the screen are more difficult to reach [6], especially with the thumb because the morphology of the hand constrains thumb movements. This will degrade interaction in the screen areas that are farthest from the natural thumb extent (*i.e.* the top and left border for a right-handed user). Besides, thumb movements may also be hampered near borders because of the thickness of the device's edges around the screen.

Visual occlusion. When interacting, the finger hides a part of the screen and can even totally occlude small targets. This problem is more pronounced when interacting with one hand because the thumb pivots around the thumb joint and can hide half the screen.

Accuracy. A study [9] showed that 9.2 mm is the minimum size for targets to be easily accessible with the thumb. Some mobile devices, such as the iPhone or the HTC Touch, rely on a limited set of large buttons. But, this approach reduces the number of targets because of the lack of screen estate and is thus inappropriate for many applications. Besides, the exact location of the pointer tends to be imprecise because of the large contact surface between the thumb and the screen. As current touch-screen hardware technology computes the barycenter of the multiple contact points, small variations in the way of pressing the thumb can provoke jerky movements of the pointer.

In the following, we group the existing attempts to solve those issues in three categories depending on how they handle input: "tapping","dragging"and "hybrid" techniques.

2.1 Tapping Techniques

Tapping techniques capture the position of the pointer when the thumb touches the screen. The most widely used technique on regular or small touch-screens, Direct Touch, relies on this intuitive principle. The user must tap the screen precisely at the location where the target is displayed. This technique is fast but it is also very error prone for selecting small targets because, as mentioned previously, the location of the contact point is hard to anticipate. Finally, Direct Touch does not tackle the problem of targets located at the borders of the screen.

2.2 Dragging Techniques

Dragging techniques come from the *take-off* paradigm [11] which consists in pressing the screen, dragging a cursor, and lifting the finger to validate the selection. The former technique, Offset Cursor [11,13], was designed to avoid finger occlusion on large touch-screens and to solve the accuracy problem of Direct Touch. A cursor is always displayed at a fixed distance above the contact point to help the user reaching the topmost locations of the screen. Offset Cursor was shown to induce far fewer errors [11, 13] than Direct Touch, but it is also significantly slower. In [14], Vogel et al. noticed that users often overshoot or undershoot targets. They assumed that it is difficult for the users to estimate the offset distance and that a lengthy adjustment of the cursor, called *net correction distance*, is thus necessary to acquire targets.

Another point is that Offset Cursor does not cover the entire extent of the screen. As the cursor is always located at the same distance from the top of the finger, targets at the bottom of the screen remain unreachable. Moreover, the thickness of screen edges makes it difficult to select targets located near the corners. An adaptative horizontal offset has been proposed in [5] to improve Offset Cursor: this offset is null at the center of the screen and grows smoothly towards the left and right borders. This technique makes it easier to reach items that are close to the left and right borders, but requires slightly more training.

Thumbspace [6] has been designed to improve access to the borders and corners of the screen. It uses an on-demand "radar view" that the user can trigger at the center of the screen. Interacting directly on this radar view allows the user to reach all locations on the screen. Thumbspace thus works as an absolute positioning touchpad superimposed on the standard touch-screen. A drawback of this approach is that the thumb is above the cursor in some areas of the screen, thus causing an occlusion. To get around this issue, the authors proposed to use Thumbspace for targets that are difficult to reach and Direct Touch for near targets.

Thumbspace also relies on *Object Pointing* [3]. The original feature of this interaction technique is that the cursor never visits empty regions and jumps from one target to another, according to the direction of the pointer. Thumbspace uses this strategy with a triggering threshold of 10 pixels to avoid jerky cursor movements. The screen is subdivided into "proxy" areas which are associated to a unique target. This way of "tiling" makes unused background areas active and thus provides more motor space for selecting each target. However, this approach may lose in efficiency when many targets are present on the screen or if they are close to each other.

2.3 Hybrid Techniques

Shift [13] attempts to decrease the selection time of Offset Cursor by a hybrid approach: a coarse Direct Touch on the target can be followed by a precise cursor adjustment if needed. Touching the screen triggers a callout that shows a copy of the occluded area in a non-occluded area. The actual selection point (under the finger) is represented by a cursor in the callout, and the user adjusts its position to fine tune selection before releasing his finger. This technique reduces the *net correction distance* and selection time as the user touches the screen directly on the target. Besides, the callout only appears when needed, after a delay that depends on the target size (the larger the target, the longer the delay). This strategy should improve selection time as the callout is only used for fine-tuning small target selections. However, Shift does not completely solve the screen coverage problem as it requires users to put their fingers close to the target location. Finally, the experiment that was presented in [13] was performed by using both hands to manipulate the device.

2.4 Summary

Direct Touch is the fastest technique proposed so far. However, it remains unusable in most real-life applications because of its high error rate. Some alternatives, inspired by the *take-off* paradigm [11], have been proposed. However, even if they solve the accuracy problem of Direct Touch, the other issues of thumb interaction remain unaddressed. Offset Cursor avoids occlusion and increases accuracy but it limits access to targets at the bottom of the screen and it is not very well suited for reaching targets in the right and left corners (this problem can however be solved by using an adaptive horizontal offset). Thumbspace was specifically designed to address this accessibility problem in the corners, but it does not prevent occlusions in the center of the screen. Finally, Shift which was evaluated by using both hands, does not fully address the corner accessibility issue of the thumb as users must tap close to the desired targets. To sum up, as illustrated in Table 1, efficient solutions have been proposed to solve the problems involved with thumb interaction individually, but none of the existing techniques address them all together. This is the challenge we met by designing TapTap and MagStick, two new interaction techniques that we introduce in the next sections.

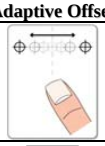
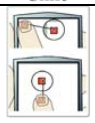
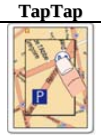
	Direct Touch	Offset Cursor	Adaptive Offset	Thumbspace	Shift	TapTap	MagStick
Overview							
Target Accessibility							
	Grayed areas are difficult to reach – Hatched areas are impossible to reach						
Thumb Occlusion	Everywhere	None	None	Center (if same relative and absolute positions)	On top left	None	None
Pointing Accuracy	Coarse	Medium (net correction distance time)	Medium (net correction distance time)	Fine (facilitated by Object Pointing)	Medium (small targets) and coarse (large targets)	One coarse and one fine (increase target size)	Fine (facilitated by Semantic Pointing)

Table1. Comparison of the features of one-handed interaction techniques

3. TAPTAP AND MAGSTICK

TapTap and MagStick are specifically designed for interacting with the thumb on small touch-screens. Both techniques address the issues of thumb interaction that we previously pointed up. Their respective designs result from a twofold strategy: TapTap was conceived as an improvement of Direct Touch and solves its accuracy and accessibility problems and MagStick is an improvement of Offset Cursor and other techniques based on the take-off principle. A video demonstration can be viewed at http://www.anneroudaut.fr

3.1 TapTap

TapTap comes from a simple idea: if a single tap is not efficient for selecting a small target accurately, a second tap should suffice to disambiguate the selection. More precisely, the first tap defines an area of interest on the screen (Fig. 2a); this area is then magnified and displayed as a popup on the center of the screen (Fig. 2b); the second tap selects the desired target in the popup (Fig. 2c) (or cancels the selection if an empty space is selected).

Selection is by design more precise because the selecting tap takes place on a magnified view of the area of interest where the targets are large enough to be easily selected with the thumb. TapTap also improves accessibility in screen border areas. Not only does the first tap not need to be performed on the desired target (it must only be performed reasonably close to this target), but also the magnified view pops up in the center of the screen. Targets that are close to the borders in the original view thus appear in a location that is much easier to reach in the magnified view.

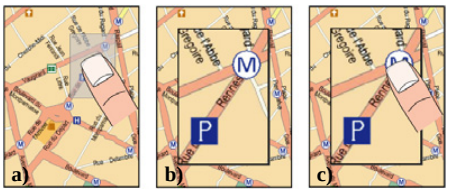


Figure 2. TapTap Design

TapTap is thus based on a temporal multiplexing strategy where the first tap serves to specify the focus area in the original view so that this focus will be displayed at a scale that makes it possible to select the target precisely. Although based on zooming, this strategy has some interesting characteristics that make it different from usual multi-scale approaches. First, there is no interactive control of the zooming factor nor of the amount of XY panning as they are automatically adjusted. Interaction is very fast and works practically like a quasi-mode: the first tap enters the selection mode and makes the zoomed view appear, while the second tap closes this view and leaves the selection mode.

The zooming factor was chosen in order to take into account the size constraints of small touch-screens on mobile devices. Besides, in an attempt to satisfy contradictory constraints, the view and the targets are not zoomed in with the same factor.

On the one hand the focus zone that is selected by the initial tap must be relatively large so that it contains the desired target even if the tap location is (reasonably) far from the target. A size of 80 x 120 pixels was empirically chosen (for a QVGA screen of 240 x 320 pixels). This makes it possible to tap as far as 40 pixels horizontally and 60 pixels vertically from the desired target, a distance that is sufficient to prevent almost all errors in the first tap.

On the other hand, the relatively large size of the focus zone constrains the zooming factor that can be applied in the magnified view because of the small size of the QVGA screen. Moreover, the whole screen real estate can not be used because of the accessibility problem (the areas close to the borders are difficult to reach with the thumb). As a consequence, the focus area is only magnified by a factor of 2 in the pop up (its size is thus 160 x 240 pixels) and placed in the center of the screen (Fig. 2). It is hence located in the most favorable area of the screen for interacting [6].

However, this zooming factor may be insufficient for making common targets large enough to be selected precisely. According to [9] targets should be at least 9,2mm large for making thumb selection easy. But many mobile applications have targets as small as 3 mm [14,12]. In order to ensure sufficient size, targets are zoomed in by a factor of 3 instead of a factor of 2 for the rest of the focus view. Ours observations showed this choice to be effective: users had no difficulties in selecting 9mm targets (i.e. 3mm targets magnified 3 times) and they were not disoriented by this dual zooming factor (in fact none of them noticed this feature).

3.2 MagStick

Dragging techniques are more accurate than Direct Touch but they are significantly slower and do not solve all screen accessibility issue. MagStick solves these problems by providing a telescopic stick that controls a "magnetized" cursor. The telescopic stick can reach any target on the screen while the magnetization of the cursor (which can be seen as form of semantic pointing [8]) speeds up the adjustment of the cursor to the target location. Finally, the offset distance of the cursor is not constant, but dynamically adjusted by the user in a highly predictable way.

MagStick works as follows: 1) when the user presses the screen, he defines a reference point (Fig. 3a); 2) by dragging his thumb he makes a two-part stick appear (Fig. 3b): the two parts emanate from the reference point and end at the current position of the thumb and the location of the cursor; 3) as both parts always have the same length and (initially) the same direction, the user can control the location of the cursor by dragging his thumb continuously on the screen (changing the size of one part of he stick automatically changes the size of its other part); 4) targets *attract* the cursor as if it was "magnetized", with the effect of bending the stick as shown in Fig. 3c); 5) finally the user releases the thumb to select the target that is currently below the cursor (or to cancel if an empty space is selected)

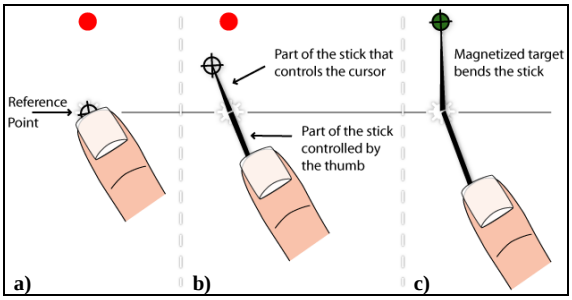


Figure 3. MagStick Design

A key feature of this technique, which was inspired by games such as electronic billiards, is that the cursor moves in the opposite direction of the fingertip. This strategy is especially efficient for avoiding visual occlusions as the thumb must be moved *away* from the desired target: not only the thumb will not hide the target but a large part of its visual context will be made visible.

Another important feature of MagStick is that its symmetrical design allows the user to easily predict the movement to perform. An important drawback of Offset Cursor is that most users, with the exception of very well-trained ones, can not know the exact location of the cursor until they touch the screen. They must thus wait for the cursor to appear before starting to adjust its position finely. Conversely, as the two parts of the stick are of equal length, this problem does not exist with MagStick. The user can predict how far he will have to move his finger *before* touching the screen as this distance is equal to the distance between the target and the reference point.

Magnetization, which derives from Semantic Pointing [2], also contributes to speed up the selection task. Each target has a proximity area that attracts the cursor and "bends" the stick. When the cursor enters a proximity area, it is attracted to the center of the corresponding target. This feature makes fine positioning unnecessary but also avoids "empty selection" errors that would otherwise occur when the user overshoots or undershoots the desired target. Conversely, when the user moves the stick (and the cursor) away from a target area, the magnification effect vanishes and the two parts of the stick become aligned again until the cursor is attracted by another target. A possible refinement would be to assign different attraction powers to targets, as proposed in the original Semantic Pointing technique. It could facilitate the selection of targets that are very frequently used, or, conversely, to prevent the accidental activation of dangerous commands. However, this feature should be carefully tested in the context of thumb interaction where cursor movements are necessarily more imprecise than when using a mouse on a desktop.

4. PROPERTIES OF THE TECHNIQUES

This section compares the properties and the respective advantages of our techniques. In particular, it shows that they provide efficient solutions to the three problems presented in the 'related work' section: target accessibility, visual occlusion and accuracy. We also investigate the compatibility of our techniques with dragging gestures and other target sizes and layouts.

4.1 Target accessibility

TapTap and MagStick can select targets anywhere on the screen although they use different principles. TapTap uses a two-step zooming strategy where the user specifies a focus of interest that is then displayed at a larger scale in the center of the screen. The first tap does not need to be very close to the target and the second tap is always performed in the most favorable area of the screen.

Conversely, MagStick relies on a space-shifting strategy by providing a "telescopic arm" that reaches targets close to the borders. As with TapTap, MagStick makes it possible to perform the dragging gesture in the most favorable area of the screen, but it leaves freedom to the user to interact by following two different strategies. The first one consists in touching the screen very close to the target in order to minimize the length of the dragging gesture. Another strategy is to systematically start the dragging gesture from the center of the screen. Any target can then be selected, either by placing the thumb below the target if it is in the upper part of the screen, or above the target if it is in its lower part. This strategy was in fact used by most of our participants during the evaluations. Another of its advantages is that it allows the user to hold the mobile device firmly with the hand that performs the interaction. The thumb joint is then located in the middle of the right border of the screen (for a right hand user) and the center of gravity of the handheld device is roughly above the center of the hand. This position is safe and convenient because it prevents the risk of dropping the device accidentally. The user then moves his thumb upward or downwards when the target is exactly located beneath the natural position of the thumb joint, but this case seldom occurs and does not require cumbersome hand movements.

4.2 Visual occlusion

The zooming strategy of TapTap prevents visual occlusion by design: as targets are magnified by a x3 ratio, they are large enough not to be completely hidden by the thumb.

The design of MagStick also ensures that visual occlusion can not occur as the thumb moves away from the desired target. Both the target and the focus of attention are clearly visible. It also prevents occlusion in the thumb joint area as shown in Fig. 4 for the same reason as explained in the previous section: the thumb is naturally located in the middle of the screen and can easily be slightly shifted up or down when needed.

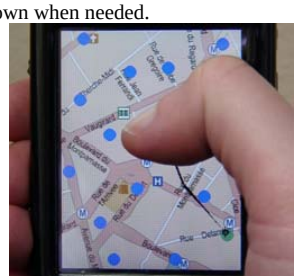


Figure 4. No occlusion on the thumb joint with MagStick

4.3 Accuracy without reducing speed

Both techniques attempt to "circumvent" the constraints of the Fitts' Law for a homogeneous 2D space in different manners. TapTap relies on a multi-scale space (that can be seen as a generalization of magnification tools as those proposed in [8,15]). As shown by Guiard et al. [4] multi-scale spaces significantly increase the range of indexes of difficulty that users can handle and Fitts' Law applies uniformly over this range. TapTap makes it possible to decrease the index of difficulty through zooming (that is to say a translation on the scale axis of the space-scale diagram). The two taps required by TapTap are thus performed faster than two "standard" successive taps (this assumption was confirmed by experimental data): The size of the "target" is increased, and the distance between the thumb and the target decreased in both steps of the interaction (the "target" being a zone of interest in the first case, and an actual but magnified and centered target in the second case). This property also increases accuracy and allows the user to view TapTap as a double click with a fast spatial readjustment between the two taps. As detailed in the experiment section, this effect was striking when conducting the evaluation: users did not give the impression that they were performing two successive taps but rather a compound gesture.

Similarly, MagStick relies on Semantic Pointing, a technique that distorts the motor space and thus artificially reduces the pointing distance. This technique also avoids the cursor leaving the target when the thumb is slightly, and involuntarily, moved. As stated above, the input signal provided by current touch-screen technology is somewhat imprecise and instable when interacting with the thumb. Although filtered by a low pass filter to remove outliers and smooth the input curve [13,14] this signal is still far from being perfectly reliable. Besides, the user may also involuntarily move his thumb when he releases it and thus miss the target. Magnetization solves both problems.

Finally, the ability to predict the movement before starting the gesture is probably another key feature for making the selection faster. The property relies on the fact that both parts of the stick always have the same length. Using a variable gain, as in [1], sounds appealing but could decrease performance in our case because this important property would be lost. This was confirmed by preliminary experiments we made when designing MagStick.

4.4 Other properties

Real mobile computer operations are combination of different interaction techniques, such as pointing or dragging. In our experiments, we focus on pointing with small and randomly laid out targets. In this section, we present some other interesting properties of TapTap and MagStick. More precisely, we investigate how our techniques work with different kinds of targets (size and layout) and their compatibility with other interaction styles.

Large targets. Although targets can hardly be much smaller, and still easily visible, than those we considered (3mm, a size found in many mobile applications [14,12]), they can however be much larger. MagStick then operates as Direct Touch: as the cursor appears below the thumb when it is pressed on the screen, the user can just release it without performing any movement to select the target. TapTap can be replaced by Direct Touch for large targets. This can be made explicit by a visual cue. But choosing target sizes in a consistent way may suffice (for instance targets with only 2 or 3 different heights). Selecting targets in two different ways may not be a real problem after some training: a) people do that all the time when using desktops (documents must be double-clicked, while other buttons are, generally, simple-clicked); b) a small inactivation delay could be used in such a way that a second click on a large button (or the view it generates) would have no effect. Hence, a useless second tap would never produce an unexpected result.

Lists and Groups. Aligned or grouped targets are often common in real applications: this case typically occurs in menus, lists, tool boxes, tabbed panes, etc. While TapTap performance is likely to be similarly high whatever the layout, the specific design of MagStick can provide interesting features in this case. It makes it for instance possible to access items organized as lists or trees by moving the thumb away and keeping it approximately at the same location of the screen. This could be very useful for browsing a menu system without having to perform multiple target selections. Besides, as the thumb can be placed rather far away from the target, this would noticeably reduce occlusion and would thus make it possible to display more contextual information.

Dragging gestures. TapTap does not interfere with interaction styles based on dragging gestures as it only requires users to tap the screen. A target can be moved by dragging the thumb on the screen instead of releasing it immediately after the second tap (the popup does not cause visual occlusion because it disappears when the user starts the second tap by pressing the screen). This way of dragging objects is in fact quite similar to the usual one except that the target is not beneath the cursor but remotely controlled by the movements of the thumb. The target moves in the global view according to the movement of the thumb from the position of the second tap. In order to move the target anywhere on the screen, this movement is multiplied by a constant gain of 2. In addition, TapTap also makes it possible to pan the entire view by dragging on its "background". An image, a map or a page could for instance be panned in this way.

MagStick also has interesting properties regarding this criterion. First, it allows an object to be dragged, although in a slightly less usual way than with TapTap. Instead of releasing the thumb immediately when the proper target is reached, the user must wait for a small temporal delay. The target is then implicitly selected and can be moved by dragging the finger.

To sum up, we have seen in this section that TapTap and MagStick address all the issues raise by one-handed interaction, and that they can be applied in different kinds of application without preventing the use of other interaction styles. The next section shows the effectiveness of TapTap and MagStick through a controlled experiment that compares them with the main techniques proposed so far.

5. EXPERIMENTAL EVALUATION

We conducted a controlled experiment to compare TapTap and MagStick with the main techniques published before: Direct Touch, Offset Cursor [11], Thumbspace [6] and Shift [14]. Since the previous techniques principally explored the pointing task, our experimentation focuses on this problem of pointing only. According to the design of our techniques and the properties that were previously described, our hypotheses are that:

***H1**: TapTap and MagStick are the fastest techniques after Direct Touch.*

***H2**: TapTap and MagStick are the techniques with the lowest error rate*

***H3**: TapTap and MagStick are efficient for accessing targets anywhere on the screen.*

5.1 Task

The task consisted in performing series of target selections with the six techniques. Participants were asked to hold the device with their dominant hand and to use their thumb. Several targets were displayed on the screen and one of them was to be selected. The participants were instructed to perform the selection as quickly and accurately as possible. Before each trial, the user presses a "Next trial" button and a city map appears with a set of 16 targets. They are displayed in blue color, except for the one to be selected that is in red. The blue targets are distractors in order to improve the realism of the target acquisition task. The color of the target changes to green when the cursor flies over it (except for tapping techniques such as Direct Touch and TapTap). The trial ends when the user lifts his thumb from the screen, whether he succeeds or not the selection. A sound indicates the result of the acquisition.

5.2 Apparatus and participants

The techniques have been implemented in C# (with the .Net Compact Framework) and operate on the Windows Mobile 5.0 OS. Experiments have been performed on a HTC P3600 PDA-phone with a QVGA (320x240) touch-screen. Twelve volunteers (1 female), ranging in age from 23 to 47 years, were recruited from our institution and received a handful of candies for their participation. All of them were using a mobile device with a touch-screen for the first time. Two subjects were left-handed and we mirrored their results so that each user used their dominant hand to perform the experimentation.

5.3 Experimental conditions

The efficiency of the interaction techniques involved in this experimentation is likely to depend on the location of the target. Karlson et al. took this aspect into account in their experiment [6]. They subdivided the screen into 12 areas arranged as a regular matrix. We used a different subdivision pattern, with 8 zones of the same surface area (Fig. 5a represents the 12 areas for a right-handed person), which provides a clear separation between the areas located at the center of the screen and those close to the borders, which may degrade performance. This analysis of the screen areas is important because it can have strong implications on the design of interactive applications.

To reduce the task time for our participants, we only considered one target size of 3 mm, because this value was reported to be the actual minimal widget size in mobile applications [14,12]. Besides, Vogel also reported in [14] that Direct Touch and Shift outperforms other techniques for targets larger than 18 pixels. The study thus focuses on small targets, as they constitute a more difficult case and are commonly found in mobile applications. The proximity areas for the MagStick magnetize effect measure 10.8mm.

A minor enhancement was made on Offset Cursor because its original design makes it impossible to reach targets on the bottom of the screen. So that this technique is not at disadvantage, the user can make the cursor appear below the thumb position (negative offset mode) by pressing a hardware button before touching the screen. The analysis of the experimental data confirmed that this improvement did not affect the results (the performance is not significantly different in the 'down' area than in "easy to reach" areas such as 'up' and 'Center'). Hence all targets can be selected by using any of the 6 tested techniques.

During the task, Time, errors and thumb movements were recorded. At the end, the subjects answered a questionnaire to give their opinion and satisfaction about all techniques (6 variables were measured on a 5 pt. Likert scale).

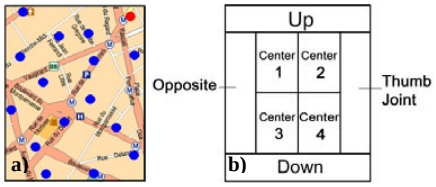


Figure 5. a) Targets layout b) Target Area subdivision

5.4 Experimental design

A repeated measures within-subject design was conducted. The independent variables are *Techniques* (Direct Touch, Offset Cursor, Thumbspace, Shift, TapTap and MagStick) and *Target Area* (8 areas shown in Fig. 4a). The presentation of *Technique* was circularly counterbalanced among participants. All of them performed 16 selections twice in all the 8 *Target Areas*. *Target Areas* were ordered in a sequence circularly counterbalanced for each technique. This sequence aims at balancing the regions that are easy or hard to reach. Finally, at the beginning of each technique, subjects performed 10 practicing trials. In summary, the design was: 6 *Techniques* x 8 *Target Areas* x 2 *blocks* = 96 selections (15-20 minutes) per participant.

5.5 Results

Repeated measures analysis of variance showed that the order of presentation of the techniques had no significant effect on selection time or error rate.

5.5.1 Selection time

Task time was measured from the moment the user released the "Next trial" button to the moment his thumb was lifted up from the screen. Trials with selection errors were excluded from the selection time analysis. We performed a 6 x 8 (*Technique*, *Target Area*) within subject analysis of variance. We found significant main effects for *Technique* ($F_{5,55}$=14.59, p<.001) and *Technique* x *Target Area* interaction ($F_{35,268}$=2.31, p<.001). Post hoc multiple means comparison tests allowed us to rank the techniques as follows: Direct Touch (1177.8 ms) and TapTap (1547.4 ms) (no significant results between them), MagStick (2037.6 ms), Shift (3046 ms) and Offset Cursor (3562.7 ms) (no significant results between them), and the slowest, Thumbspace (3897.3 ms). The results show that: TapTap is about to 2.3 times faster than Offset Cursor, 2 times faster than Shift, and almost 2.5 times faster than Thumbspace; MagStick is about 1.7 times faster than Offset Cursor, 1.5 faster than Shift and 1.9 faster than Thumbspace. These results, illustrated in Fig. 6, are all significant. We found that Direct Touch was the fastest, but (as described in the error result) the quantity of data collected is small compared to the other techniques. We can considerate Direct Touch "out of range" because a technique that produces so many errors is of course very frustrating for users, and can not be compared in this experiment with the other techniques that all provide better results.

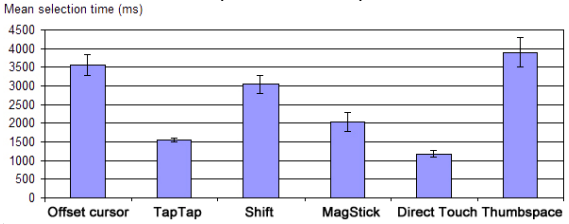


Figure 6. Mean time (ms) for Technique
Bars represent 95% confidence interval.

The analysis of the *Technique* x *Target Area* interaction showed that *Target Area* has no significant effect on selection with TapTap and MagStick. There is a significant effect for Offset Cursor, which is less efficient in the 'joint' area (see Fig.4) (2246.2 ms mean difference) and in the 'opposite' area (1250 ms mean difference) than in other zones. A similar effect was found for Shift in the 'up' (2445.6 ms mean difference) and 'opposite' (936.8 ms mean difference) areas. These results confirm our observations during the experimentation sessions where we noticed that users often hide the target with their thumb in these two areas. Some other significant effects were also found with Thumbspace, which performed better on the borders of the screen than in the center area (1381.9 ms of difference on average). This result corroborates the assumptions of the authors [6].

In summary, without considering Direct Touch, TapTap is the fastest and MagStick the second. The border areas are reached faster with MagStick than with the hybrid and the dragging techniques. Not only is MagStick quite efficient for reaching the edges, but it also does not impair interaction in the center of the screen (as Thumbspace does). TapTap is particularly fast and consistent across screen areas.

5.5.2 *Error rate*

The error rate measurement aggregates both empty and wrong target selections. We performed a 6 x 8 (*Technique*, *Target Area*) within subject analysis of variance on the aggregated number of errors. Error rate was significantly affected by *Technique* ($F_{5,55}$=45.91, p <.001) and *Technique* x *Target Area* interaction ($F_{35,268}$ = 1.74, p<.001). Post hoc multiple means comparison tests showed that TapTap (6.7%) has the lowest error rate and Direct Touch (59.9%) the highest in comparison to all other techniques (Fig. 6). No significant results were found in comparing the other techniques (i.e. Offset Cursor (16.1%), Shift (17.1), MagStick (10.4%) and Thumbspace (18.7%)). We can notice that the error rate of Direct Touch is considerably high. The error rate of TapTap is about 2.5 (and 1.6 for MagStick) times smaller than for Offset Cursor, Shift and Thumbspace.

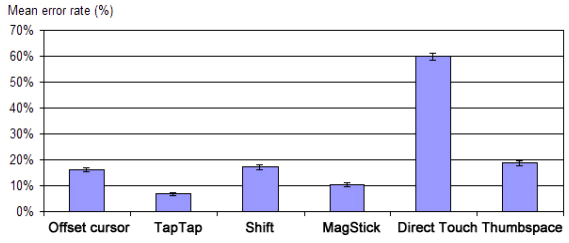


Figure 7. Mean Error rate for *Technique*
Bars represent 95% confidence interval.

The only significant result about *Technique* x *Target Area* interaction is mainly due to Direct Touch. Considering its high error rate and dissatisfaction of our participants with it, we will not discuss on these results. By considering empty and wrong selections separately (they were previously merged), we found that Thumbspace only produces wrong selections while the other techniques induce mostly empty selections. In fact these results are not surprising because by design Thumbspace "tiles" the space. This approach, which could be efficient because the target is then larger in the motor space, have also the disadvantage of causing more wrong errors that are much more costly than empty selections (canceling an action triggered by a wrong selection may be time-consuming and frustrating).

In summary, TapTap has the lowest error rate and Direct Touch the highest. All the "dragging" techniques and Shift have approximately the same error rates, except that Thumbspace errors are only wrong selections.

5.5.3 *Subjective preferences*

With the post-study questionnaire, participants ranked the six techniques as follows: TapTap, MagStick, Shift, Offset Cursor, Thumbspace and the most disliked Direct Touch. Their opinions about the *speed*, *accuracy*, *pleasantness*, *simplicity*, *learning* and *fun* are illustrated in Fig. 8. TapTap is the most liked technique for all criterions, except for 'fun' where it is placed second. Tapping approaches (TapTap and Direct Touch) are ranked first for the 'speed' assessment and users estimated that TapTap performs faster than Direct Touch, even if quantitative results showed the contrary. Direct Touch is disliked for the 'accuracy', 'pleasantness' and 'fun' criteria. Results for dragging approaches have a similar shape, with MagStick and Offset Cursor generally above Shift and Thumbspace. MagStick is judged slightly inferior for 'learnability' but ranked first for 'fun'.

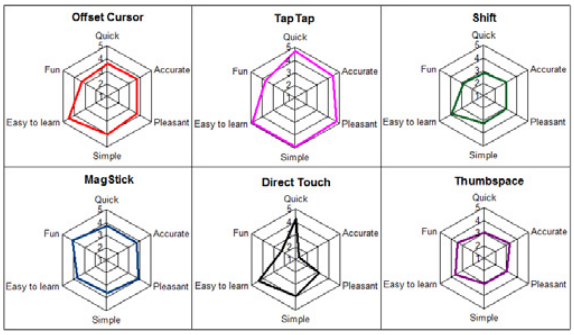


Figure 8. Questionnaire results (means).

5.6 Discussion

The results of our experimentation confirm our hypotheses. TapTap has the lowest error rate (H2) of all techniques and it is the fastest technique after Direct Touch (H1). In fact, it would be even faster than Direct Touch in the case of real usage. As Direct Touch is very error prone, many selections will have to be performed again. The average time needed to select a target is thus significantly higher than the time to correct selections given in the previous section. This average time can be estimated by considering that the selection task will take at least twice as much time in the case of wrong selections as the target must then be selected again (in fact it will take more time because a wrong selection may launch an undesired application that the user will have to close). According to this hypothesis, Direct Touch would require an estimated average time of 2002 ms while TapTap would only need 1676 ms as it produces much fewer errors.

Another interesting point is that the single tap of Direct Touch takes more time (1177.8 ms) than each tap of TapTap (803.3 ms for the first tap and 744.1 ms for the second tap). These results confirm the validity of the design hypotheses presented in section 3. Besides, users seem to perform the second tap slightly faster, an effect that may come from the fact that the magnified area is centred and the target thus pretty close to the natural position of the thumb.

Our results also validate the hypotheses that MagStick is faster (H1) than other techniques (TapTap and Direct Touch except, but Direct Touch is too error prone to be really usable, as stated before) and that it produces fewer errors (H2) than other

techniques (TapTap except again). Another interesting observation is that the time to press the screen is slightly faster for MagStick (844.2 ms) than for other dragging techniques (1140,6 ms for Offset Cursor, 958,7 ms for Shift and 935,8 ms for Thumbspace). This may be explained by the fact that users tend to place their thumb systematically in the centre of the screen without spending time to adjust the position of the thumb. Once they touch the screen (approximately) at its center they then move the thumb for the same distance as the distance between the center and the target. As the execution time of MagStick is also faster than for other dragging techniques, we hypothesize that the users make an estimation of this distance before touching the screen (only one participant among the twelve has made errors due to a wrong positioning of the thumb).

The rapidity of MagStick may also be explained by Semantic pointing. However this mechanism depends on target density, and should be carefully tested in this context. Our first experiments with a high target density (32 instead of 16 targets of 3mm randomly displayed), shows that MagStick performance is then equivalent to those of Shift and Offset Cursor (while TapTap efficiency is almost the same for both densities). To increase the performance of MagStick, we plan to implement a density-dependent approach that dynamically adapts the strength of the magnetizing effect according to the position of the cursor and its local context on the screen.

Our experimentation also shows that the selection time and the number of errors do not depend on screen areas when using TapTap and MagStick (H3). Conversely, Thumbspace is less efficient in central areas (as also demonstrated in [6]), Shift impairs interaction in the top and left corners because of visual occlusion, and Offset Cursor degrades performance in all screen corners. TapTap and MagStick both provide efficient solutions to these issues as they help users to reach any target in a short and constant time, whatever its location on the screen. MagStick performed well in border areas without decreasing efficiency in the center. Finally, MagStick tends to concentrate most thumb movements in the center as shown in Fig. 9. It also provides a comfortable grip for user interaction in mobility conditions and it is well-adapted to thumb morphologic capabilities.

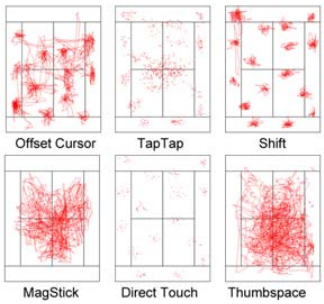


Figure 9. Thumb gesture traces.

6. CONCLUSION AND FUTURE WORK

We have presented TapTap and MagStick, two new interaction techniques that improve target acquisition on small touch-screens for mobile devices. TapTap is based on time-multiplexing through an automatic two-step zooming strategy. MagStick relies on magnetization, a variant of semantic zooming and also makes it possible to predict thumb movements and thus to reduce the net correction distance. Our experiments showed that both techniques are faster and produce fewer errors than the current state of the art. They also cover the other issues raised by thumb interaction on small touch-screens such as visual occlusion and target accessibility in all parts of the screen. They are also both compatible with interaction techniques relying on dragging gestures. Finally, this paper also offers a significant benefit by presenting a thorough analysis of the techniques published so far.

In future work, we plan to adapt our techniques to constraints that depend on the application context (higher target densities, specific target layouts such as lists or trees…) and to perform further evaluations to evaluate their efficiency under these conditions.

7. ACKNOWLEDGMENTS

This work has been done in collaboration with Bruno Aidant, Bruno Legat and Johann Daigremont from Alcatel-Lucent that we thank for their useful advices. We also thank Yves Guiard for his precious help for statistical analysis and all the participants for their pleasant contributions.

8. REFERENCES

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

2 Blanch, R., Guiard, Y., Beaudouin-Lafon, M. Semantic pointing: improving target acquisition with control-display ratio adaptation. *Proc. CHI'04*. 519-526. 2004.

3 Guiard, Y., Blanch, R., Beaudouin-Lafon, M. Object pointing: a complement to bitmap pointing in GUIs. *Proc. Graphics interface 2004*.Vol. 62. 9-16. 2004.

4 Guiard, Y., Beaudouin-Lafon, M. (2004). Target Acquisition in Multi-Scale Electronic Worlds. International Journal of Human-Computer Studies, 61, 875-905.

5 Huot, S., Lecolinet, E. Focus+Context Visualization Techniques for Displaying Large Lists with Multiple Points of Interest on Small Tactile Screens. *Proc. Interact'07*.

6 Karlson, A., Bederson, B. ThumbSpace: Generalized One-Handed Input for Touchscreen-Based Mobile Devices. *Proc. Interact'07*.324-338.2007.

7 Karlson, A., Bederson, B., Contreras-Vidal, J. Understanding on User Interface Design and Evaluation for Mobile Technology, Idea Group, 2007.

8 Mankoff, J., Hudson, S. E., and Abowd, G. Interaction techniques for ambiguity resolution in recognition-based interfaces. *Proc. UIST'00*. 11-20. 2000.

9 Parhi, P., Karlson, A., Bederson, B. Target Size Study for One-Handed Thumb Use on Small Touchscreen Devices. *Proc. MobileHCI'06*. 203-210. 2006.

10 Pascoe, J., Ryan, N., Morse, D. Using while moving: HCI issues in fieldwork environments. *ACM Trans. Comput.-Hum. Interact.* 7(3):417-437. 2000.

11 Potter, R., Weldon, L., Shneiderman, B. Improving the Accuracy of Touchscreens: An Experimental Evaluation of Three Strategies. *Proc. CHI'88*. 27-32. 1988.

12 Ren, X. and Moriya, S. Improving selection performance on pen-based systems: a study of pen-based interaction for selection tasks. *ACM TOCHI*. 7(3).384-416. 2000.

13 Sears, A., Shneiderman, B. High precision touchscreens: design strategies and comparisons with a mouse. *Int. J. Man-Mach. Stud.* 34(4):593-613. 1991.

14 Vogel, D. and Baudisch, P. 2007. Shift: a technique for operating pen-based interfaces using touch. *Proc. CHI'07*. 657-666. 2007.

15 Grossman, T., Balakrishnan, R. The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area. *Proc.CHI'05*. 281-290. 2005.

DOI : 10.1145/1629826.1629845

FlowStates: Prototypage d'applications interactives avec des flots de données et des machines à états

Caroline Appert[1,2] *Stéphane Huot*[1,2] *Pierre Dragicevic*[2] *Michel Beaudouin-Lafon*[1,2]

[1]LRI - Univ. Paris-Sud & CNRS
Bât. 490, F-91405 Orsay, France

[2]INRIA
Bât. 490, F-91405 Orsay, France

{appert,huot,dragice,mbl}@lri.fr

RESUME

Cet article présente FLOWSTATES, une boîte à outils compatible avec Java Swing qui associe deux modèles de gestion des événements : les flots de données et les machines à états. Le modèle à flot de données permet de facilement interfacer des périphériques non standard et de reconfigurer les interactions en fonction des périphériques disponibles, tandis que les machines à états permettent de réaliser aisément des interactions complexes. À travers des exemples, l'article illustre la puissance et l'expressivité de cette approche hybride et la flexibilité qui résulte du choix explicite de ne pas fixer les limites entre les rôles de chaque modèle.

MOTS CLES : Boîte à outils, flot de données, machine à états

ABSTRACT

This article introduces FLOWSTATES, a user interface toolkit compatible with Java Swing that combines two models for managing events : dataflow and state machines. The dataflow model makes it easy to support non-standard input devices and to reconfigure interactions according to the available devices, while state machines support the programming of complex interactions. The article illustrates the power and expressivity of this hybrid approach and the flexibility afforded by the explicit decision to not set strict limits between the roles of each model.

CATEGORIES AND SUBJECT DESCRIPTORS: H5.m. Information interfaces and presentation (e.g., HCI) : Miscellaneous.

GENERAL TERMS: Languages, Design.

KEYWORDS: User interface toolkit, dataflow, state machine

INTRODUCTION

La gestion des entrées dans les interfaces graphiques est un problème notoirement complexe. La plupart des boîtes à outils d'interface actuelles utilisent le modèle des fonctions de rappel ("callbacks") qui sont appelées lorsqu'un événement se produit. Ce modèle est parfois mis en oeuvre sous la forme d'une délégation dans les langages à objets, comme par exemple avec les "listeners" de Java. Un autre modèle qui a été largement exploré est celui des machines à états, fondées sur des automates à états finis dont le langage d'entrée est le vocabulaire de types d'événements disponibles. Cette approche a par exemple été récemment intégrée à la boîte à outils Java Swing avec l'extension SWINGSTATES [3]. D'autres modèles utilisent des automates plus élaborés comme les machines à états hiérarchiques [6] ou les réseaux de Petri [28]. Un troisième modèle considère les événements comme des signaux qui se propagent selon un flot de données, paradigme issu des langages et modèles réactifs. La boîte à outils ICON [14] en est un bon exemple.

Chacun de ces modèles comporte des avantages et des inconvénients selon les types d'événements qui sont traités, selon le type de techniques d'interaction à implémenter, selon le niveau d'abstraction des événements manipulés, selon le degré de dynamicité souhaité, etc. Jusqu'à présent, à de rares exceptions près [22, 28, 13], la plupart des boîtes à outils ont utilisé un seul de ces modèles, obligeant le programmeur à s'accomoder de ses limitations.

Cet article présente FLOWSTATES, une boîte à outil qui combine ICON et SWINGSTATES et associe ainsi flot de données, machines à états et graphe de scène. Les événements de bas niveau sont d'abord traités par ICON, ce qui permet de facilement interfacer des périphériques non standard et de reconfigurer les interactions en fonction des périphériques disponibles. Les modules ICON qui s'interfacent normalement avec l'application [14] ou un graphe de scène [21] sont connectés à SWINGSTATES : les signaux d'entrée de ces modules sont transformés en événements abstraits traités par les machines à états de SWINGSTATES. Ceci permet de facilement réaliser des interactions complexes, qui sont souvent décrites dans la

littérature par des automates. Les machines à états agissent à leur tour sur le graphe de scène et le noyau applicatif via les actions associées aux transitions.

Cette organisation en trois couches (*dispositifs d'entrée*, *logique d'interaction* et *actions sur les objets de l'application*) résulte de notre expérience de ces différents modèles et de leur adéquation respective à différents niveaux d'abstraction lors du traitement des événements. Les différents styles de spécification (langage visuel pour ICON et langage impératif pour SWINGSTATES) reflètent également des besoins différents : la gestion des périphériques d'entrée est hautement variable alors que la logique de l'interaction (également appelée dialogue) est plus figée et plus fortement couplée au noyau applicatif [1]. Cependant les limites entre les modèles ne sont pas fixées, de telle sorte que le programmeur garde un contrôle total sur l'utilisation qu'il fait de chaque modèle. En outre, les machines à états peuvent réinjecter des événements vers le flot de données et être déconnectées du noyau applicatif, ce qui permet de s'affranchir des limites du modèle à couches et de combiner les techniques d'interaction à la manière du modèle de l'interaction instrumentale [5].

Dans la suite de cet article, nous passons en revue les différents types de boîtes à outils d'interface. Nous présentons ensuite FLOWSTATES et donnons un aperçu de son pouvoir d'expression à travers des exemples de prototypage d'interactions avancées. Enfin, nous discutons et mettons en évidence les différences entre notre approche et les précédents travaux d'intégration de modèles de machines à états et de flots de données.

APPROCHES POUR PROGRAMMER L'INTERACTION

De nombreux travaux de recherche ont eu pour objet de rendre la programmation de l'interaction avancée possible et plus facile. Nous donnons ici un aperçu des systèmes à écouteurs d'événements classiques et de leurs évolutions, des systèmes à états et des systèmes à flots de données.

L'approche basée sur les écouteurs d'événements

C'est l'approche que le programmeur doit suivre avec les boîtes à outils largement utilisées comme Java Swing ou Gtk. La construction d'une interface avec ces boîtes à outils se résume à un assemblage de composants graphiques ("widgets") dont les liens internes et les liens avec les périphériques d'entrée se font à l'aide d'écouteurs d'événements. Ce type d'application interactive est connu pour être difficile à modifier et à maintenir [25]. Cet état de fait a motivé le développement de boîtes à outils expérimentales avec une gestion des entrées plus flexible. Deux contributions importantes dans ce domaine sont *subArctic* [20] et *Garnet/Amulet* [26, 27]. La force de subArctic est de rendre plus transparente la politique d'aiguillage des événements AWT et de permettre la redéfinition de celle-ci par le programmeur. Quant aux boîtes à outils Garnet/Amulet, elles introduisent le concept d'*interacteur*, un objet qui intercepte les événements et les transforme en opérations sur un objet graphique. Garnet/Amulet propose 6 interacteurs prédéfinis qui reposent sur des machines à états très simples de type "press-drag-relase" mais ne fournit pas de support permettant la programmation d'autres types d'interacteurs. Dans ces deux exemples, le vocabulaire d'interaction n'est pas vraiment enrichi et les moyens de l'enrichir ne sont pas transparents pour le programmeur d'interfaces.

L'approche dirigée par les états

Contrairement au modèle d'écouteurs dans lequel chaque type d'événement doit spécifier sa fonction de rappel, certains modèles sont centrés autour des états du système interactif et ne considèrent que les événements ayant une sémantique dans ces états. Le formalisme des machines à états, basé sur les automates à états finis, a fait ses preuves quant à sa capacité à décrire intelligiblement des interactions complexes "sur le papier" [8, 19]. La boîte à outils SWINGSTATES intègre cette structure de contrôle directement dans le langage Java, afin de faciliter son adoption par les programmeurs d'applications interactives.

Les deux principales limites des machines à états sont l'explosion potentielle du nombre d'états et une gestion limitée du parallélisme. Certains travaux ont donc considéré des formalismes plus complexes comme la boîte à outils hsmTk [6] qui propose des machines à états hiérarchiques dans le langage C++ ou encore l'environnement PetShop [28] qui permet de spécifier la logique de l'interaction par des réseaux de Petri. Mais si la puissance de ces formalismes a été démontrée pour modéliser des interactions complexes, les machines à états simples comme celles de SWINGSTATES ont l'avantage d'être rapides à maîtriser. Notre expérience avec des étudiants de Master a d'ailleurs montré que des développeurs non aguerris étaient capables d'utiliser ce formalisme pour décrire une grande variété de techniques publiées dans la littérature en IHM ces dernières années [3]. En outre, le parallélisme peut être décrit en exécutant simultanément plusieurs machines à états et/ou en les faisant communiquer [3].

L'approche à flot de données

Dans les systèmes à flot de données, chaque changement de valeur déclenche immédiatement le recalcul des valeurs qui en dépendent. Ils sont couramment utilisés dans les domaines du traitement d'images, de vidéos et du son [11, 24], ainsi que pour le prototypage d'interfaces 3D [32, 12]. Auparavant confinée à ces domaines, cette approche est de plus en plus appliquée au prototypage d'interfaces homme-machines avancées [14, 4, 23]. L'un de ces systèmes, ICON, a été couplé avec un modèle de graphe de scène [21], permettant ainsi de décrire non plus seulement des traitements de données mais aussi des techniques d'interaction comportant du graphisme riche. La boîte à outils Qt propose également une extension au langage C++ pour proposer une gestion des écouteurs d'événements selon une logique flot de données. Un objet source peut émettre des signaux alors qu'un objet cible

peut déclarer des slots d'entrée avec des fonctions de rappel qui seront exécutées chaque fois que le slot d'entrée correspondant recevra un signal. La connection entre signal et slot doit être déclarée à l'objet application.

Les flots de données ont plusieurs avantages : ils peuvent être hautement modulaires (les calculs peuvent être encapsulés dans des briques indépendantes), ils permettent de décrire des traitements séquentiels et parallèles avec la même facilité, et ils se prêtent bien à la programmation visuelle. Le principe d'ICON est d'exploiter cette flexibilité pour rendre les applications interactives plus facilement configurables et personnalisables en fonction des périphériques d'entrée disponibles. Par exemple, une application de dessin peut facilement être livrée avec une configuration standard de type souris-clavier, une configuration exploitant le canal de pression des tablettes graphiques, et une configuration d'accessibilité reposant sur la reconnaissance vocale. Ces configurations peuvent ensuite être personnalisées par des utilisateurs experts en fonction de leurs besoins et préférences personnelles, ou dans le but d'exploiter au mieux leur configuration matérielle (par exemple, exploiter l'orientation du stylet fournie par certaines tablettes graphiques). Les configurations d'entrée d'ICON peuvent être vues comme des spécifications InTml [15]. Toutefois, elles sont éditables et exécutables par le moteur réactif d'ICON alors que InTml reste une spécification XML purement descriptive des composants d'une interface de réalité virtuelle.

Malgré sa flexibilité, l'approche à flots de données a un inconvénient majeur : les comportements décrits avec ce paradigme peuvent devenir visuellement très complexes, en particulier si les interactions comportent beaucoup de contrôle (traitements conditionnels et modes). C'est pourquoi FLOWSTATES adopte une *approche hybride* qui couple flot de données et machines à états. Les autres solutions hybrides existantes seront discutées et comparées à FLOWSTATES à la fin de cet article.

SWINGSTATES ET ICON : RAPPELS

Les articles [3] et [14] décrivent respectivement les boîtes à outils SWINGSTATES et ICON en détail. Nous ne rappelons ici que les éléments nécessaires à cet article.

SwingStates

Une des particularités de SWINGSTATES est l'introduction des machines à états comme structure de contrôle pour programmer l'interaction en Java. La figure 1 met en parallèle la représentation graphique d'une machine à états permettant de faire du déplacement de formes graphiques par "drag-and-drop" et son code SWINGSTATES. La machine à états est une classe Java dont les champs sont les états de la machine (en gras). Ces états sont aussi des classes, et leurs champs sont les transitions sortantes de l'état (en italique). Les transitions sont à leur tour des classes.

```
1 StateMachine sm = new StateMachine() {
2   SMShape dragged = null;
3
4   public State start = new State() {
5     Transition dragOn =
6     new PressOnShape(BUTTON1, ">> drag") {
7           public void action() { dragged = getShape(); }
8     };
9   };
10
11  public State drag = new State() {
12    Transition drag =
13    new Drag(BUTTON1) {
14      public void action() { move(dragged); }
15    };
16    Transition dragOff =
17    new Release(BUTTON1, ">> start") {
18    };
19  };
20 };
```

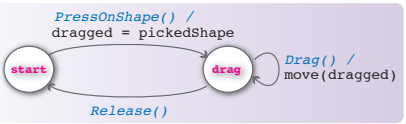


FIGURE *1* : Illustration de la syntaxe de SWINGSTATES pour programmer une machine à états.

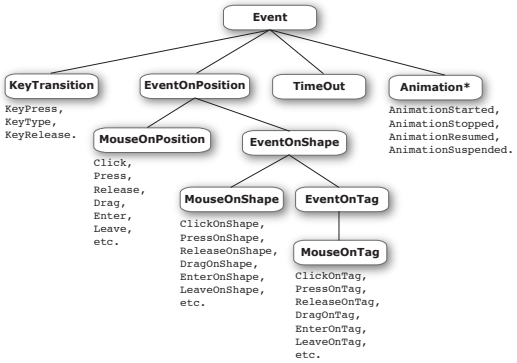


FIGURE *2* : Les classes de transitions disponibles dans SWINGSTATES.

SWINGSTATES propose un ensemble de classes de transitions qui sont déclenchées par différents types d'événements. Par exemple, la transition *Drag* (Fig.1, lignes 12-13) est déclenchée par un déplacement de la souris avec le bouton (gauche ici) appuyé. Le constructeur d'une classe de transition comporte un argument textuel optionnel qui indique l'état d'arrivée. Par exemple, la transition *DragOff* (Fig.1, lignes 16-17) mène vers l'état *start*).

Comme illustré sur la figure 2, SWINGSTATES propose un ensemble de classes de transitions qui tire profit de son modèle de dessin structuré pour offrir un plus grand pouvoir d'expression que Java AWT en intégrant en particulier des mécanismes de sélection avancés au niveau de la boîte à outils ("picking"). En effet, pour chaque événement positionnel (click, par exemple), SWINGSTATES propose 3 classes de transition qui permettent de s'abonner à des événements de ce type survenant : sur une forme graphique particulière (*ClickOnShape*), sur toutes les formes graphiques portant une certaine étiquette ou *tag* (*ClickOnTag*), ou survenant n'importe où (*Click*). Bien

que cette approche soit plus flexible pour le traitement des événements, SWINGSTATES repose sur le système d'événements AWT et ne permet donc pas de gérer les périphériques d'entrée non standard. Une première extension pour la gestion d'entrées avancées avait été développée en utilisant la librairie jInput [1]. Mais elle ne permet qu'une connexion directe aux canaux physiques détectés par jInput, sans représentation structurée des périphériques, ni possibilité d'adaptation des données ou de configuration dynamique.

L'architecture de SWINGSTATES reste pourtant ouverte à une gestion plus large des événements grâce à des transitions génériques (*Event*, *EventOnPosition*, *EventOnShape* et *EventOnTag*) qui répondent à des événements virtuels. Le programmeur peut spécifier la classe d'événements capables de déclencher une telle transition dans le constructeur de cette dernière. Par exemple, l'extrait de code ci-dessous montre la définition d'une classe d'événements de type *Zoom* (ligne 1) et une transition qui répond à ce type d'événements (ligne 14) :

```
1 public class Zoom extends VirtualPositionEvent {
2    private double zoomFactor;
3    public Zoom(Point2D zCenter, double zFactor) {
4       super(zCenter);
5       zoomFactor = zFactor;
6    }
7    public double getZoomFactor() {
8       return zoomFactor;
9    }
10 }
11
12 CStateMachine interaction = new CStateMachine() {
13   ...
14   Transition t = new EventOnPosition(Zoom.class) {
15      public void action() {
16         Zoom event = (Zoom)getEvent();
17         zoomBy(event.getPoint(),
18                event.getZoomFactor());
19      }
20   };
21   ...
22 };
```

ICon

ICON repose sur un modèle à flots de données dont les briques, appelées *dispositifs*, sont une généralisation des dispositifs d'entrée : ils peuvent produire des valeurs de sortie mais peuvent aussi en recevoir (Figure 3, en bas). Ces valeurs sont émises sur des *slots de sortie* et reçues sur des *slots d'entrée*. Dans le langage visuel d'ICON, ces slots varient selon leur type (cercles pour les booléens, triangles pour les entiers, etc.) et peuvent être organisés de façon hiérarchique pour représenter des types structurés.

Une *configuration d'entrée* est un flot de données qui relie des périphériques à un noyau applicatif (Figure 3, en haut). Les configurations d'entrée sont construites en interconnectant des dispositifs dans l'éditeur graphique. Pour cela, ICON fournit des *dispositifs système* (qui décrivent des ressources matérielles comme les périphériques d'entrée) et des *dispositifs utilitaires* (allant de simples opérateurs booléens jusqu'à des techniques d'interaction

1. https://jinput.dev.java.net/

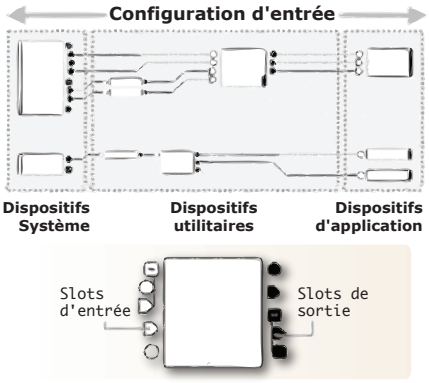


FIGURE *3* : Configuration d'entrée dans ICON.

comme des "toolglass" ou des interpréteurs de gestes, en passant par des retours graphiques tels que des curseurs). De son côté, le programmeur doit concevoir les *dispositifs d'application* qui vont contrôler les fonctionnalités ou les objets de son application.

Comparé au modèle événementiel standard, ce modèle rend la description de l'interaction en entrée plus explicite, plus fine, et plus facilement reconfigurable (y compris pendant l'exécution du programme). Il est aussi beaucoup plus facile à étendre. En effet, avec une boîte à outils classique, prendre en charge un nouveau périphérique d'entrée ou une nouvelle technique d'interaction nécessite de : 1) définir un nouveau type d'événement lié au périphérique ; 2) modifier et adapter le mécanisme de propagation des événements ; 3) étendre les objets qui doivent réagir à ces événements. Avec ICON, il suffit de créer un dispositif qui encapsule le nouveau périphérique ou la nouvelle interaction, et externaliser son interface sous la forme de slots d'entrée et/ou de sortie. Le périphérique ou la technique d'interaction peuvent alors être réutilisés dans de nombreuses configurations d'entrée.

Enfin, avec ICON, le noyau applicatif et les techniques d'interaction ne sont plus câblés à des périphériques d'entrée spécifiques. Il leur suffit de déclarer les canaux d'entrée dont ils ont besoin pour fonctionner. C'est, entre autres, cette propriété dont nous avons tiré parti pour l'association entre SWINGSTATES et ICON, comme nous le décrivons dans la section suivante.

FLOWSTATES

Cette section illustre, via des scénarios, l'utilisation de FLOWSTATES et en détaille les mécanismes sous-jacents.

Contrôle de machines SwingStates avec ICon

Bob est étudiant en Master IHM. Dans le cadre d'un projet de cours, il doit prototyper un éditeur de dessin prenant en charge les opérations de déplacement et de changement d'échelle (*pan* et *zoom*). Il n'a pas encore fait le choix des techniques de navigation qu'il désire rendre disponibles

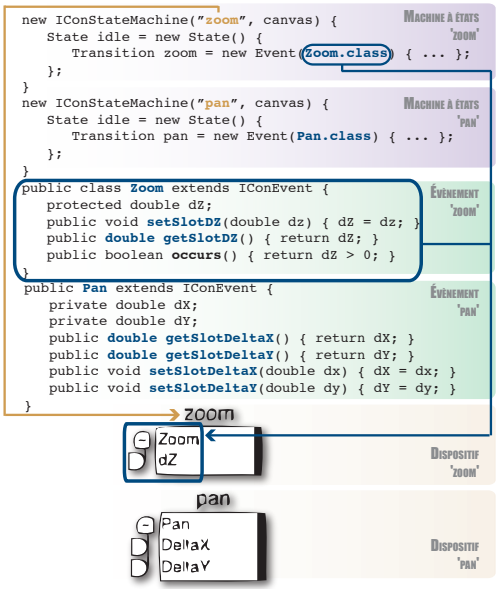


FIGURE *4* : Le programme SWINGSTATES (partie supérieure) permettant de générer les périphériques logiques ICON (partie inférieure)

pour l'utilisateur final. Il commence donc par programmer des interactions avec SWINGSTATES en utilisant des événements virtuels de *Pan* et de *Zoom* (figure 4).

Bob instancie d'abord deux machines à états de type IConStateMachine, une pour le pan et une pour le zoom. Il déclare ensuite les classes d'événements Pan et Zoom de type IConEvent (qui étend la classe VirtualEvent de SWINGSTATES). Dans chacune de ces deux classes, il déclare les données qui composent l'événement en ajoutant des méthodes spécifiques pour y accéder et les modifier. Par exemple, un événement *Zoom* comporte un réel dZ qui correspond au changement d'échelle à appliquer. Dans la classe Zoom, Bob déclare donc un champ dZ et les méthodes getSlotDZ et setSlotDZ.

FLOWSTATES transforme automatiquement chaque machine à états en dispositif ICON selon le mécanisme suivant : à l'instanciation d'une machine, les transitions sont explorées afin de découvrir les événements de type IConEvent. Pour chaque type d'événement, FLOWSTATES crée un groupe de slots d'entrée contenant un slot par couple (getSlot ?,setSlot ?) déclaré dans l'événement. Par exemple, un événement Pan est représenté par un groupe de slots nommé *Pan* et composé de deux slots réels DeltaX et DeltaY (figure 4). Au moment de l'exécution, à chaque fois qu'un slot d'entrée recevra un nouveau signal au niveau d'ICON, un événement sera crée et envoyé à la machine à états. La génération dynamique des événements se fait par des mécanismes d'introspection, qui instancient et initialisent les objets événement via leurs accesseurs.

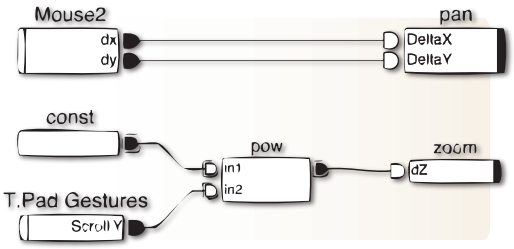


FIGURE *5* : Navigation multi-échelle bimanuelle. Le *Pan* est contrôlé par une souris dédiée, et le *Zoom* est contrôlé par des gestes verticaux sur un Trackpad.

Par défaut, un événement est créé si au moins l'un des slots qui le composent reçoit un signal et si tous les slots ont été initialisés. Des conditions supplémentaires peuvent être spécifiées en surchargeant la méthode occurs de la classe d'événement correspondante. Par exemple, il est possible d'imposer qu'un événement Zoom ne soit généré que lorsque la valeur du champ dZ est strictement positive. Pour construire des conditions plus complexes, FLOWSTATES fournit aussi des mécanismes permettant d'effectuer des tests sur les slots associés, pour déterminer notamment quels slots ont été mis à jour.

Lorsque Bob lance son programme, il voit dans l'éditeur graphique d'ICON ses deux machines à états sous la forme de deux dispositifs (figure 4). Il peut alors brancher ses machines à états sur les entrées physiques qu'il désire, sachant qu'il n'est pas contraint par le système et peut directement commencer à explorer des techniques non-standard. Inspiré par l'article qu'il a étudié en cours sur l'efficacité d'une configuration bimanuelle pour la navigation multi-échelle [7] et disposant d'un ordinateur portable équipé d'un trackpad ainsi que de deux souris, Bob décide alors de tester une configuration à deux mains : le trackpad servira au *zoom*, une des deux souris sera dédiée au *pan*, et l'autre souris contrôlera le pointeur système. Il couple donc le changement de *Zoom* aux gestes verticaux sur son trackpad, et le changement de *Pan* aux déplacements d'une des deux souris (Figure 5). Sur cette figure, le dispositif utilitaire d'ICON "*pow*" (qui réalise l'opération $in1^{in2}$) est utilisé pour transformer des déplacements en changements d'échelle relatifs selon la formule $dZ = 1.1^{scrollY}$.

Nous avons vu dans ce premier exemple les principes de base du contrôle d'une machine à états SWINGSTATES avec un dispositif ICON et l'approche déclarative qui en résulte : le programmeur déclare les événements de haut niveau dont il a besoin pour contrôler les interactions plutôt que de programmer l'interaction autour des événements qui lui sont imposés par le modèle et la boîte à outils (en général très fortement liés aux périphériques d'entrée standard). Il peut ensuite spécifier comment sont émis ces événements par des configurations d'entrée ICON.

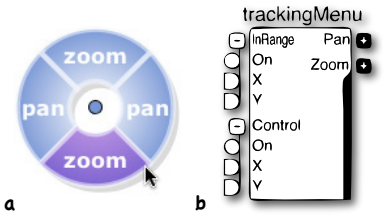


FIGURE *6* **:** Un tracking menu pour les changements d'échelle et les déplacements

Une machine plus complète : le Tracking Menu

Bob est maintenant satisfait de sa configuration bimanuelle, mais réalise que tout le monde ne partage pas sa configuration matérielle. Il décide donc de programmer un *tracking menu* [16] au cas où seulement un seul dispositif de pointage est présent. Un tracking menu est un menu circulaire initialement destiné à être utilisé avec un stylet. Une fois activé, il reste toujours visible et suit le curseur tant que le stylet reste proche de la surface de la tablette (mode 'tracking'). Une commande est sélectionnée au moment où le stylet touche la surface, et un paramètre peut être ensuite ajusté à la manière d'un *control menu* [31]. La figure 6a montre le rendu graphique du menu tandis que la partie droite de la figure 7, extraite de l'article de Fitzmaurice et al. [16], décrit le comportement de ce menu. Le rendu graphique a été programmé avec le graphe de scène de SWINGSTATES et ne sera pas détaillé ici.

La partie gauche de la figure 7 montre le code que Bob a écrit pour reproduire la machine à états de droite. Comme pour les machines habituellement manipulées dans SWINGSTATES, les conditions de déclenchement des transitions sont réparties entre le type de transition, les arguments passés à cette transition et, le cas échéant, la garde.

Il y a deux grandes familles de transitions disponibles dans FLOWSTATES, *Switch* et *Event*, qui permettent de décrire respectivement les *changements de mode* et les *entrées continues*. Jusqu'ici nous n'avons utilisé que des transitions de type *Event*. Un exemple de transition de type *Switch* est donné Fig.7, ligne 11 : la transition stopTracking est déclenchée à chaque fois que le stylet s'éloigne de la surface de la tablette, ce qui correspond à la réception d'un événement InRange dont le booléen on est passé à faux. De manière générale, une transition *Switch* attend en constructeur une classe d'événements qui hérite de SwitchEvent et contient un champ on. En fonction du deuxième argument passé au constructeur (SWITCH_ON ou SWITCH_OFF), la transition sera déclenchée à chaque fois que le champ on de l'événement passe à vrai ou à faux. Comme pour n'importe quel champ d'événement, la sémantique de on est indéfinie : elle dépendra de ce qui lui sera connecté dans ICON.

Nous avons déjà vu précédemment que des événements pouvaient être déclenchés sur des formes graphiques particulières (en utilisant par exemple la classe ClickOnShape). Ceci est également vrai pour les Switch. De manière générale, chaque type de transition possède quatre variantes : une variante *absolue*, et trois variantes *positionnelles* suffixées par OnPosition, OnShape et OnTag. Les transitions positionnelles attendent en argument une classe d'événements qui contient au minimum les champs x et y, interprétés comme une position dans la scène graphique. Ceci permet notamment au développeur de s'affranchir de la gestion du *picking* graphique.

Dans notre exemple, le picking graphique est requis parce que les comportements diffèrent selon l'endroit où les événements surviennent. Par exemple, le menu doit passer en mode *tracking* lorsque le stylet s'approche de la tablette (événement InRange), et il doit être repositionné si le stylet sort du menu. Ce comportement est spécifié Fig.7, lignes 5-6 : la transition startTrackingInMenu ne sera déclenchée que si l'événement survient au-dessus du menu (toutes les formes du tracking menu portent un tag MenuItem), alors que la transition startTrackingOutMenu sera déclenchée quelle que soit sa position. Des conditions plus fines peuvent également être décrites avec des gardes. Par exemple, une garde permet de repositionner le menu lorsque le pointeur se trouve à l'extérieur de celui-ci dans l'état *tracking* (Fig.7, lignes 17-21).

Notons pour finir que notre automate utilise seulement deux classes d'événements : InRange pour le passage de l'état *outOfRange* vers l'état *tracking*, et Control pour le passage de l'état *tracking* vers l'état *touching*. Lorsque la même classe d'événements apparaît dans plusieurs transitions, FLOWSTATES ne crée qu'un seul groupe de slots sur le dispositif ICON (Figure 8). Par exemple, le type d'événements Control, utilisé par plusieurs transitions pour modéliser l'entrée dans le mode *control* et l'interprétation des événements positionnels jusqu'à la sortie du mode (Fig.7, lignes 12-13, 25-26 et 27), ne se retrouve qu'une fois dans le dispositif ICON. Cette convention de nommage permet de spécifier que des transitions sont liées entre elles en termes de logique d'interaction.

Connexion de deux machines

Bob ayant écrit la machine à états du tracking menu, il lui reste à la connecter avec les machines de navigation pan et zoom décrites précédemment. Pour ce faire, il peut utiliser les mécanismes déjà présents dans SWINGSTATES en faisant émettre des événements Pan et Zoom par la machine du menu, qui seront écoutés par les machines de navigation (une machine peut émettre des événements en utilisant sa méthode fireEvent). L'extrait de code suivant abonne les machines de navigation à la machine du tracking menu :

```
1 smTrackingMenu.addStateMachineListener(smPan);
2 smTrackingMenu.addStateMachineListener(smZoom);
```

```
1 IConStateMachine smTrackingMenu = new IConStateMachine("trackingMenu", canvas) {
2    MenuItem currentItem;
3
4    State outOfRange = new State() {
5       Transition startTrackingInMenu = new SwitchOnTag(MenuItem.class, InRange.class, SWITCH_ON, ">> tracking") { };
6       Transition startTrackingOutMenu = new SwitchOnPosition(InRange.class, SWITCH_ON, ">> tracking") {
7         public void action() { menu.showMenu(getPoint()); }
8       };
9    };
10   State tracking = new State() {
11      Transition stopTracking = new Switch(InRange.class, SWITCH_OFF, ">> outOfRange") { };
12      Transition startControl =
13      new SwitchOnTag(MenuItem.class, Control.class, SWITCH_ON, ">> touching") {
14        public void action() { currentItem = (MenuItem)getTag(); }
15      };
16      Transition moveOutMenu = new EventOnPosition(InRange.class) {
17        public boolean guard () {
18          double d = menu.getCurrentPosition().distance(getPoint());
19          return d > menu.getRadius();
20        }
21        public void action() { /* compute dx, dy */ menu.moveBy(dx, dy); }
22      };
23    };
24    State touching = new State() {
25      Transition stopControl =
26      new SwitchOnPosition(Control.class, SWITCH_OFF, ">> tracking") { };
27      Transition control = new EventOnPosition(Control.class) {
28        public void action() { /* perform action associated with current menu item */ }
29    };
30  };
31 };

// classes d'événements
public class InRange extends IConPositionSwitchEvent { }
public class Control extends IConPositionSwitchEvent { }
```

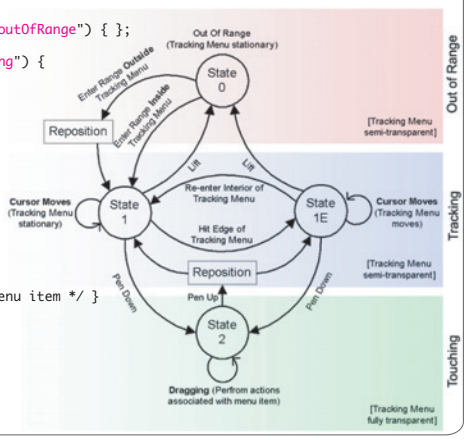


FIGURE **7 :** L'automate décrivant le comportement d'un tracking menu (extrait de [16]), et le code SWINGSTATES correspondant.

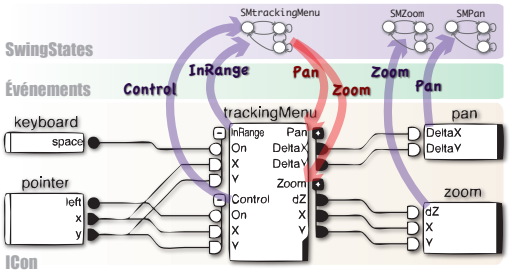


FIGURE ***8*** **:** Connecter les machines via leur représentation en périphériques logiques. Bien que le tracking menu soit prévu pour être utilisé avec une tablette, les entrées utilisées ici sont la souris et la touche espace du clavier pour simuler la proximité du stylet.

Mais FLOWSTATES permet aussi de transformer le menu en un dispositif utilitaire ICON qui pourra être intégré dans le flot de données via le configurateur graphique (Figure 8). Pour cela, Bob surcharge dans un premier temps la méthode getOutputTypes de la machine afin qu'elle retourne le tableau des types d'événements générés par la machine (lignes 2-4 dans le code qui suit). Ainsi, FLOWSTATES peut créer les groupes de slots de sortie adéquats sur le dispositif ICON de la machine (figure 8).

Ensuite, Bob doit écrire le code pour que la machine à états émette des événements de type Pan ou Zoom au cours de l'interaction, c'est-à-dire lors des transitions *control* de l'état *touching* pour notre exemple du tracking menu. Il va utiliser pour cela la méthode fireEvent qui va permettre à FLOWSTATES de mettre à jour les slots de sortie correspondants du dispositif (lignes 9-21 dans le code qui suit). Il est alors possible de connecter la sortie du tracking menu aux interactions Pan et Zoom décrites précédemment, comme l'illustre la figure 8.

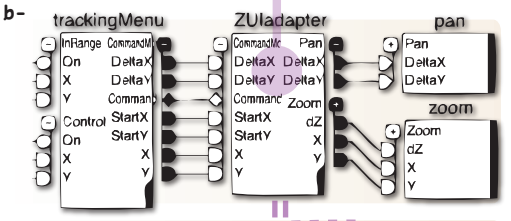
```
1 smTrackingMenu = new IConStateMachine([...]) {
2   public Class[] getOutputTypes() {
3     return new Class[] {Pan.class, Zoom.class};
4   }
5   State touching = new State() {
6     [...]
7     Transition control = new EventOnPosition(
8                                 Control.class) {
9       public void action() {
10        if (currentItem.getName().equals("zoom")) {
11          Zoom event = new Zoom();
12          [...]
13          fireEvent(event);
14        } else {
15          if(currentItem.getName().equals("pan")) {
16            Pan event = new Pan();
17            [...]
18            fireEvent(event);
19          }
20        }
21      }
22    };
23  };
24 };
```

Réalisation d'une technique d'interaction générique

Bob est satisfait de son tracking menu et pense que d'autres utilisateurs d'ICON pourraient en profiter. Cependant, sa technique est peu réutilisable dans l'état actuel : les sorties ont été spécialisées pour être dirigées vers les techniques Pan et Zoom. En effet, les valeurs d'entrées sont adaptées dans le code de la machine à états pour produire les bonnes valeur de sorties (conversion des déplacements et calcul du facteur de zoom aux lignes 12 et 17 du code précédent qui ont été masquées pour plus de

a-
```
1 IConStateMachine zuiAdapter =
2 new IConStateMachine ("ZUIAdapter", canvas) {
3     public Class<? extends OutSlotEvent>[] getOutputTypes() {
4       return new Class[] {Pan.class, Zoom.class};
5     }
6     State idle = new State() {
7       double zoomMin = 0.01;
8       double deltaZ = Math.pow(zoomMin, 1/getCanvas().getWidth());
9       Transition cmd = new EventOnPosition(CommandMove.class) {
10        public void action() {
11          CommandMove event = (CommandMove) getEvent();
12          if (event.getSlotCommand().equals("zoom")) {
13            Zoom zEvent = new Zoom();
14            double deltaX = event.getSlotDeltaX();
15            zEvent.setSlotX(event.getSlotStartX());
16            zEvent.setSlotY(event.getSlotStartY());
17            if (deltaX > 0)
18              zEvent.setSlotdZ(Math.pow(deltaZ, deltaX));
19            else
20              zEvent.setSlotdZ(1.0/Math.pow(deltaZ, -deltaX));
21            fireEvent(zEvent);
22          } else if (event.getSlotCommand().equals("pan")) {
23            Pan pEvent = new Pan();
24            pEvent.setSlotDeltaX((int)event.getSlotDeltaX());
25            pEvent.setSlotDeltaY((int)event.getSlotDeltaY());
26            fireEvent(pEvent);
27        } }
28    };};
29 };
```

b-



c-

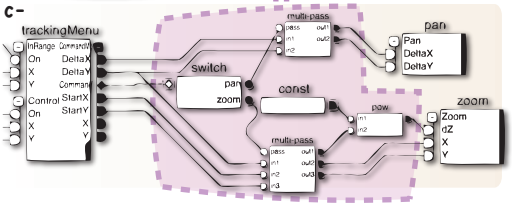


FIGURE ***9*** **:** Les machines à états comme adaptateurs ICon

lisibilité). Ce mécanisme peut cependant être généralisé afin de proposer un instrument "tracking menu générique" qui pourra être branché via ICON à des objets du domaine proposant d'autres opérations que les déplacements et changements d'échelle selon les principes de l'interaction instrumentale [5].

Bob modifie alors légèrement le code de la machine du tracking menu afin que cette dernière n'envoie plus que des événements de la classe générique CommandMove pendant la phase de contrôle de la commande. Un événement CommandMove est composé des champs génériques suivants : DeltaX et DeltaY (déplacement), Command (l'intitulé de la commande), StartX et StartY (la position au moment de la sélection de la commande) et X et Y (la position courante). Le dispositif aura maintenant les slots de sortie correspondants, comme illustré dans la figure 9-b, et pourra être utilisé pour d'autres applications.

Réalisation d'un adaptateur

Maintenant que Bob a réalisé une technique d'interaction générique, il ne lui reste plus qu'à adapter ses sorties aux entrées spécialisées des techniques Pan et Zoom. Il peut

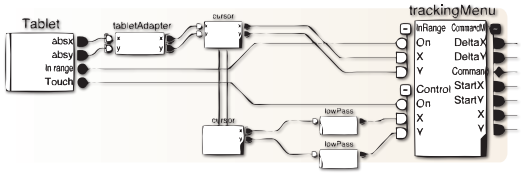


FIGURE ***10*** **:** Ajout d'un comportement de type "trailing widget" avec ICon.

le faire directement au niveau d'ICON, en utilisant des dispositifs utilitaires (Figure 9-c). Cependant, l'approche en flot de données peut s'avérer complexe pour décrire des aiguillages et des contrôles comme requis dans ce cas. Bob préfère décrire ce comportement avec des machines à états, et voudrait également pouvoir l'encapsuler pour pouvoir le distribuer avec sa technique. Il choisit donc de développer un dispositif *ZUIadapter* à partir d'une machine a états (Figures 9-a et 9-b). Le rôle de cette machine est très proche de celui des *transducteurs d'événements* de Accot et al. [2] : il traduit des événements d'un niveau d'abstraction à l'autre. Dans notre exemple, il transforme les événements génériques CommandMove en événements spécifiques Pan et Zoom.

Discussion

Les différents exemples ci-dessus illustrent les possibilités multiples qu'offre FLOWSTATES pour la réalisation de dispositifs ICON à l'aide de machines à états, que ce soit des dispositifs d'interaction spécialisés ou génériques (les deux versions du tracking menu) ou des dispositifs utilitaires (l'adaptateur ZUIadapter). Ils démontrent la flexibilité de FLOWSTATES entre les deux modèles machine à états/flot de données : le programmeur peut choisir où se situe la frontière entre les deux approches en fonction du problème qu'il veut résoudre et de ses besoins.

Pour illustrer à nouveau ce point, Bob ou d'autres utilisateurs de sa technique peuvent facilement doter le tracking menu d'un comportement à la "trailing widget" [17] en insérant un filtre passe-bas entre le périphérique de pointage et le tracking menu dans le flot de données ICON (Figure 10). Le menu va alors "suivre" le pointeur avec un retard qui permet d'interagir avec le fond de l'application (pour dessiner par exemple). Cette fois, le flot de données est plus adapté car implémenter ce comportement avec SWINGSTATES aurait demandé de faire le changement de coordonnées dans chaque transition.

COMPARAISON AVEC L'ETAT DE L'ART

Peu de travaux à notre connaissance ont traité de la combinaison de l'approche à flots de données et de celle dirigée par les états pour la spécification de systèmes interactifs, approches que nous qualifions d'*hybrides*. Le langage Lustre [18] est probablement le plus ancien. Il propose des constructions pour décrire des flots de données synchrones, qui peuvent être mélangées avec des structures

de contrôle classiques mais qui ne sont pas explicitement décrites par des machines à états comme avec FLOWSTATES. Intuikit [9] est un second exemple qui intègre flot de données et contrôle. Dans Intuikit, une interface est un ensemble de composants non seulement capables de produire des événements et d'en consommer (notamment grâce à des machines à états) mais aussi capables d'écouter des changements de propriétés et de changer des propriétés. Flot de données et machines à états sont donc fortement intégrés au sein d'un même composant alors que FlowStates propose une approche plus structurée. Les machines à états peuvent en effet aussi être utilisées pour le contrôle au sein d'un device ICON mais FlowStates permet également au programmeur de s'abstraire de la logique flot de données lors de la programmation du comportement avec une machine à états, l'externalisation vers le flot de données étant faite automatiquement à partir de la machine. Par ailleurs, Lustre comme Intuikit reposent sur des langages purement textuels.

Les travaux les plus proches de notre approche sont le modèle PMIW [22], l'architecture VRMeer [13] et l'environnement qui intègre PetShop et ICON [28].

Dans PMIW / VRMeer, l'interaction est spécifiée en deux parties : la logique de l'interaction est représentée par un formalisme dirigé par les états alors que chaque état peut contenir un ensemble de relations continues entre variables. VRMeer et PMIW couvrent cependant un ensemble restreint d'entrées physiques. Par exemple, PMIW se limite aux événements clavier et souris couverts par le système X Window et les événements d'un Polhemus et du dispositif de suivi de regard ISCAN. PMIW est d'ailleurs plus une preuve de concept qu'un réel outil de développement et de prototypage.

Dans le couple PetShop/ICON, la logique de l'interaction est spécifiée par des réseaux de Petri dans l'environnement graphique de PetShop, et la gestion des dispositifs d'entrée est déléguée à ICON. FLOWSTATES et PetShop visent cependant des objectifs différents : PetShop est principalement dédié à la spécification et à la vérification formelle, alors que FLOWSTATES privilégie la simplicité, parfois au détriment d'une approche plus formelle. En conséquence, nous avons choisi les machines à états qui présentent l'avantage d'être plus faciles à manipuler. En outre, cette logique est programmée en Java dans SWINGSTATES alors qu'elle est spécifiée visuellement dans PetShop. Bien que de manière générale les langages visuels se prêtent bien au prototypage, ils introduisent des couches de traduction qui peuvent rendre difficile l'identification de l'origine d'une erreur dans le processus de prototypage. En effet, PetShop comme PMIW requièrent de spécifier visuellement la logique de l'interaction, puis d'écrire des morceaux de programme pour la lier au noyau fonctionnel. Dans FLOWSTATES, cette couche de traduction est absente. La seule couche de traduction se situe entre le langage visuel d'ICON et la syntaxe de SWINGSTATES. Par contre, nous pensons que le recours à un langage visuel pour le traitement des entrées est pleinement justifié, car comparé à la logique de l'interaction, c'est un aspect qui est très changeant et volatile : les périphériques d'entrée peuvent être très variables d'une machine à l'autre ou nécessiter des reconfigurations au cours de l'exécution d'une application. De plus, cette partie peut être totalement découplée du code du noyau applicatif. Nous avons vu que dans FLOWSTATES, ces deux aspects peuvent être clairement séparés, voire spécifiés par des concepteurs différents.

Pour finir, l'intégration entre SWINGSTATES et ICON va plus loin que l'intégration entre PetShop et ICON. D'une part, FLOWSTATES exploite le graphe de scène de SWINGSTATES et offre donc une prise en charge implicite des mécanismes graphiques. Dans PetShop/ICON, ceux-ci doivent être spécifiés à la main, nécessitant alors la programmation et l'ajout de dispositifs *de service* ICON. D'autre part, les automates de FLOWSTATES peuvent fournir des sorties vers ICON, ce qui permet au programmeur de basculer d'un modèle à l'autre selon sa convenance. Ce dialogue à "double-sens" et cette flexibilité entre les modèles permettent notamment de décrire avec des automates des techniques d'interaction qui sont indépendantes du noyau applicatif, et qui peuvent ensuite être connectées en cascade avec ICON. Cette approche inspirée de l'interaction instrumentale [5] et des architectures à composants offre plus de flexibilité que le modèle à couches classique de Seeheim ou de Arch [1] pour la spécification d'interactions avancées.

CONCLUSION

Les passerelles entre flots de données et machines à états développées dans FLOWSTATES offrent un grand pouvoir d'expression et une grande flexibilité pour le prototypage de l'interaction. SWINGSTATES, basée sur les machines à états, est particulièrement appropriée à la spécification de la logique de l'interaction tandis qu'ICON, basée sur les flots de données et la programmation visuelle, se prête tout à fait à la spécification de relations continues et dynamiques. L'intégration des machines à états dans des flots de données permet non seulement de faciliter la gestion du contrôle et de l'aiguillage mais aussi d'offrir un vocabulaire d'entrée très riche aux machines à états (des périphériques d'entrée standard comme la souris aux périphériques les plus "exotiques" comme ceux de la Nintendo wii en passant par des commandes vocales).

FLOWSTATES (http ://www.lri.fr/∼appert/FlowStates) fait le pont entre deux boîtes à outils sans modifier les paradigmes de programmation sous-jacents de chacune des boîtes à outils dans le souci de favoriser leur adoption par les développeurs qui sont familiers avec l'une et/ou l'autre. Nos pistes futures s'inscrivent dans cette lignée avec l'intégration de la boîte à outils ZVTM [29] pour la programmation du rendu graphique. En effet, si SWINGSTATES propose un graphe de scène adapté à la pro-

grammation de scènes vectorielles, elle n'offre pas les mécanismes fins de rendu au niveau pixel de ZVTM pour la programmation de techniques composant plusieurs vues comme les Sigma Lenses [30]. Nous prévoyons également de tester la couverture et l'utilisabilité de FLOWSTATES grâce à un benchmark incluant la réalisation de techniques d'interaction avancées. Enfin, si le but premier de FLOWSTATES est le prototypage, des langages de spécification en Java comme Biscotti [10] permettraient d'instrumenter les machines à états pour faire de la vérification formelle.

REMERCIEMENTS

Ce travail a bénéficié du soutien de l'ANR - projet iStar.

BIBLIOGRAPHIE

1. A metamodel for the runtime architecture of an interactive system : the UIMS tool developers workshop. *SIGCHI Bull.*, 24(1) :32–37, 1992.
2. J. Accot, S. Chatty, and S. Maury. Formal transducers : Models of devices and building bricks for the design of highly interactive systems. In *Proc. DSV-IS'97*, 143–160. Springer-Verlag, 1997.
3. C. Appert and M. Beaudouin-Lafon. SwingStates : Adding state machines to Java and the Swing toolkit. *Software : Practice and Experience*, 38(11) :1149 – 1182, 2008.
4. R. Ballagas, M. Ringel, M. Stone, and J. Borchers. iStuff : a physical user interface toolkit for ubiquitous computing environments. In *Proc. CHI'03*, 537–544. ACM, 2003.
5. M. Beaudouin-Lafon. Instrumental interaction : an interaction model for designing post-WIMP user interfaces. In *Proc. CHI'00*, 446–453. ACM, 2000.
6. R. Blanch and M. Beaudouin-Lafon. Programming rich interactions using the hierarchical state machine toolkit. In *Proc. AVI'06*, 51–58. ACM, 2006.
7. F. Bourgeois and Y. Guiard. Multiscale pointing : facilitating pan-zoom coordination. In *Proc. CHI EA'02*, 758–759. ACM, 2002.
8. W. Buxton. A three-state model of graphical input. In *INTERACT*, volume 90, 449–456, 1990.
9. S. Chatty, A. Lemort, and S. Vales. Multiple input support in a model-based interaction framework. In *Proc. TABLETOP '07*, 179–186, Oct. 2007.
10. C. D. T. Cicalese and S. Rotenstreich. Behavioral specification of distributed software component interfaces. *Computer*, 32(7) :46–53, 1999.
11. Cycling '74. max/msp/jitter. `http://www.cycling74.com/`.
12. Dassault Systemes. Virtools Dev. `www.virtools.com/`.
13. G. de Haan and F. H. Post. Flexible architecture for the development of realtime interaction behavior. In *Workshop VR'08*. IEEE Computer Society, 2008.
14. P. Dragicevic and J.-D. Fekete. Support for input adaptability in the ICON toolkit. In *Proc. ICMI'04*, 212–219. ACM, 2004.
15. P. Figueroa, M. Green, and H. J. Hoover. InTml : a description language for VR applications. In *Proc. Web3D '02*, 53–58. ACM, 2002.
16. G. Fitzmaurice, A. Khan, R. Pieké, B. Buxton, and G. Kurtenbach. Tracking menus. In *Proc. UIST'03*, 71–79. ACM, 2003.
17. C. Forlines, D. Vogel, and R. Balakrishnan. Hybridpointing : fluid switching between absolute and relative pointing with a direct input device. In *Proc. UIST'06*, 211–220. ACM, 2006.
18. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. *Proc. of the IEEE*, 79(9) :1305–1320, 1991.
19. K. Hinckley, M. Czerwinski, and M. Sinclair. Interaction and modeling techniques for desktop two-handed input. In *Proc. UIST'98*, 49–58. ACM, 1998.
20. S. Hudson, J. Mankoff, and I. Smith. Extensible input handling in the subArctic toolkit. In *Proc. CHI'05*, 381–390. ACM, 2005.
21. S. Huot, C. Dumas, P. Dragicevic, J.-D. Fekete, and G. Hégron. The MaggLite post-WIMP toolkit : Draw it, connect it and run it. In *Proc. UIST'04*, 257–266. ACM, 2004.
22. R. J. K. Jacob, L. Deligiannidis, and S. Morrison. A software model and specification language for non-WIMP user interfaces. *ACM Trans. Comput.-Hum. Interact.*, 6(1) :1–46, 1999.
23. W. A. König, R. Rädle, and H. Reiterer. Squidy : a zoomable design environment for natural user interfaces. In *Proc. CHI EA'09*, 4561–4566. ACM, 2009.
24. Meso Group. vvvv : a multipurpose toolkit. `http://vvvv.org/`.
25. B. Myers. Separating application code from toolkits : eliminating the spaghetti of call-backs. In *Proc. UIST'91*, 211–220. ACM, 1991.
26. B. Myers, D. Giuse, R. Dannenberg, B. Zanden, D. Kosbie, E. Pervin, A. Mickish, and P. Marchal. Garnet : Comprehensive support for graphical, highly interactiveuser interfaces. *Computer*, 23(11) :71–85, 1990.
27. B. Myers, R. McDaniel, R. Miller, A. Ferrency, A. Faulring, B. Kyle, A. Mickish, A. Klimovitski, and P. Doane. The Amulet environment : new models for effective user interfacesoftware development. *IEEE Trans. Soft. Eng.*, 23(6) :347–365, 1997.
28. D. Navarre, P. Palanque, P. Dragicevic, and R. Bastide. An approach integrating two complementary model-based environments for the construction of multimodal interactive applications. *Interact. Comput.*, 18(5) :910–941, 2006.
29. E. Pietriga. A toolkit for addressing hci issues in visual language environments. In *Proc. VL/HCC'05*, 145–152. IEEE Computer Society, 2005.
30. E. Pietriga and C. Appert. Sigma lenses : focus-context transitions combining space, time and translucence. In *Proc. CHI'08*, 1343–1352. ACM, 2008.

31. S. Pook, E. Lecolinet, G. Vaysseix, and E. Barillot. Control menus : excecution and control in a single interactor. In *Proc. CHI EA'00*, 263–264. ACM, 2000.
32. SENSE8. World Up. http://www.sense8.com/.

TorusDesktop: Pointing via the Backdoor is Sometimes Shorter

Stéphane Huot[1,2] huot@lri.fr
Olivier Chapuis[1,2] chapuis@lri.fr
Pierre Dragicevic[2] dragice@lri.fr

[1]LRI - Univ. Paris-Sud & CNRS
F-91405 Orsay, France

[2]INRIA
F-91405 Orsay, France

ABSTRACT

When pointing to a target on a computer desktop, we may think we are taking the shortest possible path. But new shortcuts become possible if we allow the mouse cursor to jump from one edge of the screen to the opposite one, i.e., if we turn the desktop into a torus. We discuss the design of TORUSDESKTOP, a pointing technique that allows to wrap the cursor around screen edges to open this pointing backdoor. A dead zone and an off-screen cursor feedback make the technique more usable and more compatible with everyday desktop usage. We report on three controlled experiments conducted to refine the design of the technique and evaluate its performance. The results suggest clear benefits of using the backdoor when target distance is more than 80% the screen size in our experimental conditions.

Author Keywords

Pointing Technique, Cursor Wrapping, Torus.

ACM Classification Keywords

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

General Terms

Human Factors, Experimentation.

INTRODUCTION

When flying from New-York to San Francisco, one usually does not fly around the globe across the Atlantic and the Pacific Oceans. Yet we often do it on our computers: we routinely move our mouse pointer from one side of the screen to the opposite side – e.g., to select a tool or invoke a menu command – ignoring potential trajectory shortcuts. Such shortcuts would only require a small modification to the mouse behavior: when the pointer goes past a screen edge it re-appears on the opposite side, as in the Asteroids or Pac-Man video-games (see Figure 1).

We introduce TORUSDESKTOP, a pointing technique which opens these shortcuts on our computer desktops. Although many pointing facilitation techniques have been already proposed, most of them are *target-aware* [26, 3], i.e., they require knowledge of all the potential targets the user may acquire. These techniques can be extremely efficient but they are sensitive to distractors and are difficult to integrate to existing systems. Only a few *target-agnostic* pointing facilitation techniques have been introduced and the results have been mixed. TORUSDESKTOP is target-agnostic, making it easy to integrate to existing systems and compatible with most existing pointing facilitation techniques.

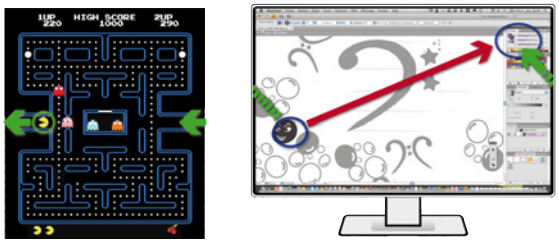


Figure 1. The Pac-Man video-game and a case scenario where pointing through screen edges could be beneficial.

TORUSDESKTOP teleports the mouse cursor to the opposite side of the screen when it goes past one of the screen's edges. This technique is sometimes referred to as *cursor wrapping*. One consequence of this wrapping behavior is that the shortest path between two points is not necessarily the on-screen segment that connects them. Although this may evoke a sphere topology, wrapping the cursor around screen edges actually turns the computer desktop into a torus.

The idea of wrapping the mouse cursor around screen or window edges is not new. In addition to video games from the early 80's, a few system tweaks and mouse drivers support this technique. But current implementations are all under-designed as the cursor immediately jumps when it reaches a screen edge. This can yield several problems: first, it is easy to trigger the wrapping inadvertently. Second, it might be difficult to find the new location of the cursor. Third, the technique prevents the user from using the border to acquire targets that are located on screen edges. TORUSDESKTOP addresses these issues by introducing a wrapping dead zone and visual feedback to anticipate cursor jumps.

As cursor wrapping has never been studied experimentally, it is not clear whether it should be supported natively by operating systems and better publicized among end users, or simply abandoned. Our initial Fitt's Law simulations (con-

S. Huot, O. Chapuis and P. Dragicevic. TorusDesktop: Pointing via the Backdoor is Sometimes Shorter. In CHI '11: Proceedings of the SIGCHI Conference on Human Factors and Computing Systems, 829-838, ACM, May 2011.
Authors Version
doi: http://doi.acm.org/10.1145/1978942.1979064

sidering all possible pointing tasks on a 2560x1600 display with 40-pixel targets) suggest that cursor wrapping should outperform direct pointing in more than 40% of all possible pointing tasks. But it is unlikely that the question can be adequately answered by a naive Fitts' Law simulation: choosing to use cursor wrapping or not might have an impact on efficiency, and large cursor jumps might be distracting to users and could result in a drop in performance.

Thus we conducted three controlled experiments to refine our design and evaluate its performance. The results of the two first experiments identify the best off-screen feedback, and suggest that a dead zone of $5 - 10\%$ the size of the screen should be provided to enable edge pointing. Our final experiment confirms that our naive Fitts' Law simulation is overly optimistic as it does not account for factors such as the distraction produced by cursor teleportation or the cost of having to chose whether or not to use the backdoor. Nevertheless, our experiment reveals that TORUSDESKTOP is still faster than direct pointing for targets whose distance is greater than 80% the width of a 2560-pixel wide display. This suggests that enabling cursor wrapping is worthwhile, especially in situations where commonly-accessed widgets are located close to the edges of the screen (Figure 1) or when going back-and-forth between two very distant targets.

RELATED WORK

A fundamental tool in the area of target acquisition is Fitts' law [20]. This law models the movement time to acquire a target of size W at distance D as a linear function of an index of difficulty ID usually defined as $\log_2\left(\frac{D}{W}+1\right)$. According to this law, techniques that try to facilitate pointing increase W, reduce D or do both [3]. They are either target-aware or target-agnostic.

Target-Aware Techniques

Most techniques that increase W are target-aware. They either expand the targets themselves [21] – sometimes in the motor space only [27, 9] – or expand the cursor's activation area [15, 12]. Target-aware techniques for reducing D try to predict the target(s) the user wants to acquire. They then bring the cursor closer to the target [2, 16] or bring potential targets closer to the cursor [4]. Another way to reduce D is to use a grid of cursors and a target-aware algorithm that tries to select the appropriate cursor [19].

However, target-aware techniques fail when there are a large number of potential targets, and they are difficult to implement at a system-wide level because they require access to target information that is solely available at a system-level.

Target-Agnostic Techniques

Effective target-agnostic pointing facilitation techniques are relatively rare. Speed-adaptive C-D gain has been modeled as a technique that increases W in motor space, but experiments did not confirm the improvements predicted by the model [11]. Angle Mouse adapts C-D gain to trajectory curvature, but it has been shown to only benefit motor-impaired users [26]. Finally, visual and motor-space uniform magnification (i.e., W and D increased in the same proportion) have been shown to improve pointing performance, but only for very small targets [24].

Other techniques employ more than one input device at a time. For example, D can be reduced in a target-agnostic manner using eye tracking. MAGIC [28] uses eye tracking to define an area where the pointer is automatically warped. The Rake cursor uses a grid of cursors and eye tracking for cursor selection [10, 25]. These techniques benefit from the increase in input bandwidth provided by gaze tracking, but they cannot be implemented on standard computer hardware.

Adaptive [18] and adaptable [13] methods have also been considered: DirtyDesktops [18] creates magnetic fields around frequently-selected locations on the screen and UIMarks [13] lets users specify on-screen locations whose acquisition will be facilitated. However, adaptative techniques improve pointing only for frequently-selected targets and adaptable techniques require user intervention.

Edge and Displayless Pointing

HCI practitioners early noticed that targets on screen edges are easier to acquire because screen edges stop the cursor, effectively increasing W in the motor space. Edge pointing has been studied experimentally in [14, 1].

Edge pointing becomes problematic in multi-display environments: by default, desktop environments treat multiple displays as a single space, disabling edge pointing between them. Mouse Ether [5] takes into account the space between the displays as well as display size and resolution to compute a motor space – the *ether* – that lies between the displays. This re-enables edge pointing, since stopping the mouse in the ether warps the pointer to the closest display edge.

Mouse Ether is conceptually similar to our dead zones: they both add off-screen pointing space that (among other things) enable edge pointing. One problem with Mouse Ether is the absence of visual feedback when the cursor is in the ether. Several techniques have been proposed to visualize the location of off-screen objects: Halo [7] surrounds off-screen objects with rings large enough to reach the edge of the display, and Wedge [17] uses a triangle pointing towards the off-screen object. A recent study suggested that augmenting Mouse Ether with Halo helps, while also suggesting that Mouse Ether itself (with or without feedback) hurts performance when displays are sufficiently far apart [23].

Cursor Warping vs. Cursor Wrapping

Cursor warping refers to the sudden teleportation of the mouse cursor to a possibly distant place. It has been used to reduce pointing distance in some target-aware pointing techniques [2, 16] as well as in target-agnostic ones [28]. Manually-triggered cursor warping has also been used for rapidly switching between displays in multi-monitor environments [8]. However, it is also believed that sudden cursor jumps can be confusing to users and can slow them down [6].

Cursor wrapping should not be confused with cursor warping: wrapping the mouse cursor around screen edges involves a specific type of cursor warping, going from one edge of the screen to the opposite one. Several applications exist that support cursor wrapping. More than 15 years ago, the FVWM X Window Manager could be configured to en-

Figure 2. Wrapping dead zone (right) and expansion of targets located on the screen edge (left).

able it. Today, system-level tools provide the same feature[1]. But as discussed previously, none of these applications provide a dead zone or off-screen feedback. Moreover, to our knowledge, such techniques have never been evaluated.

THE TORUSDESKTOP TECHNIQUE

The TORUSDESKTOP extends direct cursor wrapping techniques with two additional features in order to make it usable and compatible with everyday desktop usage:

- a *wrapping dead zone* that adds a displayless pointing space around screen edges in order to help users anticipate cursor jumps and to re-enable edge pointing;
- a *wrapping feedback* that provides visual feedback on the cursor's location inside the dead zone to further increase user's control over cursor wrapping.

Wrapping Dead Zone

The *wrapping dead zone* is a displayless frame added around the screen edges. When the cursor reaches a screen edge, the user needs to cross this space before the cursor gets teleported to the opposite edge (Figure 2). This design presents three advantages:

Prevention of accidental triggering. In situations where users do not want to cross the screen, the wrapping dead zone prevents them from wrapping the cursor accidentally. Accidental wrapping can be distracting – especially repetitive wrapping when following a screen edge – and can slow users down since they have to bring the cursor back once they realize it has jumped. They may even lose the cursor altogether if they do not realize it has moved to the opposite side. The dead zone addresses this issue by making it more difficult to trigger the wrapping and allowing to cancel it.

Support for anticipation. In cases users want to cross the screen, the wrapping dead zone helps them anticipate the cursor jump and gives them time to switch their visual attention to the region where the cursor will re-appear. Additionally, it provides users with more flexibility, as they can adapt their mouse movement while crossing the dead zone to control where and when the cursor will re-appear.

Compatibility with edge pointing. As discussed previously, targets located on screen edges are faster to acquire, a feature that is now commonly used in window management systems and desktops (e.g., Mac OS' menu bars and MS Windows' task bar). While a naive implementation of the wrapping technique defeats edge pointing, using a large enough dead zone re-enables this feature as clicks within the dead zone are dispatched to the screen edge where the cursor comes from (Figure 2 left).

However, using a dead zone raises two issues. First, it increases the distance users have to cover during cursor wrapping so it may reduce the number of cases where the technique is useful. Second, it is not clear which dead zone sizes are small enough not to impede cursor wrapping, while being large enough to allow comfortable edge pointing. These questions will be later addressed in our experiment sections.

Wrapping Feedback

When crossing a dead zone, a standard mouse cursor would stop on the screen's edge and the user would have to blindly move a virtual cursor within the dead zone. It has been suggested that visual feedback about the position of an off-screen cursor helps pointing in displayless space [23], so we chose to augment the dead zone with visual feedback. Since there are many possible designs, we identified the three following requirements for TORUSDESKTOP visual feedback:

Position along the edge. The feedback needs to show where the cursor is located along the screen edge: for example if the exiting edge is vertical, users need to keep track of the cursor's y-coordinate to be able to predict where it will re-appear on the opposite edge.

Position within the dead zone. The feedback also needs to show the cursor's position in the orthogonal direction, i.e., how deep the cursor is in the dead zone. This is necessary for users to be able to predict when the cursor will be teleported and better anticipate its arrival. This also allows users to see how far they can go before the cursor jumps to prevent accidental triggering, especially during edge pointing.

Feedback mirroring. The two pieces of information above should be shown both near the edge where the cursor exits the screen and near the opposite edge. Thus, users can use the feedback whether they are focusing on the exiting side – i.e., when moving close to the edge or when doing edge pointing – or on the reentering side – i.e., when using cursor wrapping to point to a distant target.

We experimented with three feedback methods: *Halos*, *Arrow* and *Ghost*. Figure 3 explains these three techniques in detail: DZ is the dead zone size, d is the cursor's distance to the dead zone entrance and the constant k is Halo's intrusion distance. Gray arrows depict how cursor movements map to movements of visual feedback.

Halos. *Halo* [7] is a technique for providing on-screen feedback for off-screen objects, e.g., showing the location of points of interest in a map on a handheld device. It shows an arc of circle next to the screen edge; the circle is centered on the off-screen object in order to convey its direction and distance. In our case the off-screen object is the cursor itself, so when it enters the dead zone, we display a Halo both on the exit and on the entrance sides of the screen (Figure 3a). As in the original technique, the arcs stick out from the displayless space with a fixed intrusion distance k.

[1] E.g., `www.networkactiv.com/SoundyMouse.html`, `www.digicowsoftware.com/detail?_app=Wraparound`

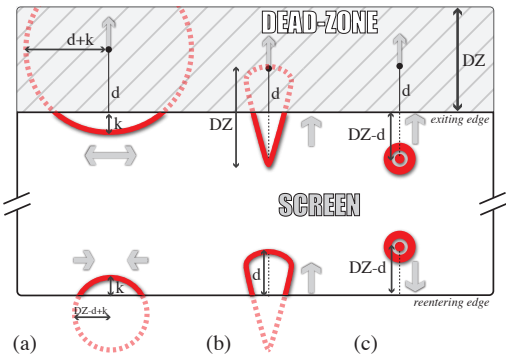


Figure 3. Halos, Arrow and Ghost TORUSDESKTOP feedback techniques for a top-to-bottom cursor wrapping.

Arrow. The Arrow feedback is inspired from a variant of Halo called *Wedge* [17]. Arrow's triangular shape is similar to Wedge's but unlike Wedge, its intrusion distance is not fixed and its shape does not change. Instead, it is a solid triangle of constant size, always perpendicular to the screen edge, that sticks out on both sides (Figure 3b). On the exit side, its flat end is attached to the cursor and its tail sticks out. On the entrance side, its tail is attached to the cursor and its flat end sticks out. The angle formed by the flat end conveys the cursor's distance in a way similar to Wedge.

Ghost. Finally, we propose a simpler visual feedback called *Ghost*, specifically designed for wrapping dead zones. Next to the dead zone's exit (bottom of Figure 3c), a circular shape is displayed whose distance to the edge is the same as the cursor's. In other terms, the edge acts like a mirror and the circular shape is like the cursor's reflection on that mirror. The same circular shape is displayed at the same distance near the dead zone's entrance.

Even if these wrapping feedback techniques fulfill the requirements we identified, it is not clear how much they help, if they help at all. We investigate this question in our study.

Corners

In a torus topology, the four screen corners are equivalent. So when the cursor reaches a corner, it is not clear where it should exit. Besides, the behavior of the cursor in the vicinity of a corner can be disturbing. For example, a cursor going to the top-right corner will re-appear either on the top-left or on the bottom-right. These two locations are nevertheless close to each other on the torus, so if the user approaches a corner with a 45-degree angle, the cursor will eventually appear on the opposite corner no matter which path it takes. However, the cursor will rapidly jump twice on the screen, which can be visually disturbing. To address these issues, we added four *corner zones* of 20 pixels each. When the cursor reaches one of these zones, it simply re-appears on the diagonally opposite corner after the dead zone crossing.

PRELIMINARY EXPERIMENTS

We conducted two preliminary experiments in order to refine the design of the technique before comparing it with direct pointing. The first experiment compares the feedback techniques and provides a first sense of the impact of the dead zone on movement time. The second one investigates the compatibility of TORUSDESKTOP with edge pointing.

Apparatus & Participants

The two experiments were conducted on a workstation running Mac OS X and with a 2560×1600 30" LCD monitor [2]. Such large displays are becoming more and more common and are likely to become a standard once their price drops. The TORUSDESKTOP software was implemented in Java. The mouse was a standard optical mouse with 500 dpi resolution and default system acceleration.

Eight unpaid volunteers, all male and right-handed, participated in the experiments. Participants were experienced mouse users with ages ranging from 24 to 31 (median 26.5). Each participant took about 60 minutes to complete each experiment after which they were given a short questionnaire.

Experiment 1: Feedback & Dead Zone

This experiment addresses the following questions:

- *Q1: Which wrapping feedback (including no feedback) is the best, with and without a dead zone?*
- *Q2: Does dead zone size affect movement time?*

Task & Design. A trial was a TORUSDESKTOP pointing task requiring subjects to cross either the left or the right edge of the screen. Subjects had to click on a *start target* at a distance DB1 to its closest edge and then acquire a *goal target* at a distance DB2 to the opposite edge by crossing the closest edge. Both start and goal targets were circles of 40 pixels. Targets were lying on the screen's horizontal centerline or placed above and below the centerline at a distance of 300 pixels, depending on the factor ALIGN (see Figure 4a). Task direction was either left to right (DIR = LR) or right to left (DIR = RL).

The experiment was a within-subject design with the main factors: (i) Feedback: FB = *None*, *Halos*, *Arrow*, *Ghost*; and (ii) Dead zone size: DZ = 0, 125, 250, 500.

Auxiliary factors were: (i) Distance of the start target to its closest edge, DB1 (Distance to Border 1) = 50, 150; (ii) Distance of the goal target to the opposite edge, DB2 (Distance to Border 2) = 50, 150, 300; (iii) ALIGN and DIR.

Concerning the values we chose for dead zone size, 0 is the baseline condition implemented in former cursor wrapping techniques. 125 and 250 seem to be realistic values for edge pointing [1]. We added 500 for completeness although we expect it to be too large to be used in practice. Note that for DZ = 0, the feedback condition is irrelevant and we only need to test the condition for feedback = *None*.

We grouped trials into blocks according to DZ × FB. We used 2 orders of presentation for DZ, increasing and decreasing, and counterbalanced the presentation order of FB. Before each DZ × FB condition, participants did one block of 24 practice trials then 2 blocks of measured trials. We hence collected 8 (PARTICIPANT) × 48 × (4(DZ)×1(FB=*None*) + 3(DZ=125,250,500)×3(FB)) = 4992 trials for analysis.

[2] Yielding 100.63 ppi and a pixel size of 0.025 cm.

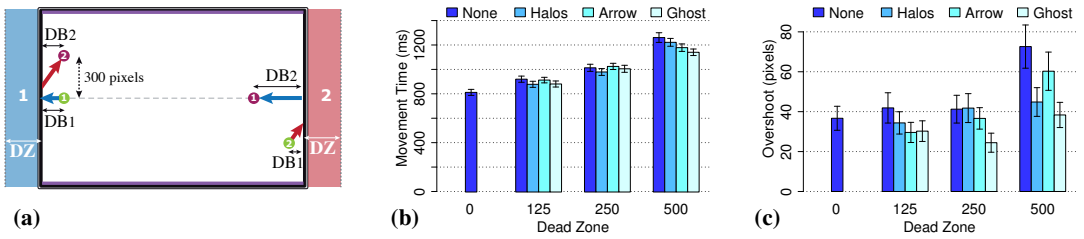


Figure 4. (a) Examples of target placement in experiment 1. Start targets are green, goal targets are red. Case 1: ALIGN = yes, DIR = RL, DZ on the left. Case 2: ALIGN = no, DIR = LR, DZ on the right. (b) MT per deadzone and feedback. (c) *OverShoot* per deadzone and feedback.

We collected three measures: (i) *MT*, the time from the click on the start target to a successful click on the goal target; (ii) *Error*, whether or not there was a click outside the target; and (iii) *OverShoot*, the distance in pixels of the furthest point reached by the pointer to the goal target.

Quantitative Results. We removed 0.75% outliers (trials with a *MT* that is 3 standard deviations apart from the mean *MT* within the condition) and duplicated the data for DZ = 0 for each FB in order to perform a full factorial analysis: FB × DZ × Random(PARTICIPANT) with *MT*, *Error* and *OverShoot*.

An analysis of variance reveals an effect of DZ on *MT* ($F_{3,21} = 80.0, p < 0.0001$). A Tukey post-hoc test shows a significant difference in means between all DZ, with *MT* increasing with DZ (Figure 4b). We observed no significant effect of FB on *MT*. However, we found a significant interaction FB × DZ ($F_{9,63} = 2.62, p = 0.0123$), which can be observed in Figure 4b: the difference between mean *MT*s for each FB value is the largest for DZ = 500. Indeed a post-hoc test shows no significant difference between the FBs for DZ $\leq$ 250, whereas for DZ = 500, *Ghost* is significantly faster than *Halos* and *None*, and *Arrow* is significantly faster than *None*.

We found an average error rate of 7.9%. An analysis of variance using a nominal logistic test for the model *Error* $\sim$ FB × DZ reveals no significant effect, error rates being very close for each FB × DZ (min 5.0%, max 9.9%).

For *OverShoot*, we found a significant effect of both FB ($F_{3,21} = 6.32, p = 0.0032$) and DZ ($F_{3,21} = 8.01, p = 0.0010$) (see Figure 4c). Post-hoc Tukey tests show that *Ghost* exhibits significantly less *OverShoot* than other feedback and that *OverShoot* is significantly larger for DZ = 500. However, there is a significant interaction FB × DZ ($F_{9,63} = 3.82, p = 0.0007$), which can be observed in Figure 4c: *OverShoot* is significantly lower for *Ghost* than for all other FB when DZ = 250. For DZ = 125, the only significant difference is between *Ghost* and *None*. For DZ = 500 we observe more *OverShoot* for *None* than for other feedback and less for *Ghost* than for *Arrow*.

Qualitative Results. In the post-experiment questionnaire, participants were asked to rank the feedback techniques globally and for each dead zone size. Among the eight participants, five globally ranked *Ghost* first, and each of the three other techniques was ranked first by one participant (*Ghost* was ranked second, third and last in these cases). Rankings by DZ are consistent with global ranking. Only three participants ranked *None* higher for DZ = 125.

Summary. Back to our first question *Q1*, *Ghost* seems to be the best choice for TORUSDESKTOP for all dead zone sizes: even if it does not exhibit a significantly better performance – except for the limit case DZ = 500 – it yields the smallest *OverShoot* and was preferred by participants. Concerning *Q2*, it is confirmed that *MT* increases with DZ.

Experiment 2: Edge Pointing

The questions this second experiment addresses are:

- *Q1: Does a dead zone help users performing edge-pointing tasks? If yes, is there an optimal dead zone size?*
- *Q2: Does wrapping feedback help or impede users during edge pointing?*

Stimuli & Design. A trial consisted in an edge pointing task where the subject had to click on a circular *start target* and then acquire a *goal target* on a screen edge. The goal target was located to the left or right end of the screen, and was vertically centered (Figure 5a). It had a width of 40 pixels and two possible heights H = 40, 125 – a size comparable to buttons on typical task bars and menu bars. Start targets were located on a 3 × 3 grid designed to cover several angles of approach. Their location was defined by DB = 200, 1200, 2200, their distance to the edge where the goal target was, and DH = 0, 600, −600, their distance to the horizontal centerline of the screen.

The experiment was a within-subject design with the same main factors as the first experiment: (i) Feedback: FB = *None*, *Ghost*; and (ii) Dead zone: DZ = 0, 125, 250, 500, inf. Given the findings of the first experiment, we only tested the *None* and *Ghost* feedback techniques in this experiment. We used the same dead zone sizes as in the first experiment and added an 'infinite' size (i.e., no cursor wrapping) as a baseline condition to test standard edge pointing.

Trials were grouped into blocks by DZ × FB. For each DZ × FB condition, participants started with one practice block of 2 × 2(H) × 3(DB) × 3(DH) = 36 trials, then proceeded with two measured blocks. Thus, for each participant, we collected 2 × 36 × (2(DZ=0,inf) + 2(FB) × 3(DZ=125,250,500)) = 576 trials for analysis.

In addition to *MT* and *Error* (defined as in previous experiment), we measured the dead zone distance effectively used *UseDistDZ* – i.e., the maximum horizontal travel distance inside the dead zone – and the number of times the cursor went past the dead zone *DZOverShoot*.

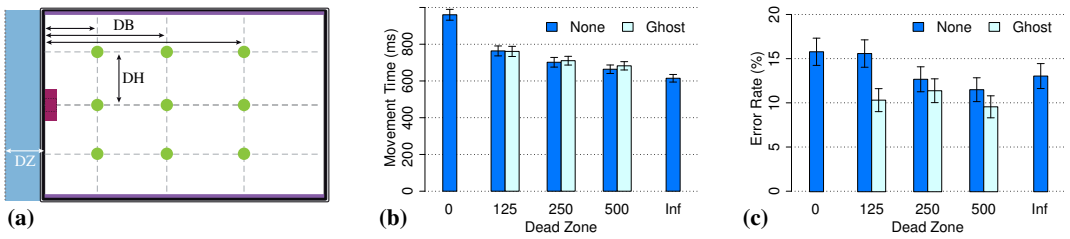


Figure 5. (a) Target placement in experiment 2. (b) MT per deadzone and feedback. (c) Error rate per deadzone and feedback.

Quantitative Results. We removed 0.97% outliers and duplicated the data for DZ = 0 and DZ = inf with the *Ghost* feedback to be able to perform a full factorial analysis FB × DZ × Random(PARTICIPANT).

An analysis of variance reveals an effect of DZ on *MT* ($F_{4,28} = 38.6, p < 0.0001$). As expected *MT* decreases as DZ increases (Figure 5b). A post-hoc Tukey test shows that (i) DZ = 0 is significantly slower than DZ ≥ 125; (ii) DZ = 125 is significantly slower than DZ ≥ 500; (iii) DZ = 250 is significantly slower than DZ = inf; and that (iv) difference is not significant for other DZ pairs. Indeed, we observe that the biggest improvement is from DZ = 0 to DZ = 125 (a 20% speed up).

As Figure 5b suggests, an analysis of variance reveals no effect of FB ($F_{1,7} = 0.42, p = 0.5463$) and no interaction FB × DZ ($F_{4,28} = 0.55, p = 0.6989$) on *MT*. Practical equivalence tests with a threshold of 20 ms (less than 3% of the grand mean) give positive results ($p \leq 0.02$), confirming there is no difference in MT between *None* and *Ghost*.

Regarding error, a nominal logistic ANOVA for the model FB × DZ ∼ *Error* (on the data set where $125 \leq$ DZ ≤ 500) shows a significant effect of FB ($\chi^2 = 6.12, p = 0.0134$) but no effect of DZ ($\chi^2 = 2.79, p = 0.2477$) and no interaction FB × DZ ($\chi^2 = 1.99, p = 0.3697$). Figure 5c shows that *Ghost* is less error-prone than *None*.

For *DZOverShoot*, the percentage of trials with accidental cursor wrapping is significantly higher without a dead zone (24.37% for DZ = 0 and less than 7% for DZ > 0). We noticed small differences between *Ghost* and *None* − *Ghost* always yielding less overshoots – but these are not significant.

Regarding *UseDistDZ*, i.e., the distance covered in the dead zone, we observed that the 90% quantile is close to half the dead zone size for all DZ < inf. It is close to 600 pixels for DZ = inf, a result consistent with previous studies [1].

Qualitative Results. After the experiment, participants were asked to tell (i) whether the *Ghost* feedback helped them select the target and (ii) whether they found the feedback distracting. Six participants out of eight agreed or strongly agreed that the feedback helped (one was neutral and one disagreed). However, half the participants agreed or strongly agreed that the feedback was also distracting.

Summary. Back to our first question *Q1*, this study confirms that when cursor wrapping is enabled, users are more efficient at selecting targets on the screen edges if a dead zone is provided. Not only a dead zone expands these targets ($W = 40 + DZ$), but it also prevents accidental cursor wrapping that can be time-costly to recover from. The high cost of accidental wrapping is confirmed by participants' conservative use of dead zones when doing edge pointing. As Figure 5b does not exhibit an asymptote for DZ < inf, the study does not suggest an optimal dead zone size. It however reveals that a small deadzone (125 pixels) is enough to reduce movement time by 20%.

Regarding *Q2*, we observe that the *Ghost* feedback does not impair performance but does not improve it either. However, it significantly reduces errors, suggesting that feedback makes users more accurate. Since in real systems pointing errors can have a high cost in terms of time and user frustration, this further confirms that *Ghost* feedback should be provided. Some users might however find the feedback distracting, as suggested by answers to our questionnaire.

COMPARING DIRECT POINTING & TORUSDESKTOP

In the two previous experiments, we validated and refined the design of TORUSDESKTOP by confirming the benefits of a wrapping dead zone and by identifying the best wrapping feedback technique. The goal of this third experiment is to evaluate TORUSDESKTOP by comparing it with conventional pointing (i.e., is it worth opening the backdoor?).

To this end, we presented subjects with various pointing tasks and had them either use direct pointing only (condition *Direct*) or use the backdoor only (condition *Wrapping*). The goal was to assess if TORUSDESKTOP can help, and when. But since in real settings deciding whether or not to use cursor wrapping may take time and/or yield suboptimal choices, we added a more realistic condition where it was up to the subject to go through the backdoor or not (condition *Torus*).

Figure 6b qualitatively illustrates our initial expectations. Using *Direct*, the further apart the start and the goal targets are, the higher the movement time. *Wrapping* is likely to show the opposite trend since the further apart the targets are, the closer they are on a torus topology (but note that Fitts' law cannot account for possible distracting effects of cursor wrapping). We hypothesized that performance under the *Torus* condition would roughly follow the minimum of *Direct* and *Wrapping*, plus a possible penalty due to choice.

Apparatus & Participants

The apparatus was the same as in the previous experiments. We recruited a total of 18 participants (5 female), all right-handed and experienced mouse users, with ages ranging from 23 to 35 (median 27.8). 14 of them participated in at least one of the previous experiments or pilot studies.

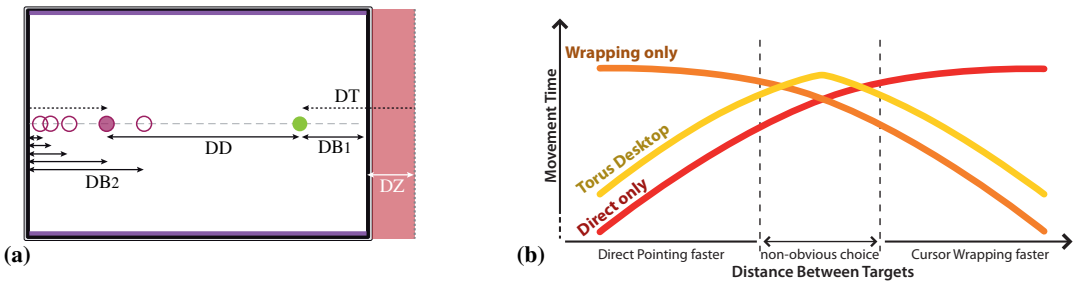


Figure 6. (a) Targets placement in experiment 3. The start target is green and potential goal targets are outlined in red. When the start target is acquired, the actual goal target appears with a solid color and other targets disappear. (b) Expected performance model. TORUSDESKTOP should behave as the most efficient technique according to the distance, though a penalty may be considered because of the choice between the techniques.

Stimuli & Design

Given the results of previous experiments, we used the *Ghost* feedback and a dead zone of 125 pixels for *Wrapping* and *Torus*. Recall this dead zone size yields a reasonable trade-off that meets the demands of both edge pointing and Torus pointing (i.e., neither of them is strongly penalized). As before, subjects had to click on a start target and acquire a goal target as fast as possible. Both targets were located on the horizontal centerline of the screen and were 40-pixel large[3].

At the beginning of a trial, all potential goal targets were shown. When the subject acquired the start target, the actual goal target appeared with a solid color and non-targets disappeared (see Figure 6a). This design was motivated by the inclusion of the *Torus* condition. In real settings, users might or might not know exactly where to click when they initiate a pointing movement. Our design is a trade-off between these two situations, since it reminds users of the possible target locations, but does not give them complete information about the task to prevent them from carefully deciding whether or not use the backdoor before the timing starts.

In addition to the pointing conditions TECH, the experiment included the factor DB1, the distance from the start target to the closest screen edge; and DB2, the distance from the goal target to the opposite edge. These two factors fully define the pointing tasks, whose direct pointing distance is DD = 2560 − (DB1+DB2), where 2560 is the screen width; and whose torus pointing distance is DT = DB1+DB2+125, where 125 is the size of the dead zone (Figure 6a). Both DB1 and DB2 values were $\{50, 125, 250, 500, 750\}$. We chose these values according to an extensive pilot study suggesting that among all possible pointing tasks defined by these DB1×DB2 pairs, 7 clearly favor *Wrapping*, 7 clearly favor *Direct* and the remaining 11 yield comparable performances.

The presentation order of the techniques was counterbalanced. Prior to the experiment, the *Direct* and *Wrapping* techniques were introduced to the participants with two short practice sessions. Then, the experiment was divided in two parts. First, participants performed 4 series of 25 trials per technique. A series of trials included all the possible combinations of DB1 and DB2 and was fully randomized. This part was exclusively a training session. Then, participants performed 1 series of practice trials followed by 5 series of measured trials per technique. Thus, for each participant, we collected a total of 3 (TECH) × 5 (repetition) × 5 (DB1) × 5 (DB2) = 375 trials for analysis.

We collected movement time (*MT*) and errors (*Error*) defined as in previous experiments. The experiment lasted about 45 minutes after which participants were given a short questionnaire and were interviewed about the strategies they developed in the *Torus* condition.

Quantitative Results

We removed 0.76% outliers defined as in previous experiments and performed a full factorial analysis with the model TECH×DD×Random(PARTICIPANT) and the finer model TECH×DB1×DB2×Random(PARTICIPANT). We found no learning effect and no significant difference in performance between the 14 subjects who were involved in preliminary experiments and the 4 new subjects.

Average Performance. The ANOVA reveals no effect of TECH on *MT* ($F_{2,34} = 0.245, p = 0.7836$ for the DD model and $F_{2,34} = 0.612, p = 0.5481$ for the DB1×DB2 model). Mean *MT* are close for *Direct* (1091 ms), *Wrapping* (1094 ms) and *Torus* (1108 ms). These similarities confirm that we chose well-balanced pairs of DB1×DB2 for *Wrapping* and *Direct*, but also suggest that the improvement promised by *Torus* may have been outweighted by the cost of choice. This will be discussed later.

We found a significant effect of TECH on *ErrorRate* ($F_{2,34} = 7.74, p = 0.0017$ for the DD model and $F_{2,34} = 7.40, p = 0.0021$ for the DB1×DB2 model). A post-hoc Tukey test shows that *Wrapping* and *Torus* are significantly less error-prone than *Direct*, with an error rate of about 6.6% for *Wrapping* and *Torus* versus 9.2% for *Direct*. A possible explanation is that participants were more careful with *Wrapping* and *Torus*, as they were less familiar with these techniques than with *Direct*.

Effect of Direct Distance. The ANOVA reveals a significant effect of DD ($F_{13,221} = 14.0, p < 0.0001$) and a significant interaction TECH×DD on *MT* ($F_{26,442} = 12.5, p < 0.0001$). We found no significant effect or interaction on *ErrorRate*.

Figure 7 shows *MT* as a function of DD for the three techniques. In accordance with our first intuitions, *Direct* gets slower as DD increases and *Wrapping* gets faster, although the

[3]Pilots studies did not show any effect of target size when comparing TORUSDESKTOP and direct pointing.

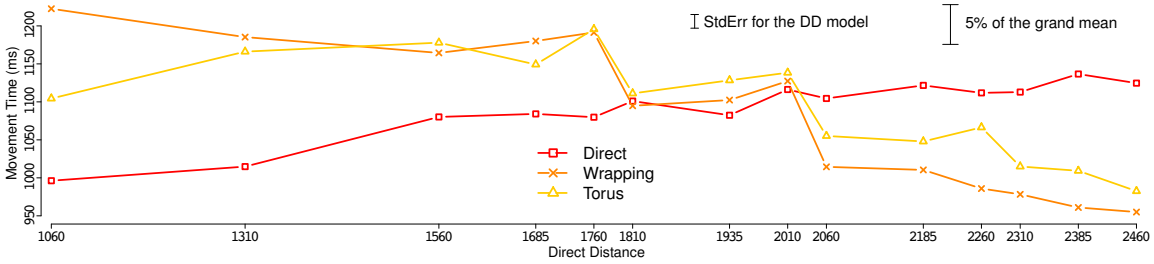


Figure 7. *Direct* **vs.** *Wrapping* **vs.** *Torus* **as a function of** DD**. Since the y-axis has its origin at 950, scale bars of error and grand mean are shown.**

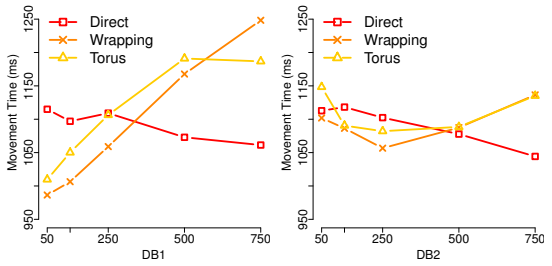


Figure 8. *MT* **as a function of** DB1 **(left) and** DB2 **(right) per** TECH**.**

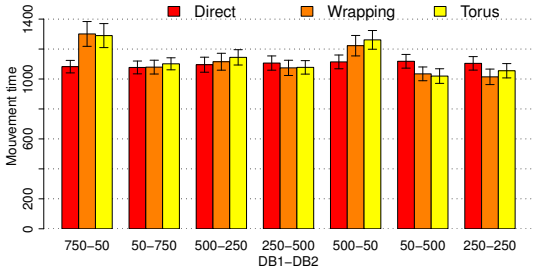


Figure 9. *MT* **as a fct. of certain critical** DB1-DB2 **couples per** TECH**.**

curve exhibits some irregularities (which will be explained when analyzing DB1 and DB2). The two techniques are comparable where the two curves cross, i.e., between DD=1810 and 2010 pixels. This corresponds to a Torus travel distance of only DT=675 to 875 pixels, suggesting that *Wrapping* is slower than what Fitts' law would have predicted. Taking trials where DD∼DT, we estimate this penalty to about $200\,ms$ [4]. This penalty is likely due to the difficulty in reacquiring the mouse cursor, but far from invalidating the whole approach, it merely increases the target distance above which *Wrapping* starts to be beneficial. Indeed, post-hoc tests show clear benefits for *Wrapping* above DD=2010 pixels, i.e., 80% the screen size in our experimental setup.

Figure 7 shows that the behavior of *Torus* is similar to *Wrapping* for DD>2010 pixels, where it exhibits a choice penalty of about $50\,ms$ but still clearly outperforms *Direct*. For DD=1810 to 2010 the 3 conditions exhibit similar performance. The left part of the curve is however less consistent with our initial expectations: for DD<1810, the performance with *Torus* is close to *Wrapping* instead of being close to *Direct* as in Figure 6b. One explanation is that participants failed to choose direct pointing when it was more efficient (our later experimental data confirms this). However, *Torus* also gets closer to *Direct* as DD decreases, which suggests that subjects might still favor *Direct* when it is clearly beneficial.

Effects of DB1 *and* DB2. We found significant effects of both DB1 ($F_{4,68} = 68.9$, $p < 0.0001$) and DB2 ($F_{4,68} = 3.68$, $p < 0.0090$) on *MT*. We also found significant interactions DB1×DB2 ($F_{16,272} = 6.29$, $p < 0.0001$), DB1×TECH ($F_{8,136} = 28.5$, $p < 0.0001$) and DB2×TECH ($F_{8,136} = 6.67$, $p < 0.0001$). We found no significant effect or interaction on *ErrorRate*.

[4] This value is consistent with a Fitts' law analysis we conducted on an extensive pilot study comparing *Wrapping* with *Direct*.

The interaction DB1×TECH can be observed in Figure 8 left. As DB1 increases, *MT* decreases for *Direct* (because DD decreases) and increases for *Wrapping* (because DT increases). For *Torus*, *MT* behaves like *Wrapping* up to DB1=750, suggesting *Direct* was preferred when the start target was very far from the edge. Ideally, *MT* should have followed *Direct*'s trend starting from DB1=500, but both the cost of the choice and the overuse of the backdoor seem to have prevented this.

Surprisingly, the interaction DB2×TECH is quite different (see Figure 8 right). For the same reasons as above, *MT* decreases with DB2 for *Direct*. But for *Wrapping*, *MT* follows a catenary curve with a minimum at DB2=250. Since *Wrapping* should normally increase with DB2, this suggests an issue with goal targets being very close to the edge. This issue also impacts the *Torus* condition, which exhibits the same minimum at DB2=250.

The asymmetric effects of DB1 and DB2 are further detailed in Figure 9, which shows *MT* by TECH for each DB1-DB2 pair that yields a DD value of 1760, 1810, 2010 or 2060. These values correspond to the irregularities we previously observed in Figure 7. Figure 9 confirms that *Wrapping* does poorly when DB2 is small. For example, *Wrapping* does much worse with the pair 750-50 than the pair 50-750, despite these pairs yielding the same DD=1760. This is the cause for the peak in Figure 7. This peak is followed by a sharp decline at DD=1810 which involves the more balanced pairs 500-250 and 250-500. *Torus* exhibits the same irregularities.

This asymmetry can be explained in the light of the optimized dual sub-movement model [22] and by considering *when* in the pointing movement cursor warping occurs. Using *Wrapping*, the mouse cursor first travels the distance DB1 + DZ, then warps and re-appears on the opposite side, after which it travels a distance DB2 before reaching the target. If DB2 is small compared to DB1 + DZ, the warping occurs at

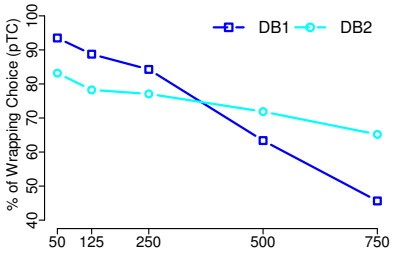


Figure 10. Percentage of *Wrapping* choice for DB1 and DB2.

the end of the movement – i.e., the corrective phase where visual feedback is the most crucial. Cursor warping requires attention shift, which likely disrupts the corrective process and slows users down. Conversely, if DB2 is large compared to DB1 + DZ, the warping happens during the initial ballistic phase of the movement where visual feedback is not used, thus its impact on performance is less severe.

Note that we could have ran a post-hoc analysis, but doing so with such a large number of data points is subject to methodological issues (high risks of type I or type II errors) and a correct analysis would have required a fair amount of space to justify and report. Since the significant interactions we found and the Figures 7, 8 and 9 are already quite informative we chose not to perform theses analyses.

Choice Strategies

So far, our results show that using the backdoor was beneficial when more than 80% of the screen had to be traveled. However, we observed mixed results when subjects had to make a choice, especially when direct pointing was the best choice. Therefore, we further analyze the choices made in the *Torus* condition. We consider the measure *pTC*, i.e., the % of the time where subjects chose wrapping, the model DD×Random(PARTICIPANT) and the finer model DB1×DB2×Random(PARTICIPANT) for this condition.

Unsurprisingly, DD has an effect on *pTC* ($F_{13,221} = 25.2$, $p < 0.0001$) and *pTC* increases with DD: the larger the distance to travel, the more often the backdoor was used.

DB1 and DB2 also have an effect on *pTC* ($F_{4,68} = 28.0$, $p < 0.0001$ and $F_{4,68} = 9.61$, $p < 0.0001$ respectively), with no DB1×DB2 interaction. As can be seen in Figure 10, the closer to the edges the targets were, the more often cursor wrapping was used. The dissimilar slopes further suggest that subjects gave more weight to the distance of the start target when they had to make a choice. This might be due to the fact that this information was available before DB2.

During the post-experiment interviews, participants reported using different strategies that can be summarized as:

- START: only wrap when the start target is close to the edge.
- GOAL: only wrap when the goal target is close to the edge.
- WRAP: always wrap the mouse cursor.
- DIRECT: always use direct pointing.
- NOCLUTCH: take the path that minimizes mouse clutching.
- RANDOM: choose more or less randomly.

10 participants reported relying mostly on START and 3 reported using it as a secondary strategy. 5 participants reported using WRAP as their primary strategy and 1 mentioned it as a secondary strategy. All the other strategies have been mentioned as a main strategy only once. GOAL was mentioned as a secondary strategy 3 times. Reported strategies were consistent with mean *pTC* per participant and with our analyses of the effects of DB1 and DB2 on *pTC*, except for two participants who reported using START and DIRECT but actually chose wrapping 93% and 76% of the time.

Overall, participants overused the backdoor: the global mean of *pTC* is 75% (std dev. 15%, median 74%). Even in the worst case scenario (DB1=DB2=750, DD=1060 and DT=1625), wrapping was used about 30% of the time. This trend could be partly due to a "good user" effect. It is also likely that participants were not accurate enough at estimating when cursor wrapping would beat direct pointing. It could be that with more training, users would develop a habit of the technique and start making close-to-optimal choices. But since we used an extensive training session and did not find a learning effect – even for subjects who were not involved in preliminary experiments – TORUSDESKTOP could probably benefit from visual clues that help users make optimal choices and develop more rational strategies.

Another question concerns the cognitive load associated with the choice. Although we did not measure cognitive load formally, we gave a post-experiment questionnaire where we asked subjects if they found it difficult to choose between direct pointing and wrapping. Out of 18, 4 strongly disagreed and 9 disagreed, suggesting cognitive load is moderate.

CONCLUSION AND FUTURE DIRECTIONS

Despite being an old idea, cursor wrapping is a simple and target-agnostic way of reducing target distance in pointing tasks. We discussed how such a technique should be designed and proposed the TORUSDESKTOP technique: it includes a dead zone that prevents accidental cursor warping and facilitates edge pointing, and a visual feedback that helps keeping track of the cursor inside the dead zone.

We tested several variations over this design and found that our *Ghost* off-screen feedback reduces overshoots during both edge pointing and cursor wrapping and is well-received by end-users, and that a 125-pixel dead zone (5% the screen size[5]) yields good performance for edge pointing while not sacrificing cursor wrapping performance. Recall the optimal dead zone size is infinite for edge pointing and is zero for cursor wrapping. However, a 125-pixel dead zone size is a reasonable trade-off where neither task is strongly penalized.

We also compared TORUSDESKTOP with direct pointing and uncovered the following potential sources of difficulties with the cursor wrapping approach:

- Cursor teleportation adds a time penalty of ∼200ms,
- Targets very close to the edges are harder to acquire,
- Chosing whether or not to use the backdoor has some cost.

[5] All figures are given according to our experimental setup that involves a 30" 2560x1600 display.

In our study, the cost of choice took the form of a small time penality (∼50ms) when cursor wrapping was the most beneficial, and of suboptimal choices (overuse of the backdoor) when direct pointing was the best option.

However, rather than invalidating the whole approach, these difficulties merely increase the travel distance above which TORUSDESKTOP starts being beneficial. Indeed, our study shows that cursor wrapping outperforms direct pointing above a travel distance of 2010 pixels (80% the screen size), and these benefits are preserved when users have to choose between direct pointing and cursor wrapping. These benefits can translate to much higher gains when one needs to regularly acquire targets close to an edge (e.g., toolbar buttons), or when going back-and-forth between two very distant targets (e.g., toolbars placed at opposite sides of the screen). However, despite extensive training, our study participants were not very accurate at estimating which technique will be the most efficient under a given condition. We are investigating how to augment TORUSDESKTOP with visual clues and feedforward techniques to help users make optimal choices and develop better strategies in the long run.

Further design is also required to support TORUSDESKTOP in multi-display environments. Several strategies can be considered such as disabling cursor wrapping on adjacent screen edges, restricting wrapping to the active screen or supporting on-demand wrapping/screen jump.

Finally, a field study of TORUSDESKTOP is clearly needed to validate the approach and ensure that cursor wrapping can be adopted and effectively used by end users in their everyday desktop usage. As a first step towards this goal, we implemented an application that enables TORUSDESKTOP at the system-level on Mac OS X and that is freely available at `http://insitu.lri.fr/TorusDesktop`.

REFERENCES

1. C. Appert, O. Chapuis, and M. Beaudouin-Lafon. Evaluation of pointing performance on screen edges. In *AVI '08*, 119–126. ACM, 2008.
2. T. Asano, E. Sharlin, Y. Kitamura, K. Takashima, and F. Kishino. Predictive interaction using the delphian desktop. In *UIST '05*, 133–141. ACM, 2005.
3. R. Balakrishnan. "Beating" Fitts' law: virtual enhancements for pointing facilitation. *IJHCS*, 61(6):857–874, 2004.
4. P. Baudisch, E. Cutrell, M. Czerwinski, D. Robbins, P. Tandler, B. Bederson, and A. Zierlinger. Drag-and-pop and drag-and-pick: techniques for accessing remote screen content on touch- and pen-operated systems. In *Interact '03*, 57–64. IOS, 2003.
5. P. Baudisch, E. Cutrell, K. Hinckley, and R. Gruen. Mouse ether: accelerating the acquisition of targets across multi-monitor displays. In *CHI '04 EA*, 1379–1382. ACM, 2004.
6. P. Baudisch, E. Cutrell, and G. Robertson. High-density cursor: a visualization technique that helps users keep track of fast-moving mouse cursors. In *Interact '03*, 236–243. IOS, 2003.
7. P. Baudisch and R. Rosenholtz. Halo: a technique for visualizing off-screen objects. In *CHI '03*, 481–488. ACM, 2003.
8. H. Benko and S. Feiner. Pointer warping in heterogeneous multi-monitor environments. In *GI '07*, 111–117. ACM, 2007.
9. R. Blanch, Y. Guiard, and M. Beaudouin-Lafon. Semantic pointing: improving target acquisition with control-display ratio adaptation. In *CHI '04*, 519–526. ACM, 2004.
10. R. Blanch and M. Ortega. Rake cursor: improving pointing performance with concurrent input channels. In *CHI '09*, 1415–1418. ACM, 2009.
11. G. Casiez, D. Vogel, R. Balakrishnan, and A. Cockburn. The impact of control-display gain on user performance in pointing tasks. *HCI*, 23(3):215–250, 2008.
12. O. Chapuis, J.-B. Labrune, and E. Pietriga. Dynaspot: speed-dependent area cursor. In *CHI '09*, 1391–1400. ACM, 2009.
13. O. Chapuis and N. Roussel. UIMarks: Quick graphical interaction with specific targets. In *UIST '10*, 173–182. ACM, 2010.
14. J. Farris, K. Jones, and B. Anders. Factors affecting the usefulness of impenetrable interface element borders. *Human factors*, 44(4):578, 2002.
15. T. Grossman and R. Balakrishnan. The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area. In *CHI '05*, 281–290. ACM, 2005.
16. Y. Guiard, R. Blanch, and M. Beaudouin-Lafon. Object pointing: a complement to bitmap pointing in guis. In *GI '04*, 9–16. CHCC Society, 2004.
17. S. Gustafson, P. Baudisch, C. Gutwin, and P. Irani. Wedge: clutter-free visualization of off-screen locations. In *CHI '08*, 787–796. ACM, 2008.
18. A. Hurst, J. Mankoff, A. Dey, and S. Hudson. Dirty desktops: using a patina of magnetic mouse dust to make common interactor targets easier to select. In *UIST '07*, 183–186. ACM, 2007.
19. M. Kobayashi and T. Igarashi. Ninja cursors: using multiple cursors to assist target acquisition on large screens. In *CHI '08*, 949–958. ACM, 2008.
20. S. MacKenzie. Fitts' law as a research and design tool in human-computer interaction. *HCI*, 7:91–139, 1992.
21. M. McGuffin and R. Balakrishnan. Fitts' law and expanding targets: experimental studies and designs for user interfaces. *ACM ToCHI*, 12(4):388–422, 2005.
22. D. Meyer, J. Smith, S. Kornblum, R. Abrams, and C. Wright. Optimality in human motor performance: Ideal control of rapid aimed movements. *Psych. Review*, 95(3):340–370, 1988.
23. M. Nacenta, R. Mandryk, and C. Gutwin. Targeting across displayless space. In *CHI '08*, 777–786. ACM, 2008.
24. G. Ramos, A. Cockburn, R. Balakrishnan, and M. Beaudouin-Lafon. Pointing lenses: Facilitating stylus input through visual- and motor-space magnification. In *CHI '07*, 757–766, 2007.
25. K.-J. Räihä and O. Spakov. Disambiguating Ninja cursors with eye gaze. In *CHI '09*, 1411–1414. ACM, 2009.
26. J. Wobbrock, J. Fogarty, S.-Y. Liu, S. Kimuro, and S. Harada. The angle mouse: target-agnostic dynamic gain adjustment based on angular deviation. In *CHI '09*, 1401–1410. ACM, 2009.
27. A. Worden, N. Walker, K. Bharat, and S. Hudson. Making computers easier for older adults to use: area cursors and sticky icons. In *CHI '97*, 266–271. ACM, 1997.
28. S. Zhai, C. Morimoto, and S. Ihde. Manual and gaze input cascaded (MAGIC) pointing. In *CHI '99*, 246–253. ACM, 1999.

Rapid Development of User Interfaces on Cluster-Driven Wall Displays with jBricks

Emmanuel Pietriga[1,2] **Stéphane Huot**[2,1] **Mathieu Nancel**[2,1] **Romain Primet**[1]

[1]INRIA
F-91405 Orsay, France

[2]LRI - Univ Paris-Sud & CNRS
F-91405 Orsay, France

ABSTRACT
Research on cluster-driven wall displays has mostly focused on techniques for parallel rendering of complex 3D models. There has been comparatively little research effort dedicated to other types of graphics and to the software engineering issues that arise when prototyping novel interaction techniques or developing full-featured applications for such displays. We present jBricks, a Java toolkit that integrates a high-quality 2D graphics rendering engine and a versatile input configuration module into a coherent framework, enabling the exploratory prototyping of interaction techniques and rapid development of post-WIMP applications running on cluster-driven interactive visualization platforms.

Keywords
Wall Displays, Clusters, Interaction, Toolkit, Prototyping

INTRODUCTION

Over the last decade, wall-sized displays have evolved from experimental, CRT monitor-based setups to sophisticated arrays of tiled projectors or LCD panels. The latter are often called *ultra-high-resolution* displays to emphasize their significantly higher display capacity compared to projector-based *very-high-resolution* displays. They typically accommodate several hundred megapixels, and are driven by clusters of computers. As an example, the setup depicted in Figure 1 uses 32+1 graphic processing units in 16+1 computers to display $20480 \times 6400 \simeq 131$ megapixels on a $5.5m \times 1.8m$ surface ($\simeq 100dpi$). These displays enable the visualization of truly massive datasets. They can represent the data with a high level of detail while retaining context [16], and enable the juxtaposition of data in various forms. To make them interactive, wall-sized displays are increasingly coupled with advanced input devices, e.g., motion-tracking systems, wireless multitouch devices, in order to enable multi-device and/or multi-user interaction with the displayed data [16, 17]. These interactive ultra-high-resolution displays can be used in many application domains, including command and control centers, geospatial imagery, scientific visualization, collaborative design and public information displays.

E. Pietriga, S. Huot, M. Nancel, R. Primet.
Rapid Development of User Interface
on Cluster-Driven Wall Displays with jBricks
In EICS '11: Proceedings of the 2nd ACM SIGCHI symposium on
Engineering interactive computing systems, ACM, June 2011.
Authors Version

These new environments pose new research challenges. From a *computer graphics perspective*: how to render complex graphics at high frame rates, taking advantage of the cluster's computing and rendering power. From a *human-computer interaction perspective*: how to design effective visualizations that take advantage of the specific characteristics of large, ultra-high-resolution surfaces; how to design interaction techniques that are well-adapted to this particular context of use, and how to handle the multiple and heterogeneous input devices and modalities typically used in this context. Finally, from a *software engineering perspective*: how to enable the rapid prototyping, development, testing and debugging of interactive applications running on clusters of computers, providing the right abstractions.

In this paper, we focus on the latter research question, that we consider essential to foster more research and development from the HCI perspective. We present jBricks, a Java toolkit for the development of post-WIMP applications executed on cluster-driven wall displays, that extends and integrates a high-quality 2D graphics rendering engine and a versatile input management module into a coherent framework hiding low-level details from the develeoper. The goal of this framework is to ease the development, testing and debugging of interactive visualization applications. It also offers an environment for the rapid prototyping of novel interaction techniques and their evaluation through controlled experiments, such as the one we recently conducted about mid-air pan-and-zoom techniques for wall-sized displays [16].

Background and Motivation

The parallel-rendering techniques developed over the last ten years enable the efficient display of 3D graphics on tiled displays driven by clusters of computers. This is usually done by sending already rendered images to the cluster nodes, or by sending geometry and performing compositing operations to produce the final wall-sized image. Different techniques exist, including *sort-first* and *sort-last* pipelines as well as various hybrid solutions. Well-known frameworks include Chromium [11], Equalizer [10] and SAGE [13]. See Ni *et al.* [17] for a comprehensive survey.

However, not all wall display applications use 3D graphics. With the introduction of ultra-high resolution, high-quality 2D graphics open wall-sized displays to new applications, e.g., in astronomy, geospatial intelligence and visual analytics at large, to give a few examples. These applications essentially combine very large bitmap images, high-quality text and 2D vector graphics, e.g., satellite imagery aug-

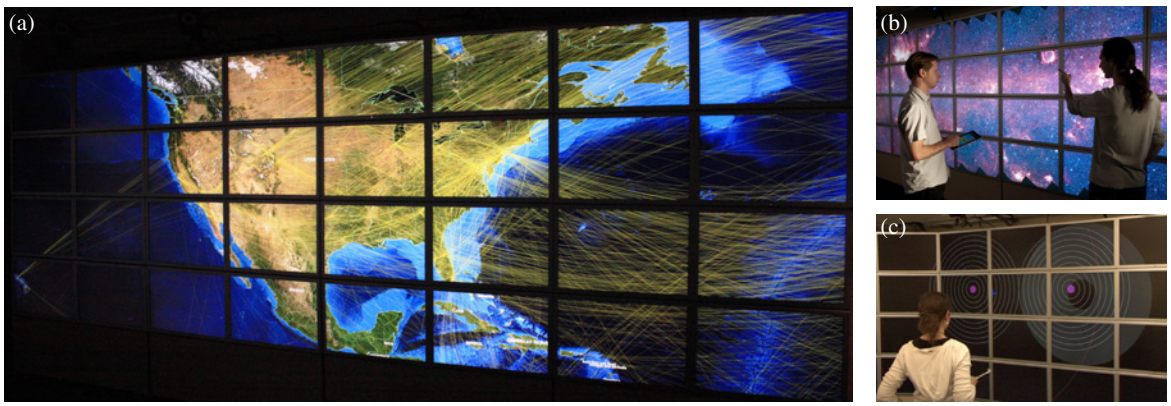


Figure 1. jBricks applications running on the WILD platform (32 tiles for a total resolution of 20 480 × 6 400 pixels). (a) Zoomed-in visualization of the North-American part of the world-wide air traffic network (1 200 airports, 5 700 connections) overlaid on NASA's Blue Marble Next Generation images (86 400 × 43 200 pixels) augmented with country borders ESRI shapefiles. (b) Panning and zooming in Spitzer's Infrared Milky Way (396 032 × 12 000 pixels). (c) Controlled laboratory experiment for the evaluation of mid-air multi-scale navigation techniques [17].

mented with data layers, or information visualization techniques for the display of large datasets, e.g., for the visual exploration of large networks (Figure 1-a). However, there is currently no good solution for the distributed rendering of high-quality 2D graphics on cluster-driven wall displays.

Low-level 3D graphics APIs such as OpenGL are currently the main solution for developing cluster-driven visualizations. They work well for the high-performance visualization of textured 3D scenes, but are ill-suited to programming high-quality 2D graphics interfaces, lacking appropriate support for the management and efficient rendering of text, line styles, arbitrary 2D shapes and WIMP widgets. This was already observed for desktop application programming [6], and remains true for cluster-driven wall-displays. Pixel streaming approaches à la SAGE work well when combining different windows of relatively limited size from different applications, potentially running on different machines. They would however not work for full-screen, highly-dynamic visualizations on ultra-high-resolution displays: updating hundreds of megapixels forming a single coherent image at an interactive refresh rate would require significantly more network bandwidth than is commonly available and would put an extremely heavy load on the node in charge of rendering the image to be streamed.

Rich interactive 2D desktop applications, usually termed post-WIMP applications, are typically developed with structured graphics toolkits [2, 7, 12, 18] that provide useful abstractions on top of low-level APIs. They enable rapid prototyping and development of advanced interactive visualizations. Our goal is to offer a structured graphics toolkit capable of running transparently on cluster-driven wall displays and capable of handling a wide range of input devices and modalities. From a graphics perspective, this requires hiding the complexity entailed by having to distribute rendering on multiple computers. While our focus is on expressiveness and ease-of-use, we also pay attention to scalability issues, adapting ideas originally developed for efficient distributed 3D rendering to our context, such as the use of a multicast protocol to transmit updates to cluster nodes, and a culling algorithm adapted to zoomable user interfaces. From an input management perspective, this requires going beyond the basic redirection mechanisms found in existing distributed rendering frameworks that only support conventional input devices, i.e., mouse and keyboard operated from the master computer. For now, support for other devices is mostly achieved *via* ad hoc solutions (drivers or libraries) that are strongly integrated and statically linked within applications. This approach is not generic and flexible enough when exploring and prototyping novel interaction techniques [9]. An alternative approach consists in providing high-level abstractions of input modalities that enable association and runtime substitution of devices. It has proven successful in other domains, including physical ubiquitous computing [5], virtual reality (Gadgeteer for VR Juggler [8]) and in the more general context of post-WIMP applications (ICon [9], Squidy [14]), and we adapt it to interactive wall displays.

jBricks FRAMEWORK ARCHITECTURE

The framework is essentially composed of two independent modules: one for managing all graphical operations, and one for handling input. The two modules are loosely coupled. They communicate via a dynamic plugin architecture and network sockets using high-level protocols such as OSC. This makes the framework highly flexible: modules can be instantiated multiple times and can run on different nodes.

Structured Graphics

Our goal is to provide an API and feature-set similar to those of desktop structured graphics toolkits [2, 7, 12, 18] while i) hiding the complexity entailed by distributed rendering, ii) promoting ease of learning and ease of use, and iii) enabling code reuse: visualization components initially developed for desktop computers should run on cluster-driven wall displays with minimal changes to the original application code. With these high-level objectives in mind, we chose to extend an existing structured graphics toolkit rather than start developing a new one from scratch.

We used the open-source ZVTM toolkit [18], that supports most Java2D drawing primitives but offers higher-level ab-

stractions that ease the management and manipulation of graphical objects: rendering is handled in retained mode, meaning that the toolkit retains a complete model of the objects to be rendered. ZVTM follows a monolithic approach, as opposed to a polylithic one[1]. Experience has shown that monolithic approaches are conceptually easier to handle by developers, generate less lines of code and require managing a smaller number of objects [7]; properties that we consider of high importance for rapid UI development.

Featured types of graphical objects include polygons of arbitrary shape, splines, Swing widgets, bitmap images and high-quality text, with support for advanced stroke and fill patterns. Those objects (*Glyphs*) are placed on infinite drawing surfaces (*Virtual Spaces*) that are observed through one or more *Cameras*. A camera renders the objects that lie in its viewing frustum in a *View*, that corresponds to a window on the screen. The toolkit makes it easy to create zoomable user interfaces (cameras can be smoothly panned and zoomed). It supports multiple independent views, as well as *Portals* (views within views) [6], multiple layers within a view (each corresponding to a different camera), as well as a variety of built-in focus+context visualization techniques. Cameras and glyphs can be animated using various pacing functions.

Cluster-based Structured Graphics Rendering

jBricks' extension of ZVTM to render graphics on cluster-driven tiled displays is conceptually straightforward. It takes an approach similar to what *sort-first* algorithms do for parallel rendering of 3D graphics in retained mode: as ZVTM already enables multiple cameras to observe a given virtual space, implementing tiled rendering basically consists in sharing that virtual space between all cluster nodes and setting one camera per display tile. Each camera's viewing frustum is configured so that their juxtaposition forms an overall coherent image from the user's perspective, according to the physical layout of display tiles.

Distributed Virtual Spaces. jBricks adopts a client-server model [17]: as shown in Figure 2, a single instance of the application runs on a *client node*, generating the geometry (populating virtual spaces with glyphs) and distributing it to *render servers* running on *cluster nodes*. Virtual spaces and glyphs contained therein are broadcast to all cluster nodes. They are replicated and kept synchronized as glyphs are added, removed, or have their properties changed. Parallel rendering frameworks for 3D graphics have mainly focused on the visualization of static-geometry models where only the camera(s) are manipulated interactively. The applications that jBricks aims to support typically manage much more dynamic objects, both in terms of geometry and visual appearance (color, stroke, font, etc.), potentially requiring a lot of network bandwidth. Multicast communication can greatly decrease bandwidth requirements for those updates [15]. We use JGroups (`http://www.jgroups.org`) as our group communication layer, that provides reliable messaging over IP multicast. Over this layer, we exchange atomic changes called `Delta`, which are serialized Java objects representing a new value for a given glyph attribute, propagated to the corresponding glyphs on render servers.

[1] *Monolithic* toolkits primarily use compile time inheritance to extend functionality, while *polylithic* toolkits primarily use run-time composition to do so, typically using a scene graph [7].

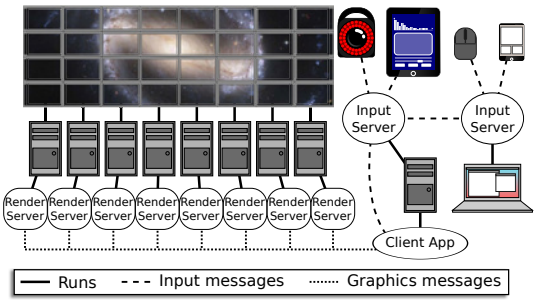


Figure 2. Example jBricks configuration: wall's graphics client and input server for motion tracker and tablet run on client node; input server for mouse, keyboard and smartphone run on user's laptop.

Performance. As noted by Bederson and Meyer [6] about zoomable user interfaces: *"Smooth real-time interaction is crucial. If the system becomes slow and jerky, the metaphor dies"*. The use of a multicast protocol for updating glyphs enables us to smoothly animate several hundred property changes simultaneously and independently of the number of render servers. Camera animations do not require significant bandwidth, as moving a camera only requires updating a maximum of three double-precision floating point values per frame. A more serious bottleneck when panning and zooming is the frame rate achieved by render servers. ZVTM already implements efficient culling algorithms for zoomable user interfaces. Glyphs get projected and rendered for a given camera only if they lie in that camera's viewing frustum. jBricks benefits from this directly: each server renders only the glyphs that will eventually be visible in the associated tile, which significantly decreases the computational and rendering load for scenes with high object counts.

Preliminary tests have shown that visualizations containing up to 200,000 objects could be rendered at interactive frame rates on the platform depicted in Figure 1. jBricks also benefits from Java2D's OpenGL pipeline, and from ZVTM's spatial indexing and dynamic external resource (un-)loading mechanisms. These were developed to support multi-scale navigation in very large datasets, such as gigapixel bitmap images decomposed recursively as a region quadtree. We adapted these mechanisms in jBricks to work in a distributed context, enabling the interactive visualization of very large images. Example images that have been visualized include the 26 gigapixel panorama of Paris (354 048 75 520 pixels) and Spitzer's Infrared Milky Way (Figure 1-b), that can be freely panned and zoomed on a wall display.

Programming. jBricks adds cluster support to ZVTM by monkey-patching the original toolkit using AspectJ, without altering its source code. This makes the cluster extension module small (3 000 lines of code vs. 39 000 for ZVTM) and facilitates forward compatibility. This also keeps API changes to a minimum: virtual spaces, glyphs, animations and most other constructs are managed through the original ZVTM API; low-level mechanisms for distribution to render servers are hidden from the developer. Only

cameras and views get created and managed in a slightly different manner. The tiled display's geometry has to be declared: number of rows and columns, size of each screen (pixels), options such as whether to paint pixels behind the bezels separating the tiles (overlay approach) or ignore them (offset approach). *Clustered Views* replace regular ZVTM views: a clustered view is divided into blocks, each block corresponding to a tile and render server. Render servers can be instantiated multiple times on a single node if that node drives multiple tiles. ZVTM-based desktop applications, originally written to run on single hosts, can be adapted to run on a cluster-driven large displays by changing as little as four lines of code. Render servers are instances of a generic display program that is part of jBricks, meaning that developers only have to modify the client application and do not have to run application-specific code on cluster nodes. This enables a quick development and deployment lifecycle. It is also interesting to note that the client application and render servers can run anywhere, including on the same computer, which facilitates development outside the cluster platform.

Advanced Input & Interaction

Wall-sized displays are often augmented with a complex interactive environment, made of heterogeneous input modalities ranging from actual input devices (e.g., mouse, 6-DOF devices, tablets), to the output of interactive systems used for input (e.g., motion-tracking system software, multi-touch table tracker, mobile device sensors interpreter). jBricks's cluster extension to ZVTM handles all aspects related to graphics distribution and rendering, but supports little beyond basic input redirection for conventional devices. An input management system is required to handle the multiple input channels and to ease their fusion so as to eventually deliver high-level input events to applications, that make the description of complex interaction techniques easier [12].

We identified three main requirements for such an input management system. The system should be able to handle various kinds of distributed input in a *generic* way to allow easy substitution of input modalities, and should provide generic output to several distributed applications, no matter whether they were specifically developed for this platform or not. The system should be *extensible*, making it easy to support new devices and functionalities with re-usable processing functions or interaction techniques. Finally, the system should be *adaptable*, enabling runtime addition of new devices and changes to the input configuration.

With these objectives in mind, we developed the jBricks Input Server (jBIS), the distributed input and interaction management system of jBricks. jBIS is built on top of the FlowStates toolkit [3], that combines the ICon [9] and SwingStates [2] libraries. ICon's dataflow model can handle multiple devices and describe advanced interactions efficiently [12]. Its visual editor makes it simple to connect them to application input endpoints (Figure 3). SwingStates extends the Java language with state machines and provides a simple yet powerful programming language that simplifies the description of interaction logics on the application side. FlowStates integrates these two models seamlessly: state machines are instantiated as dataflow processing devices that can be graphically connected to input devices or to other state machines in the dataflow configuration.

Input handling. Thanks to the ICon library, the jBricks Input Server has built-in support for various regular and advanced input devices: mouse, keyboard, various tablets, Nintendo Wii remotes, VICON motion-trackers, interactive pens, etc. These input devices are instantiated as dataflow processing devices that can be connected to adapters or application devices through the dataflow editor (see the mouse device in Figure 3). These dataflow components are high-level structured representations of input devices (or classes of input devices) with typed output slots mapped to the various channels of the input device they handle.

We extended ICon to support generic devices through various protocols with specific dataflow devices that can receive and send OSC, Ivy or TUIO messages. This approach provides an implicit way of performing automatic device registration thanks to the addressing mechanism of these protocols: each input source that sends a message addressed to a specific receiving device in a running configuration is implicitly considered. For instance, a jBIS' OSC receiver device can listen to messages addressed to `/jBIS/position` with two arguments, x and y. This device will then externalize the corresponding output slots. These will be updated each time that a new `/jBIS/position` message is received, wherever it comes from: a smartphone running an application that sends OSC messages from touchscreen events, the tracking software of an interactive table, mouse movements from a laptop running another instance of the jBIS, etc.

Interaction configuration. Input configuration and the lower-level description of interaction techniques (typically the connection to inputs) get specified in jBIS with an ICon dataflow configuration. ICon provides an extensive library of adapter devices, e.g., math or logic operators, control structures, flow control. These can be used to manipulate and transform the raw values of input channels into higher-level data structures (e.g., the mult device in Figure 3). The jBIS built-in library also extends the basic processing devices of ICon with platform-specific ones, adapted to interactive wall-sized displays: for instance, the *pointed tile* dataflow component returns the display tile that is intersected by a 3D vector received as input (typically modeling the user's arm). More than simple low-level processing components, these higher-level devices are close to the re-usable interaction techniques of [12], offering several levels of granularity to the user when building an input configuration.

The jBricks Input Server also includes a plug-in mechanism for the creation of custom dataflow devices with FlowStates [3]: state machines are instantiated as dataflow components, and their transitions are triggered by the connected inputs (pointing and pan-zoom in Figure 3). Programmers can use this descriptive and straightforward approach to extend the jBIS library and to describe some parts of the interaction logic of an application, or even more generic libraries that can be used with multiple applications running on the platform.

Link with application/visualization software. In jBricks, the higher-level interaction logic (manipulation of objects, graphical feedback) is encoded in the client application (Figure 2) developed with ZVTM. The link between the jBIS and this application can be established in two ways. The first solution consists in using specific dataflow devices in the input configuration to deliver high-level interaction events to the application through a networking protocol such as OSC; the client application interprets these messages and reacts accordingly. The other solution consists in using the plug-in mechanism of jBIS to implement application-specific devices that will be instantiated as endpoints of the dataflow. These plugins can define their own protocol to communicate with the client application, or even encapsulate it, enabling direct communication as the client node is running in the same process (same Java Virtual Machine) than jBIS.

Finally, jBIS can be controlled remotely, so that applications can trigger commands (start/stop/change the input configuration) or dynamically install a plugin. Several jBIS instances can run simultaneously, communicating through networking dataflow devices (Figure 2). This modularization, based on the description of partial input configurations, reinforces the flexibility and adaptability of the platform as partial configurations can easily be substituted.

The architecture of jBricks and the resulting development and configuration tools make it possible to develop applications outside the platform, i.e., on a simple laptop, and then deploy and run them on an actual cluster-driven wall display. On the graphics side, changes to the client application are minimal (four lines of code) and can easily be managed using, e.g., command line options or Maven profiles. On the interaction side, the jBricks Input Server makes it easy to dynamically reconfigure and adjust inputs according to available devices and modalities. In the following section, we illustrate these principles with a short scenario showing how jBricks can be used for the prototyping and implementation of interaction techniques for a controlled experiment on a wall-sized display.

jBricks IN ACTION

Abelard and Eloïse need to prepare an experiment to compare one-handed mid-air interaction techniques for selection of very small targets on wall-sized displays. They consider two techniques: a very precise bi-modal pointing technique, and a cursor-centered pan & zoom technique.

They first describe the two techniques with state machines (Figure 3) and plan to implement and configure them as follows. The pointing technique will be operated with a gyroscopic mouse and will feature a coarse mode – i.e., ray-casting – and a precise mode – i.e., relative pointing with a low CD gain. Precise mode will be triggered using the right mouse button; target selection using the left button (Figure 3-a). The pan & zoom technique is operated with an iPod Touch. Vertical thumb movements control the zoom factor, ray-casting of the user's arm controls the cursor's position. Two small areas at the bottom of the iPod's screen trigger panning and target selection, respectively (Figure 3-b).

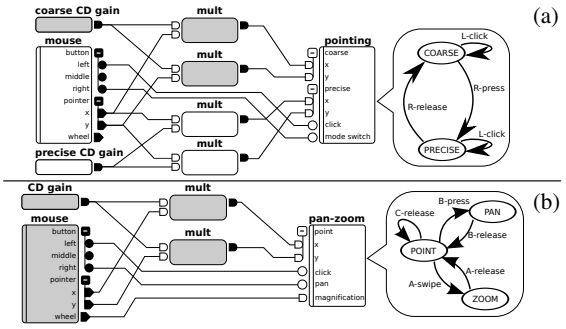


Figure 3. jBIS configurations of the pointing (a) and pan & zoom (b) techniques and their corresponding state machines. A mouse is used to control the techniques and simulate unavailable devices.

Prototyping

As jBricks' graphics and input modules are loosely-coupled, Abelard can work on the experiment's graphics while Eloïse implements and configures the two interaction techniques.

Abelard is working on the graphics part of the experiment. Using ZVTM, he creates an application that displays the targets, cursor appearance and textual instructions on his personal computer without having to worry about the specifics of the cluster-based wall display environment. He just needs to consider the actual dimensions of his graphical scene (in this case, a 20000 7000 pixel area). To make the entire scene visible on his screen, he sets the zoom factor higher than it will eventually be in the real experiment (a straightforward operation in a zoomable user interface).

Meanwhile, Eloïse implements each technique as a FlowStates state machine and encapsulates them in a jBricks Input Server plugin, making them available as dataflow processing devices. During this early prototyping stage, Eloïse focuses on developing the interaction logic, using a basic version of the graphics interface provided by Abelard. She does not need to work on the actual hardware platform either. She runs jBIS on her laptop and uses a regular mouse to simulate the actual input devices that will be used eventually (motion-capture system, gyroscopic mouse, iPod Touch). In this testing configuration, ray-casting with the motion-capture system and gyroscopic mouse are replaced by mouse coordinates; the mouse wheel and buttons are used in lieu of touch events. The output ports of the mouse device are connected to the technique devices, pan-zoom and pointing (Figure 3), the two modes of the pointing technique being simulated by applying constant multipliers to the mouse coordinates (the mul and CD gain processing devices). Later, these configurations will be slightly modified to handle the actual input devices to be used in the experiment.

Porting to the Wall Display Hardware Platform

On the input side, Eloïse substitutes the devices used for prototyping on her laptop with the platform's actual devices, as shown in Figure 4. The regular mouse can be directly substituted with the gyroscopic mouse, with only a CD gain adjustment (changing the value of precise CD gain, Figure 4-a). jBIS has built-in support for the 10-camera motion tracking sys-

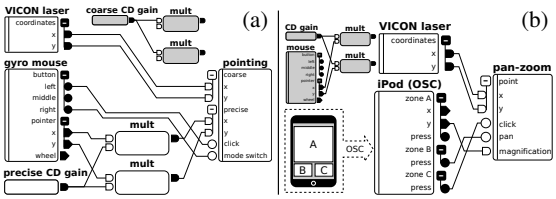


Figure 4. jBIS configurations of the final pointing (a) and pan & zoom (b) techniques. The simulation inputs (in grey) can be reused at any time simply by changing the connections.

tem in the room (the VICON laser device). For the iPod Touch, Eloïse uses a built-in OSC receiver device in her input configuration to receive touch events from a freely-available application running on the handheld (Figure 4-b). To deploy the client application on the actual hardware, Abelard only needs to add a few jBricks instructions describing the *Clustered View*. He then embeds the application into the jBIS plugin made by Eloïse. The client application is launched by jBIS; it has access to the state machines' output and will react according to the chosen interaction technique.

Further iterations, switching back and forth between the simplified configuration running on personal computers and the one for the actual wall display hardware is straightforward. Abelard and Eloïse can also easily add new techniques by implementing new state machines and test several input configurations for each of them.

CONCLUSION

The jBricks framework extends and integrates state-of-the-art structured graphics and input management toolkits to enable the rapid development of post-WIMP applications for cluster-based wall displays equipped with advanced input devices and modalities. Its architecture and features enable easy deployment and reconfiguration, allowing developers to partially implement and debug their applications on conventional hardware such as a single laptop or workstation.

We have successfully used jBricks for the rapid prototyping of novel interaction and visualization techniques, and to run controlled experiments for their evaluation [16]. It is also used for the development of various applications for the visualization of large datasets in other disciplines: astrophysics, social network analysis, geospatial intelligence. The Java-based platform makes it easy to use existing libraries in client applications. In addition, ZVTM features several extension modules that enable, e.g., the layout of large networks, the visualization of treemaps, native high-quality PDF rendering, FITS astronomy image display, interactive navigation in OpenStreetMap, from world overview down to street level. Future work will focus on improving the Java2D/OpenGL rendering pipeline by optimizing the stream of instructions. The implementation of a higher-level communication protocol, based on HID definitions on top of OSC, will improve dynamic input device registration and configuration. jBricks will be made available under an open-source software license (`http://insitu.lri.fr/JBricks`).

ACKNOWLEDGEMENTS

We wish to thank Caroline Appert and Olivier Chapuis for helpful comments on early drafts of this paper. This work is supported by a Région Île-de- France / Digiteo grant.

REFERENCES

1. C. Andrews, A. Endert, and C. North. Space to think: large high-resolution displays for sensemaking. In *Proc. CHI '10*, 55–64. ACM, 2010.
2. C. Appert and M. Beaudouin-Lafon. SwingStates: Adding state machines to Java and the Swing toolkit. *SP&E*, 38(11):1149 – 1182, 2008.
3. C. Appert, S. Huot, P. Dragicevic, and M. Beaudouin-Lafon. FlowStates: Prototypage d'applications interactives avec des flots de données et des machines à états. In *Proc. IHM '09*, 119–128. ACM, 2009.
4. R. Ball, C. North, and D. Bowman. Move to improve: promoting physical navigation to increase user performance with large displays. In *Proc. CHI '07*, 191–200. ACM, 2007.
5. R. Ballagas, M. Ringel, M. Stone, and J. Borchers. istuff: a physical user interface toolkit for ubiquitous computing environments. In *Proc. CHI '03*, 537–544. ACM, 2003.
6. B. Bederson and J. Meyer. Implementing a zooming user interface: experience building pad++. *SP&E*, 28:1101–1135, August 1998.
7. B. B. Bederson, J. Grosjean, and J. Meyer. Toolkit Design for Interactive Structured Graphics. *IEEE Trans. Software Eng.*, 30(8):535–546, 2004.
8. A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira. VR Juggler: A Virtual Platform for Virtual Reality Application Development. In *Proc. VR '01*, 89. IEEE, 2001.
9. P. Dragicevic and J.-D. Fekete. Support for input adaptability in the icon toolkit. In *Proc. ICMI*, 212–219. ACM, 2004.
10. S. Eilemann, M. Makhinya, and R. Pajarola. Equalizer: A Scalable Parallel Rendering Framework. *IEEE TVCG*, 15(3):436–452, 2009.
11. G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Trans. Graph.*, 21(3):693–702, 2002.
12. S. Huot, C. Dumas, P. Dragicevic, J.-D. Fekete, and G. Hégron. The MaggLite post-WIMP toolkit: draw it, connect it and run it. In *Proc. UIST '04*, 257–266. ACM, 2004.
13. B. Jeong, L. Renambot, R. Jagodic, R. Singh, J. Aguilera, A. Johnson, and J. Leigh. High-performance dynamic graphics streaming for scalable adaptive graphics environment. In *Proc. SuperComputing*. ACM, 2006.
14. W. König, R. Rädle, and H. Reiterer. Interactive design of multimodal user interfaces. *J Multimod. UI*, 3:197–213, 2010.
15. M. Lorenz, G. Brunnett, and M. Heinz. Driving tiled displays with an extended chromium system based on stream cached multicast communication. *Parallel Comput.*, 33(6):438–466, 2007.
16. M. Nancel, J. Wagner, E. Pietriga, O. Chapuis, and W. Mackay. Mid-air pan-and-zoom on wall-sized displays. In *Proc. CHI '11*. ACM, 2011. In press.
17. T. Ni, G. S. Schmidt, O. G. Staadt, M. A. Livingston, R. Ball, and R. May. A Survey of Large High-Resolution Display Technologies, Techniques, and Applications. In *Proc. VR '06*, 223–236. IEEE, 2006.
18. E. Pietriga. A Toolkit for Addressing HCI Issues in Visual Language Environments. In *Proc. VL/HCC'05*, 145–152. IEEE, 2005.

Gliimpse: Animating from Markup Code to Rendered Documents and Vice Versa

Pierre Dragicevic[1]
[1]INRIA
F-91405 Orsay, France
dragice@lri.fr

Stéphane Huot[2,1]
[2]LRI - Univ. Paris-Sud & CNRS
F-91405 Orsay, France
huot@lri.fr

Fanny Chevalier
OCAD University
Toronto, Canada
fchevalier@ocad.ca

Figure 1: Detail of an animation between this article and its source code.

ABSTRACT
We present a quick preview technique that smoothly transitions between document markup code and its visual rendering. This technique allows users to regularly check the code they are editing in-place, without leaving the text editor. This method can complement classical preview windows by offering rapid overviews of code-to-document mappings and leaving more screen real-estate. We discuss the design and implementation of our technique.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Human Factors.

Keywords: Document editing, Animation, Markup code.

INTRODUCTION

Despite the popularity of WYSIWYG editors, authoring and editing styled documents using markup languages such as LaTeX and HTML is a widespread practice among computer literates and has recently been democratized by Wikis. Markup languages are widely used and advocated because they provide a clean separation between content and form, with benefits in terms of maintenability, portability, visual consistency, predictability, expressive power and expert user performance [9]. Some users also find it easier to focus on the content without having to deal with the form — or even without having to *see* the form, as exemplified by the recent popularity of "dark room" word processors [4].

P. Dragicevic, S. Huot, F. Chevalier.
Gliimpse: Animating from Markup Code to Rendered Documents and Vice Versa.
In UIST '11: Proceedings of the 24th Symposium on User Interface Software and Technology, ACM, October 2011.
Authors Version

One difficulty with markup code is that, although it is human-readable, users cannot always predict the results with accuracy and need to regularly check if it is correct and produces expected results. This typically requires a series of actions to re-render the document and display it in a separate environment from the text editor. This causes disruptions of the editing flow that can be daunting to beginners who need to continuously check their code and sometimes search for commands by trial-and-error. Even the most experienced users occasionally need to check commands they are unsure about and having to switch from the domain object (e.g., an article being written) to the tools (a markup interpreter or document viewer) [3] may cause them to lose their train of thought.

A great deal of effort has been spent in text editing environments to address these issues. Markup editors help users write correct code from the start by providing tools such as syntax highlighting and auto-completion. Since these tools will never eliminate the need for checking the final document, many efforts have also been spent at improving the document preview workflow. Most editing environments now provide shortcuts for re-rendering and refreshing the document into the viewer. Some of them further support rendering on-the-fly, and sometimes WYSIWYG selection and/or editing. Although these tools dramatically improved the usability of document markup languages, they still require users to deal with two separate windows.

In this article, we present an alternative, in-place document preview approach called Gliimpse. This technique lets users quickly "glimpse" into the rendered document by having the markup code smoothly transition to the rendered document upon a hot key press. It is based on the observation that markup code bears visual similarities with the document it produces, making it a good candidate for animated transitions [16, 7, 8]. Animated transitions have unique features that can potentially make them a useful complement to existing document markup editing tools. They are:

- *Implicit*. Users do not have to explicitly select the pieces of code or regions in the document they are interested in as in synchronized views: they only have to follow them visually during animated transitions.
- *Contextual*. Animated transitions provide context on regions that are not of immediate interest to users, and thus can possibly help them by giving quick overviews of where things go from the code to the document and providing opportunities for incidental discoveries (e.g., spotting a mistake elsewhere). The explanatory power of animations might further help beginners learn a new markup language.
- *Concise*. When the animation is invoked as a quasi-mode like in Gliimpse (using a hot key), users can quickly check the results of a formatting command they are unsure about and immediately come back to the code (the only bottleneck being rendering time). Our animations are fast but smooth enough to let users follow objects of interest and/or build a quick mental map of where pieces of code end up in the final document.
- *In-place*. The document is shown within the text editor itself, which saves screen real-estate and makes it possible to display more content relevant to the writing task. Gliimpse additionally uses a visual stabilization algorithm that tries to have the region of code currently edited stay in place.

After providing a brief overview of related work, we describe the basic features of Gliimpse. We then go into more details on the design and implementation issues behind our technique and finally discuss possible future work.

RELATED WORK

Gliimpse is related to two bodies of work: document editing and animated transitions. We briefly review them here.

Document Editing

Document markup languages and WYSIWYG systems are two approaches for editing documents, each with their own advantages and drawbacks. There has been a lot of effort in developing tools that combine the benefits of the two.

Examples of such efforts are integrated editing environments that combine a markup language editor with a WYSIWYG view. This idea was first introduced with the Lilac document editor [5]. Current examples include Dreamweaver and Firebug for HTML, and Instant Preview and Whizzy-TEX for TEX. These tools typically support synchronized highlighting (hovering an element in one view highlights the corresponding element in the other) and in some cases synchronized editing (changes in one view are reflected in the other). Such tools have been proved very valuable but since the use of two windows might sometimes come with disadvantages — for one thing, two windows take more screen real-estate than one — single-view approaches have also been explored.

Single-view preview approaches are either markup-oriented or WYSIWYG-oriented. Markup-oriented ones include text editors with WYSIWYG display features like WikEd, LyX and X-Symbol [12]. These pre-render symbol commands (e.g., displaying `\int` as $\int$) or use rich font attributes in their syntax highlighting that resemble the final document. Preview-latex [12] pushes the concept further by rendering code regions such as math formulaes in-place. As for WYSIWYG input features, toolbars and menus for inserting markup commands into the code are common in advanced code editors like emacs. Conversely, WYSIWYG-oriented editors exist that let users type markup code that is either inserted in the document and interpreted later like in Wikispaces or interpreted on-the-fly like in TeXmacs [17]. To further explore this rich design space, we propose an alternative in-place preview approach that uses animated transitions.

Figure 2: Animation of an HTML form.

Animated Transitions

Animated transitions are a type of animation that consist in showing a visual change in a smooth rather than an abrupt way. Their use in user interfaces has been advocated [7, 16] and studies have shown that they can help understand the spatial relationship between views and help users track changes in a variety of tasks (for a brief review, see [8]).

Animated transitions can however be hard to design and to implement. Animated text, in particular, has been used for various purposes such as expressing ideas and emotions [14] but there has been little work on when and how to support animated text transitions. Two exceptions are Chang et al's system for showing and hiding annotations in documents [6] and more recently Diffamation, a system for showing document edits over time [8]. The latter work has shown that animating text between revisions rather than abruptly flipping pages helps users navigate in edit histories.

Gliimpse targets a different application domain and also differs from the design and implementation standpoints. Chang et al's system merely animates the scale and position of text paragraphs. The Diffamation system introduces a richer animation language involving text insertions and deletions and paragraph reflow. Gliimpse goes a step further by supporting animation between documents having a different layout, different fonts, and possibly involving complex transitions such as a markup command changing into an image.

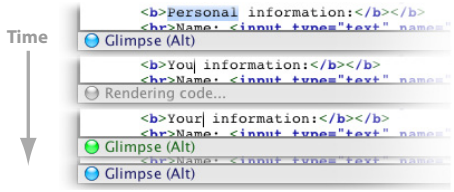


Figure 4: Gliimpse's status bar.

GLIIMPSE OVERVIEW

Gliimpse has been designed to be a quick, in-place preview tool for document markup languages. With Gliimpse, users stay in their text editing environment and can focus on their code, but are still able to check its rendering whenever they need. Whenever the user presses a hot key (we use Alt), the text editor switches to a document view with a fluid 1-second animation (Figures 1, 2). When she releases the hot key, the reverse animation is performed and she is back in her code.

The use of smooth animated transitions rather than abrupt view switches is motivated by previous work showing their benefits for navigating across text revisions [8]. Although fast and visually rich animated transitions can initially intimidate users, specific portions of text are easy to follow [8].

Gliimpse always transitions to the document region corresponding to the code currently in focus. The code focus is the area immediately surrounding the caret when it is visible, or the whole viewport content otherwise. The result is that the piece of code being edited will not move while the animation is performed, unless the document viewer has reached its scrolling limits. When the user moves the caret outside the viewport by scrolling elsewhere in the text editor and uses Gliimpse, the overall motion of the viewport is stabilized.

Since an actual document viewer is displayed upon completion of the animation, the user can scroll the rendered document while holding the hotkey. The viewport motion is then stabilized in the opposite direction at key release and Gliimpse animates back to a possibly new region of the code. Gliimpse can therefore be used as a navigation aid: the user just has to zoom the document out then gliimpse whenever she wants to jump to a different part of the document.

Because the layout of documents and markup code do not always match (e.g., with table cells and floating figures), visual objects can cross each other. To address this, Gliimpse has an option where objects that move with respect to the current focus follow curved paths. Figure 3a on the next page shows an example where a table (green square) is being edited. While gliimpsing, a larger table defined below in the code jumps above but goes around the focus (purple arrow). The user sees that the large table went to the wrong place and edits its placement options, which changes the focus to the large table and stabilizes it during the next gliimpse (Figure 3b).

This scenario illustrates how the overview and context provided by animations can help users make incidental discoveries and quickly build a mental map of where things go from the code to the document and vice versa. This didactic aspect of code / document animation can be exploited to help users learn a markup language, for example in Web tutorials.

Finally, when the code is edited, a background process rerenders the document and the animation. In our current prototype, this can last from 1/10 sec to more than a minute depending on document size (see the implementation section). A gray light in the Gliimpse status bar indicates that the process is working (Figure 4). After the user stops editing and once the animation is ready, the light switches to green, meaning that Gliimpse can be used. After a few seconds the light then switches to blue, meaning that a higher-quality, visually smooth version of the animation is ready to play.

DESIGN AND IMPLEMENTATION

We implemented the Gliimpse prototype in Java, with basic support for LaTeX, HTML, MediaWiki and RTF documents. We describe how we animate between these markup languages and the rendered documents.

Mapping the Code View with the Document View

To be able to compute animations, one needs to first generate a *code view* (i.e., a visual representation of the code as shown by the text editor), a *document view*, and retrieve a precise (character-level) mapping between the two views.

Figure 5 illustrates the problem and introduces notations that will be used in the rest of this section. It shows relationships between the markup text (T_0, bottom left of the Figure), the code view (V_0, top left), the document view (V_1, top right) and the document in raw text format (T_1, bottom right). We use the notation $X_A Y_B$ to denote a function that maps subsets of X_A to subsets of Y_B (thick black lines in the Figure):

- V_0V_1 (top) is the mapping between the two views. We assume V_0 and V_1 to be collections of glyphs and other graphical objects (possibly structured as a scene graph) with all information needed to render them individually (bounds, font, etc.). The mapping function V_0V_1 is the information we need in order to compute animations, which virtually no programming library or API directly provides.
- T_0V_0 (bottom left) is the mapping between the source code and its rendering in the text editor, which we assume to be a function that maps subsets of T_0 (i.e., collections of character indices) to subsets of V_0. This information is typically provided by the text editor's inspection methods or accessibility API.

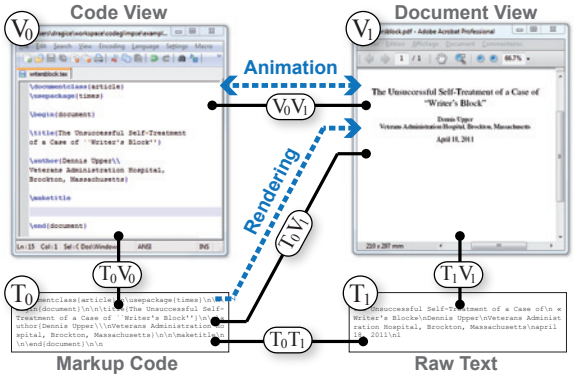


Figure 5: Mappings between the code (T_0), its view (V_0), the document view (V_1) and its text version (T_1).

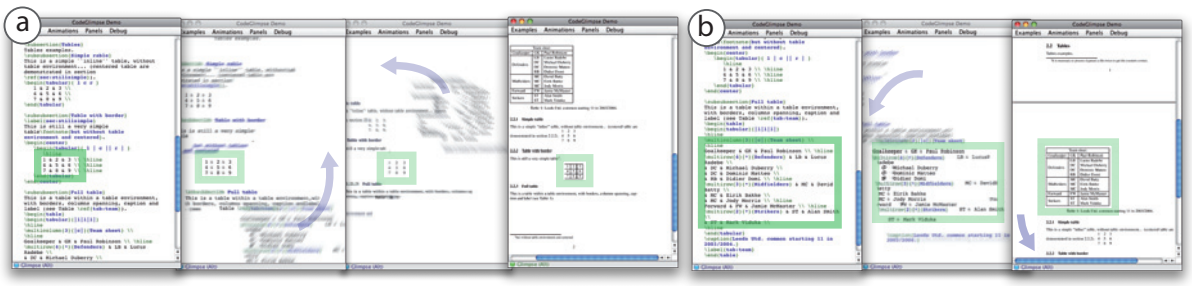


Figure 3: a) Motion of a large table (arrows) while another table is in focus (rectangle) ; b) The large table is now the focus.

- T_0V_1 (diagonal line) is the mapping between the source code and the document view. Markup language interpreters and renderers rarely maintain this information and when they do, it is often at a too coarse level of granularity to allow for precise animations. For example, TEX Source Specials and SyncTEX only provide mappings at the line level due to limitations of the TEX engine [13].
- T_1V_1 (bottom line) is the mapping between a raw text version of the document and the document view. This information is typically maintainted by document viewers to allow for text selection or to provide accessibility support.

The best way to reconstruct V_0V_1 is by computing $V_0V_1 = T_0V_0^{-1} \circ T_0V_1$. However, since getting T_0V_1 at the character level is hard in practice and because we initially just wanted to build a prototype, we chose to reconstruct V_0V_1 by computing $V_0V_1 = T_0V_0^{-1} \circ T_0T_1 \circ T_1V_1$. This approach is less robust but more flexible and makes it easier to plug Gliimpse into any interpreter and renderer available in Java.

In our current prototype, we obtain V_0 and T_0V_0 from the Java's text component (`JEditorPane`) inspection methods. The same Java component is used to render HTML, MediaWiki and RTF documents, and also provides V_1, T_1 and T_1V_1. LATEX source code is interpreted through a native call to `pdflatex` and the generated PDF file is rendered with a custom Java component based on the Sun PDF Renderer. We use the fonts from this renderer and the Apache PDFBox library to extract V_1, T_1 and T_1V_1. In all formats, non-character elements are mapped to white spaces in T_1.

The mapping T_0T_1 is built by cleaning up T_0 and T_1 and computing a diff between the two strings. We use Myer's diff algorithm [15], which supports deletions, insertions and moves. The cleaning up of T_0 essentially consists in stripping out markup elements and replacing non-character elements such as `<img*/>` with custom tags. The white spaces in T_1 that are mapped to non-character glyphs are replaced with the same tags. All string operations maintain a mapping with the original character indices. T_0T_1 can therefore be obtained by computing $T_0T_1 = T_0T_0^\star \circ T_0^\star T_1^\star \circ T_1 T_1^{\star -1}$, with $T_0^\star$ and $T_1^\star$ being the processed texts and $T_0^\star T_1^\star$ their diff.

For the formats we support, this method accurately rebuilds character mappings between markup code and the raw text version of simple documents. It is however not robust enough for a final product: duplicate text regions and complex mappings can defeat the diff algorithm and cause document regions to be either wrongly animated or not animated at all. We implemented a T_0T_1 mapping editor and used it to author the LATEX math tutorial scenario shown in the accompanying video. Fully automated approaches are arguably preferable and we hope that in-place preview techniques like Gliimpse will inspire the implementation of accurate, robust and usable APIs for mapping code views with rendered views.

Animating Between Fonts

In contrast with text animation techniques described in previous work [6, 14, 8], we need to animate between different fonts. Parametric typefaces [1] can linearly interpolate between glyphs but only if they share the exact same structure by design. Morphing between arbitrary shapes is a hard problem for which methods have been proposed [2] but they are computationally expensive.

For the purposes of animated transitions, we found that simply using alpha-blending to produce a dissolve effect yields excellent visual results. However, for this effect to work glyphs need to be properly aligned. Glyphs from different fonts can significantly differ in size even when the same font point size is used (Figure 6a). Aligning those glyphs based on their logical or geometrical bounds (Figure 6b,c) does not fully solve the problem. We therefore chose to refine glyph alignment using a pixel-based approach (Figure 6d).

We first collect all pairs of glyphs that need to be animated. For each pair, we draw one of the two glyphs off-screen with a fixed size (we use a point size of 30) and draw the other one on top using the method from Figure 6c. We compute the pixel color difference then vary the geometry of the second glyph (x, y, $width$, $height$ and x-$shear$ for italics) using a gradient descent scheme until a local minimum is reached. The result is cached and generalized to fonts of different sizes using linear interpolation. This method works best for within-character animations but can also polish animations such as changes in title capitalization.

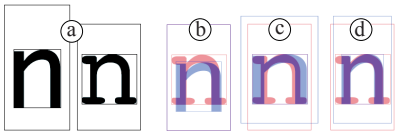


Figure 6: Alignment of two glyphs with the same font size (a), based on logical bounds (b), geometrical bounds (c) and pixels (d).

Animating the Document

Gliimpse's animation scheme is similar to the Diffamation system [8], with a few notable differences we outline here.

Object Interpolation. Like Diffamation, Gliimpse builds a parametric scenegraph that displays the initial document when rendered at $t = 0$ and the final document when rendered at $t = 1$. For $t \in]0, 1[$, the scenegraph essentially performs a linear interpolation of its nodes' bounding boxes.

We support 3 basic node types: *glyph transitions*, *non-glyph transitions* and *paragraph transitions*. Non-glyph transitions include transitions from a glyph to an image and between glyph groups. They are rendered by linearly interpolating the bounding boxes of the initial and final objects and alpha-blending them. Glyph transitions are computed the same way in addition to the alignment transformation previously described. Non-glyph objects are currently rendered as rectangles (see Figure 2). Paragraphs will be described later on.

When an object in V_0 or V_1 maps to nothing, we rapidly fade it in or out, in-place (as opposed to [8] where objects grow or shrink). Furthermore, we found that animating between groups of glyphs produces visual clutter or very wide characters (e.g., when animating from `\ref{fig:teaser}` to `1`), both of which are visually unpleasant. We therefore occlude those transitions at the middle of the animation by overlaying a rectangle with opacity $1 - 2 \cdot |t - 1/2|$ (see Figure 2). Finally, the scenegraph's root is animated by interpolating its background color and its position in the two viewports.

Paragraph Extraction. Like in [8], we group scenegraph nodes in paragraphs to be able to animate text reflow. However, we cannot rely on well-formed paragraph structures because some formats such as PDF produce V_1 content that merely consists in flat collections of glyphs. We therefore extract paragraphs from individual glyphs as follows:

We iterate over characters t^i of T_0 and at each step we compute the bounding box B^i of the object $T_0V_1(t^i)$ in V_1 and append it to the bounding box L^i of the current text line in V_1. At each step we test the following cases (for all our formats we use $vspace_{T0} = 2$ and $vspace_{V1} = 2$):

- If $\{t^{i-k}, \ldots, t^i\}$ contains only whitespaces among which $vspace_{T0}$ carriage returns, a new paragraph is created,
- if $T_0V_1(t^i) = \varnothing$ we proceed to the next index $i + 1$,
- if $B^i_{y0} > L^{i-1}_{y0} + vspace_{V1} \cdot L^{i-1}_{height}$ a new paragraph is created,
- if $B^i_{x0} < B^{i-1}_{x0}$ and $B^i_{y0} > L^{i-1}_{y0}$ a new line is started, in which case L^i is initialized to B^i,
- if $B^i_{x0} > B^{i-1}_{x0}$ and $B^i_{y1} > L^{i-1}_{y0}$ and $B^i_{y0} < L^{i-1}_{y1}$ the line continues and L^i is updated to $L^{i-1} \cup B^i$.
- otherwise, a new paragraph is created.

Once a paragraph is created for the indices i to j, we iterate over all subsets of $\{t^i, \ldots t^j\}$ of cardinality > 1 that map to content in V_1 and add them to the paragraph if their content lies within the paragraph bounds in V_1.

Text Reflow. Gliimpse animates text reflow within paragraphs in a way similar to [8]: depending on which path is shorter, a character can either follow a direct path or be "modulo-animated", i.e., move along its line and reach the edge to re-appear on the other side. In our algorithm, each character is animated independently and modulo-interpolations are limited to a full paragraph width. The visual effect on a paragraph with growing width would be that words on the second line move slowly to the left, those on the third line move similarly but faster, and so on until a line breaks in two pieces, one going quickly to the left and the other one slowly going up. Finally, our modulo metrics accounts for the fact that our paragraphs can have lines of different heights and these heights are linearly interpolated during the animation.

Scrolling stabilization. We stabilize horizontal and vertical scrolling between the code and the document viewports by minimizing the average motion of objects that are visible in the source viewport, as described in [8]. In addition, when the caret is visible, we use its position as the region of interest and stabilize the nearest object rather than the entire viewport. Stabilization is recomputed every time the user scrolls into the code, moves the caret, or scrolls into the document.

Curved Trajectories. In contrast with [8], we support diff move operations and chunks of text can therefore cross on the screen. To make these motions easier to understand and limit occlusions of the region of interest, an option allows objects to follow curved trajectories (see Figure 3). Our method, inspired from link drawing techniques in graphs [11], consists in having objects follow an arc and those moving in the opposite direction follow an arc oriented to the opposite side. More specifically, after scrolling stabilization we add to the absolute trajectory $P^{(t)}$ of each object (paragraph or isolated node) the vector $sin(\pi t)^k \cdot [r_x(P^{(1)}_y - P^{(0)}_y), r_y(P^{(1)}_x - P^{(0)}_x)]$. We use $k = 0.5$, $r_x = -0.5$ and $r_y = 0.05$.

Since arc radii are proportional to object motion, animations with no crossings will have close-to-straight trajectories (since they have been stabilized) but crossing objects will deviate from their path as if they tried to avoid each other. Even if this method does not guarantee the absence of overlaps, it presents the advantage of being context-free (every object ignores the position of others), which makes it simple and guarantees motion coherence (objects which are normally close will remain close). The asymmetry between the values we chose for r_x and r_y stems from fact that documents are structured in lines.

Playing Back and Recomputing Animations

The animated scenegraph is parented to a Java layered container that also contains the document viewer and the text editor. When the hot key is pressed, the scenegraph is set to $t = 0$, brought to the top and the animation starts. When the scenegraph shows the final document at $t = 1$ the actual document viewer is brought to the top. This transition is shown with a quick dissolve effect because the document view occasionally shows decorations that are not inspectable and hence not visible during the animation. The reverse sequence of operations is performed when the hot key is released.

We animate between $t = 0$ and 1 with a duration of 1 second, which has been shown to be appropriate for reasonably complex visual transitions and with a slow-in slow-out pacing, which has been shown to facilitate object tracking [10].

		Task a (s)	b (s)	c (s)	d (s)	e (s)	FPS (Hz)
HTML	1^{st}	0.21	0.12	0.04	0.03	0.29	75
	2^{nd}	0.21	0.07	0.02	0.01	0.01	89
LaTeX 1	1^{st}	1.90	0.28	0.41	0.06	0.64	36
	2^{nd}	0.97	0.26	0.28	0.04	0.02	40
LaTeX 2	1^{st}	3.72	37.0	8.68	10.5	19.5	12
	2^{nd}	2.68	41.3	8.53	10.5	0.62	13

Table 1: Task execution times for three documents.

Animations are updated on-the-fly by a scheduler that runs in a separate thread and skips unnecessary computations. For example, modifying the source code requires a) re-rendering the document, b) computing the T_0T_1 mapping, c) inspecting the views for T_0V_0 and T_1V_1, d) building the scenegraph, e) aligning new glyphs and f) stabilizing the views. However, when a view is resized only tasks c) to f) are performed, and when it is scrolled only task f) is done. After the animation is ready the scheduler runs an off-screen rendering task after which complex animations play back more fluidly with an optional motion blur effect (visible in the Figures).

Table 1 shows computation times on a PC with a 2.40GHz Intel Xeon processor for the tasks mentioned above (task f is negligible), as well as the animation frame rate. Figures are given for the first run (1^{st}) and after inserting a character in the code (2^{nd}). The first two examples are a short HTML file (700 characters) and LaTeX file (1900 characters, about 1 page). These computation times are adequate for interactive use but the third example, a 5-page LaTeX draft of this article (30,000 characters), shows that our current prototype does not scale up. The most expensive task is the code/document mapping task, which uses heavy regexp searches and hashtables. This operation can be optimized or avoided altogether with an API that provides T_0V_0 as previously discussed.

CONCLUSION AND FUTURE WORK

We presented Gliimpse, a quick in-place preview technique that smoothly transitions between markup code and its visual rendering. This technique is an alternative to classical preview tools that takes less screen real-estate and offers rapid overviews of code-to-document mappings.

More work is needed to identify the actual benefits of animations over well-established approaches such as synchronized views. We hypothesize that animations can save users time and effort because they involve *attentional* rather than *explicit* selection of regions of interest. This is however only a conjecture that needs to be put to the test. And even in case animation helps, it is likely that synchronized views are more suited for some tasks and that both need to be supported.

Possible future extensions include local animated previews, integration with synchronized highlighting and editing, animation of series of code transformations (e.g., XSLT) and animation of non-textual markup documents like music sheets or vector graphics. Furthermore, Gliimpse is currently only a prototype and a more robust version is necessary for users to be able to try it on real writing tasks.

ACKNOWLEDGEMENTS

We thank Jean-Daniel Fekete for insightful discussions.

REFERENCES

1. Adobe. Designing multiple master typefaces, 1995. http://partners.adobe.com/public/developer/en/font/5091.Design_MM_Fonts.pdf.
2. M. Alexa, D. Cohen-or, and D. Levin. As-rigid-as-possible shape interpolation. In *Annual Conference on Computer Graphics*, 157–164, 2000.
3. M. Beaudouin-Lafon. Instrumental interaction: an interaction model for designing post-wimp user interfaces. In *Proc. CHI '00*, 446–453. ACM.
4. S. bin Ali. 20 fantastic full screen text editor for distraction free writing, 2009. http://www.techmalaya.com/2009/02/07/full-screen-text-editor-blogger/.
5. K. Brooks. Lilac: a two-view document editor. *Computer*, 24(6):7 –19, jun 1991.
6. B.-W. Chang, J. D. Mackinlay, P. T. Zellweger, and T. Igarashi. A negotiation architecture for fluid documents. In *Proc. UIST '98*, 123–132. ACM.
7. B.-W. Chang and D. Ungar. Animation: from cartoons to the user interface. In *Proc. UIST '93*, 45–55. ACM.
8. F. Chevalier, P. Dragicevic, A. Bezerianos, and J.-D. Fekete. Using text animated transitions to support navigation in document histories. In *Proc. CHI '10*, 683–692. ACM.
9. J. H. Coombs, A. H. Renear, and S. J. DeRose. Markup systems and the future of scholarly text processing. *Commun. ACM*, 30:933–947, Nov. 1987.
10. P. Dragicevic, A. Bezerianos, W. Javed, N. Elmqvist, and J.-D. Fekete. Temporal distortion for animated transitions. In *Proc. CHI '11*, 2009–2018. ACM.
11. J.-D. Fekete, D. Wang, N. Dang, and C. Plaisant. Overlaying graph links on treemaps. In *Proc. Infovis '03 (demo)*, 2003.
12. D. Kastrup. Revisiting WYSIWYG paradigms for authoring latex, 2002. http://www.tug.org/TUGboat/tb23-1/kastrup.pdf.
13. J. Laurens. Direct and reverse synchronization with synctex. *TUGboat*, 29(3), 2008.
14. J. C. Lee, J. Forlizzi, and S. E. Hudson. The kinetic typography engine: an extensible system for animating expressive text. In *Proc. UIST '02*, 81–90. ACM.
15. E. Myers. An $o(nd)$ difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.
16. G. G. Robertson, J. D. Mackinlay, and S. K. Card. Cone trees: animated 3d visualizations of hierarchical information. In *Proc. CHI '91*, 189–194. ACM.
17. J. van der Hoeven. Gnu texmacs: A free, structured, wysiwyg and technical text editor. In *Actes du Congres GUTenberg*, volume 39-40, 39–50, 2001.

BiTouch and BiPad: Designing Bimanual Interaction for Hand-held Tablets

Julie Wagner[1,2,3]
wagner@lri.fr
[1] INRIA
F-91405 Orsay, France

Stéphane Huot[2,1,3]
huot@lri.fr
[2] Univ Paris-Sud (LRI)
F-91405 Orsay, France

Wendy E. Mackay[1,2,3]
mackay@lri.fr
[3]CNRS (LRI)
F-91405 Orsay, France

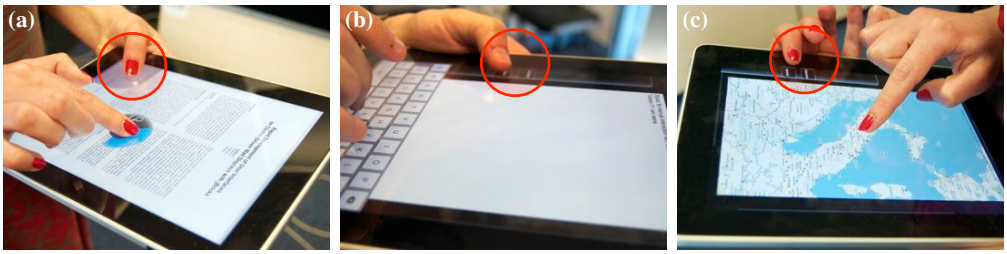


Figure 1. Bimanual interaction with BiPad: a) navigating a PDF, b) shifting to uppercase, c) zooming on a map. The non-dominant support hand can tap, make gestures or perform chords, thus modifying interaction by the dominant hand.

ABSTRACT

Despite the demonstrated benefits of bimanual interaction, most tablets use just one hand for interaction, to free the other for support. In a preliminary study, we identified five holds that permit simultaneous support and interaction, and noted that users frequently change position to combat fatigue. We then designed the BiTouch design space, which introduces a support function in the kinematic chain model for interacting with hand-held tablets, and developed BiPad, a toolkit for creating bimanual tablet interaction with the thumb or the fingers of the supporting hand. We ran a controlled experiment to explore how tablet orientation and hand position affect three novel techniques: bimanual taps, gestures and chords. Bimanual taps outperformed our one-handed control condition in both landscape and portrait orientations; bimanual chords and gestures in portrait mode only; and thumbs outperformed fingers, but were more tiring and less stable. Together, BiTouch and BiPad offer new opportunities for designing bimanual interaction on hand-held tablets.

Author Keywords

Bimanual Interaction; Hand-held tablets; Multi-touch tablets; BiTouch design space; BiPad.

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: User Interfaces—*Interaction styles; Input devices and strategies*

J. Wagner, S. Huot, W. E. Mackay.
BiTouch and BiPad:
Designing Bimanual Interaction for Hand-held Tablets.
In CHI'12: Proceedings of the 30th International Conference on Human Factors in Computing Systems, ACM, May 2012.
Authors Version

INTRODUCTION

Multi-touch tablets have become increasingly popular over the past few years, combining relatively large screens with portability. Their form factor encourages uses in situations in which the user stands or walks, for example teachers can control simulations in class and nurses can track patients on interactive clipboards [7]. Although commercial tablets offer intuitive interaction techniques such as a *swipe* to displace an object or a *tap* to select an item, they do not fully exploit the range of interaction possibilities found in the research literature. In particular, tablets are not designed to support bimanual input, despite the demonstrated ability to increase performance [18] and precision [4], as well as to enhance the user experience [16, 29].

Existing bimanual interaction techniques were designed for independently supported displays or tabletops. Portable devices pose an additional challenge: how to account for the need to hold the device while interacting with it. Very small devices, such as PDAs and smart phones, offer limited possibilities for bimanual interaction, usually just typing with both thumbs. Multi-touch tablets, with their larger screens, offer as-yet unexplored opportunities for true bimanual interaction. Our goal is to better understand the design space for bimanual, multi-touch interaction on hand-held tablets and to demonstrate how designers can obtain the benefits of bimanual techniques, taking into account the challenge of supporting the device while interacting with it.

We begin by analyzing the related literature and describe a preliminary study that investigates how users hold tablets as they interact. Next, we present the *BiTouch* design space which identifies the key dimensions for designing bimanual multi-touch interaction. We next present BiPad, a toolkit that helps designers add various bimanual interaction to off-the-shelf multi-touch tablets, illustrated with three sample

applications. We also report the results of an experiment that compares one- and two-handed interaction performance with respect to tablet orientation, finger placement and interaction technique. We conclude with implications for design and directions for future research.

RELATED RESEARCH

Desktop-based bimanual interaction techniques increase both performance and accuracy [1, 5, 12] and are more convenient when performing highly demanding cognitive tasks [16, 10]. Some techniques provide symmetric control [2]. For example, *Symspline* gives both hands equal roles when manipulating curves [15]. However, most bimanual interaction techniques build upon Guiard's *kinematic chain model* [9], based on his observations about the asymmetric relationship between the two hands [1]. For example, toolglasses, magic lenses and bimanual palettes [5, 3, 17] each use the non-dominant hand to control the position of an interactive palette while the dominant hand selects specific functions.

Bimanual Interaction: Stationary Multi-touch Surfaces

Multi-touch tables and graphics tablets are inherently well-adapted to bimanual interaction, since the user can use multiple fingers from either or both hands. Studies have shown that bimanual interaction techniques can improve performance [6, 14] and selection accuracy [4]. However, these studies assume that both hands are free to interact, e.g. on a stationary multi-touch surface or a small multi-touch device placed on a table. We are interested in hand-held tablets which require at least one hand to support the device, thus restricting the ability to interact.

Bimanual Interaction: Small Portable Devices

Commercially available PDAs and smart phones are designed primarily for one-handed interaction [20] due to their small size. Most interaction is accomplished with the index finger, although some techniques use the thumb, since it can reach the screen from most carrying positions [11, 13, 22]. Other approaches use the outer frame of the phone to improve pointing accuracy [8] or to disambiguate among actions and enrich the interaction vocabulary [21].

Several research prototypes offer the potential for bimanual interaction by adding hardware. For example, *HandSense* [27] uses capacitive sensors to distinguish among six different grasping styles. One could create simple bimanual tasks by allowing these grasps to modify the actions of the dominant interaction hand. An alternative is *HybridTouch* [25], which adds a rear touchpad to a PDA to enable simultaneous front and back interaction.

Wobbrock et al. [28] investigated how different hand positions on the front or back of a handheld device affect interaction performance with the index finger or the thumb. They found that the index finger performed best in all conditions, front or back, and that horizontal movements were faster and more accurate. Although useful for comparing thumb and finger performance on small devices, additional research is needed to understand bimanual interaction on larger portable devices, such as multi-touch tablets.

Bimanual Interaction: Multi-touch Tablets

Hand-held tablets offer new possibilities for bimanual interaction. Although their larger screen size and bezels make two-handed thumb typing less convenient, they also afford various support positions and can accommodate interaction with the thumbs and multiple fingers from both hands.

To date, most bimanual interaction techniques require additional hardware, e.g. to detect touches on the back or sides of the device. For example, *RearType* [24] includes a physical keyboard on the back of a tablet PC. Users hold it with both hands while entering text, thus avoiding an on-screen keyboard and graphical occlusion by the fingers. *Lucid Touch* [26] is a proof-of-concept see-through tablet that supports simultaneous touch input on the front and on the back of the device. Users hold the device with both hands, with thumbs on the front and remaining fingers on the back. The device is small enough that users can reach the entire screen, allowing multi-touch interaction with both support hands without graphical occlusion. However, the arm-mounted camera currently makes this approach impractical.

Another intriguing possibility is *Gummi* [23], a prototype "bendable" tablet that enables limited bimanual interaction by deforming the device. For example, a user could scroll through a list via a 2D position sensor on the back and then select an item by bending the device. Such dual-surface approaches are well suited for simple selection and navigation tasks [30], but are less appropriate for complex tasks that require additional input from the back or when users adjust how they hold the tablet.

Our goal is to incorporate bimanual interaction on tablets, using only the multi-touch surface without additional hardware. The next section describes a preliminary study that investigates how users unconsciously hold tablets while interacting with them, as they sit, stand and walk.

PRELIMINARY STUDY: HOLDING TABLETS

Studying how people 'naturally' hold tablets is tricky. Rather than asking directly, we asked users to perform a distractor task while observing how they held the tablet.

Participants. Six men and two women, average age 30. Four owned iPads, four had never used a tablet.

Apparatus. Apple iPad1 (display: 9.7", weight: 680 g, dimensions: $19 \times 24.3 \times 1.3$ cm).

Procedure. We told participants that we were interested in how pointing and scrolling performance varies as people sit, stand and walk, given different tablet orientations. This was intentionally misleading, since we were really studying how they unconsciously held the tablet while interacting with it. The true experiment was a [2x3] within-subjects design with two factors: tablet *orientation* (landscape, portrait) and *stance* (sit, stand, walk), with tablet *hold* as the dependent measure. The distractor tasks were *pointing* (tapping five successive on-screen targets) and *scrolling* (moving a slider's thumbwheel from one end to the other). Pointing targets were distributed across six equal squares on the screen; slider

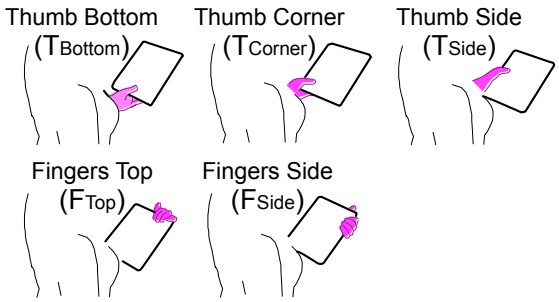


Figure 2. Five spontaneous holds (portrait orientation).

positions included the four screen borders and horizontally and vertically in the screen center.

Participants were asked to hold the iPad comfortably and perform each task as quickly as possible. They were allowed to adopt a new hold only when beginning a new block. Sessions lasted approximately 45 minutes. At the end, we debriefed each participant as to the true goal of the study to learn how they chose to hold the tablets. We first asked them to reproduce the holds they had used and then to adapt them so that the fingers or thumb of the support hand could reach the touch screen. We asked them to rate comfort and ease of interaction when using the support hand to interact and whether they had suggestions for other holding positions.

Data collection. We videotaped each trial and coded how participants supported the tablet with the non-dominant hand, wrist or forearm. We collected touch events, including those that occurred outside experiment trials and while reading instructions. We also measured completion time per trial.

Results

We did not find a single, optimal hold and found significant differences according to experience. All four novices used the same uncomfortable position: the fingers, thumb and palm of their non-dominant hand supported the center of the tablet, like a waiter holding a tray. Novices found this tiring but worried that the tablet would slip if they held it by the border. None found other holds. In contrast, the four experts easily found a variety of secure, comfortable holds. We identified ten unique holds, five per orientation, all of which involved grasping the border of the tablet with the thumb and fingers. Fig. 2 shows these five holds in portrait mode, with the thumb on the bottom, corner or side, or the fingers on the top or side.

Table 1 shows how these holds were distributed across the six conditions: most common was F-side (41%), least common was T-side (9%). The latter was deemed least comfortable, especially in landscape mode, but participants felt that they could use it for a short time. Experts tried nine of ten possible holds in the sitting and walking conditions, but only six when standing, omitting F-top or T-side in both orientations. Individuals varied as to how many unique holds they tried, from three to eight of ten possible. All switched holds at least once and two switched positions often (50% and 66%) across different blocks of the same condition.

Table 1. Total holds per condition (expert users)

	F_{side}	T_{bottom}	F_{top}	T_{corner}	T_{side}
Landscape	3	4	4	4	1
	8	4	0	4	0
	4	4	7	0	1
Portrait	8	3	1	0	4
	8	4	0	4	0
	8	1	3	1	3
	41%	21%	16%	14%	9%

We were also interested in whether *accidental touches*, defined as touches located more than 80 pixels from the target or slider, during or outside of experiment trials, interfered with intentional touches by the dominant hand. Experts who carried the tablet by the border made very few accidental touches (3%). All were with the dominant hand, far from the screen border, suggesting that they unconsciously prevented the support hand from touching the screen.

Design Implications

First, tablets can feel heavy and users are more comfortable when they can change orientation or swap the thumb and fingers. We should thus seek a small set of roughly equivalent bimanual interactive holds that are easy to shift between, rather than designing a single, 'optimal' hold. Second, users can use the thumb and fingers of the support hand for interaction. We can thus create interactive zones on the edges of the tablet, corresponding to the holds in Fig. 2, which were not vulnerable to accidental touches. Fig. 3 shows these zones in portrait and landscape mode. Although changes in the form factor of a tablet, such as its size, shape or weight, may affect these holds, users are still likely to shift between holds for comfort reasons, just as when reading a book or holding a notebook.

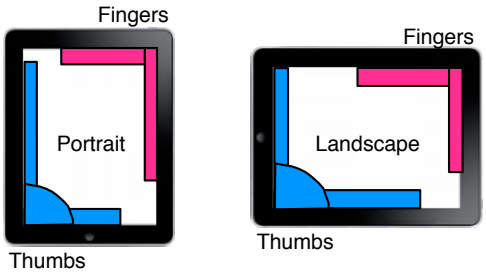


Figure 3. Five support-hand interaction zones.

The next section describes BiTouch, a design space for exploring how to incorporate bimanual interaction on hand-held multitouch tablets.

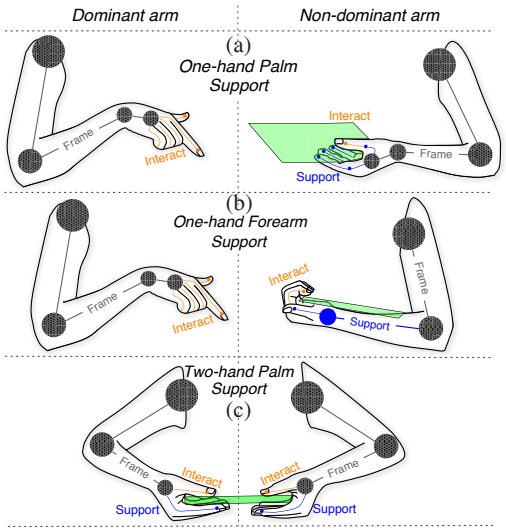


Figure 4. The user creates a spatial *frame*, *supports* the device, and *interacts* with it. Different holds offer different trade-offs with respect to interactive power and comfort.

BiTouch DESIGN SPACE

Unlike desktop PCs or multi-touch tables, bimanual interaction on hand-held tablets must account for the dual role of the non-dominant hand as it simultaneously carries the tablet and interacts with it. Although we designed the BiTouch design space to explore bimanual interaction on hand-held tablets, the reasoning applies to a wider range of human-body interaction with objects [19] and devices ranging from small, mobile devices to large, fixed interactive tables or walls.

Kinematic Chain: Frame, Support, Interact

The first step is to understand the complementary roles of support and interaction. Guiard's [9] analysis of bimanual interaction emphasizes the asymmetric relationship commonly observed between the two hands. He proposes the kinematic chain as a general model, in which the shoulder, elbow, wrist and fingers work together as a series of abstract motors. Each consists of a proximal element, e.g. the elbow, and a distal element, e.g. the wrist, which together make up a specific link, e.g. the forearm. In this case, the distal wrist must organize its movement relative to the output of the proximal elbow, since the two are physically attached.

Guiard argues that the relationships between the non-dominant and dominant hands are similar to those between proximal and distal elements: the former provides the spatial frame of reference for the detailed action of the latter. In addition, the movements of the proximal element or non-dominant hand are generally less frequent and less precise and usually precede the movements of the higher frequency, more detailed actions of the distal element or dominant hand.

We see the kinematic chain in action when users interact with hand-held tablets: the non-dominant hand usually supports the tablet, leaving the fingers and thumb of the dominant hand free to interact. Fig. 4 shows three bimanual alternatives,

Table 2. Trading off *framing*, *support* and *interaction* functions of the kinematic chain with respect to the body and the device.

Framing	
Location:	proximal link in the kinematic chain
Distribution:	1 – n body parts
Support	
Location:	none or middle link in the kinematic chain
Distribution:	0 – n body parts
Independence:	0% – 100% body support
Interaction	
Location:	distal link in the kinematic chain
Distribution:	1 – n body parts
Degrees of freedom:	0% – 100% body movement
Technique:	touch, deformation,...

based on the location of tablet support within the kinematic chain: the palm or forearm of the non-dominant arm (Fig. 4a, 4b); shared equally between the palms of both hands (Fig. 4c). In each case, the most proximal links control the spatial frame of reference; support links are always intermediate between framing and interaction links; and the most distal links use whatever remains of the thumb and fingers to interact.

The preliminary study highlighted ten user-generated support holds that permit the thumb or fingers to reach the interactive area. Each poses trade-offs between comfort and degrees of freedom available for interaction. For example, supporting the tablet with the forearm (Fig. 4b) provides a secure, stable hold but forces the fingers to curl around the tablet, leaving little room for movement. In contrast, holding the tablet in the palm (Fig. 4a) gives the thumb its full range of movement, but is tiring and less stable.

Note that comfort is subjective, influenced not only by the physical details of the device, such as its weight, thickness and size of the bezels, but also by how the tablet is held. For example, shifting between landscape and portrait orientations changes the relative distance between the tablet's central balance point and the most distal part of the support link. The tablet acts as a lever: users perceive it as heavier as support moves further from the fulcrum. The next step is to formalize these observations into a design space that describes existing and new bimanual holds and interaction techniques.

BiTouch Design Space

Table 2 summarizes the key dimensions of the BiTouch design space, according to framing, support and interaction functions of the kinematic chain. Each is affected by the relationship between specific characteristics of the human body, the physical device and the interaction between them.

Framing is handled at the most proximal *locations* within the kinematic chain and may be *distributed* over multiple parts of the body. ***Support*** always occurs in *locations* within the kinematic chain, distal to the frame. Support may be completely *distributed* over one or more body parts, symmetrically or not; shared with an *independent* support, e.g. a table or lap; or omitted, e.g. interacting on a freestanding interactive table.

Interaction is always handled at the most distal *location* in the kinematic chain, immediately after the support link. Inter-

action may be *distributed* across one or more body parts, often incorporating the thumbs or sets of fingers. The *degrees of freedom* available for interaction depend upon what remains after framing and support functions have been allocated, e.g. a finger tip, and the inherent movement capabilities of the body part, e.g. the pinky has little independent movement compared to the index finger. Possible *interaction techniques* are affected by all of the above, as well as the technical capabilities of the device. For example, touch sensors might appear on the front, side or back of the device, or the device itself might be deformable.

Hands that interact as well as support the device have fewer degrees of freedom available for movement. We thus expect the support hand to be non-dominant, capable of limited interaction, e.g. mode switches or menu choices, that frame the interaction of the freer dominant hand.

The BiTouch design space allows us to describe all of the user-generated holds from the preliminary study, as well as many from the literature, e.g. bimanual interaction on free-standing interactive tabletops. It also suggests directions for designing new bimanual interaction techniques. For example, although the hold in Fig. 4c did not appear in the preliminary study, it becomes an obvious possibility if we examine ways to share support across hands. Similarly, once we understand which thumbs or fingers are available for interaction and what constrains their potential movement, we can design novel interaction techniques.

The five basic holds in Fig. 2 can each support an interactive area on the edge of the tablet, reachable by either the thumb or fingers of the support hand. The BiTouch design space helps us create a set of novel bimanual interaction techniques that take into account the potential of the thumbs and fingers at the end of the kinematic chain. For example, all thumbs and fingers have at least a small amount of mobility available to perform *Taps*. The thumb in the T_{corner} hold is fully mobile and can perform *Gestures*. The presence of multiple fingers in the F_{side} hold makes it possible to perform *Chords*. The non-dominant role of the support hand suggests that these *Taps*, *Gestures* and *Chords* can be used to frame more elaborate interaction by the dominant hand, e.g. to select a menu item or to shift color while drawing a line.

BiPad TOOLKIT AND APPLICATIONS

Based on our preliminary study and the BiTouch design space, we designed the BiPad toolkit to help developers add bimanual interaction to off-the-shelf multi-touch tablets. BiPad creates five interactive zones, corresponding to those in Fig. 2, where the fingers or the thumb of the supporting hand can interact.

Software Prototype

The BiPad toolkit, written in Objective-C on Apple's iOS operating system, supports the development of bimanual applications as follows:

BiPad applications consist of one or more views, widgets and controllers, similar to standard iOS applications. The framework lays out the interface in the main view to control

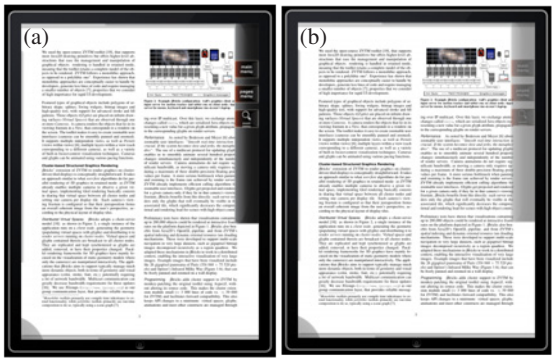


Figure 5. BiPad. a) F_{side} zone is active; other zones are shrunken. b) Unused zones remain partially visible if commands were assigned.

overlay feedback and advanced input management required to enable BiTouch interaction. The application defines BiPad-enabled functions that can be mapped to interactions with the support hand. For example, a text editing application could define `shift` and `num` functions equivalent to pressing the shift or number keys of a virtual keyboard.

BiPad zones appear on the sides and corners of the screen (Fig. 5). Applications can define various interactions for the support hand and modify the default visual representation, e.g., buttons for taps and guides for chords. Zones are displayed as 80-pixel strips, of which the 40 outermost are semi-transparent, on top of the edges of the application view. Zones may be permanently or temporarily visible and the user's hand position determines which is active. Temporarily visible areas shrink automatically when not in use, displaying only a narrow semi-transparent strip of pixels on the appropriate side. Touching once on the outer part of a shrunken BiPad zone causes it to slide out and enables interaction. If a zone contains interaction widgets and is configured to be temporarily visible, it does not shrink completely but remains semi-transparent (Fig. 5b).

BiPad Interaction Techniques

BiPad introduces three predefined interaction techniques for the support hand: bimanual *Taps*, *Chords* and *Gestures*. Bimanual *Taps* involve a press-and-release action on a button within a BiPad zone, using a finger or the thumb (Fig. 6a). Bimanual *Chords* involve multiple fingers pressing down simultaneously within a BiPad zone, and are not possible with thumbs. Fig. 6b shows how pressing the 'stroke' button with the index finger adds additional finger positions below. The user can adjust the stroke size by holding down a second finger on the appropriate button.

Bimanual *Gestures* involve sliding the thumb or finger, starting from a BiPad zone or from an edge related to a BiPad zone, as in Bezel Swipe [21]. In the border zones, *Gestures* are limited to orthogonal movements from the edge, but offer additional degrees of freedom for the thumb in the corner (up-to-down, right-to-left and diagonal). Small stroke shapes indicate the direction of the gesture and its function (Fig. 6c).

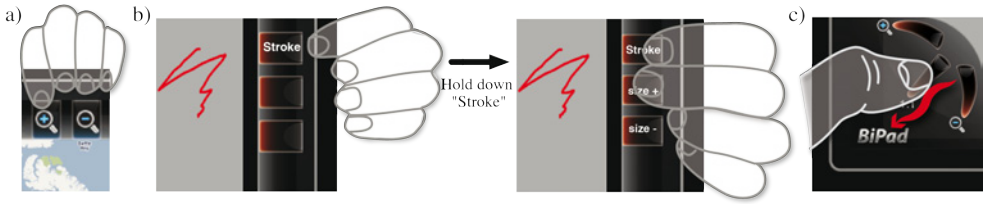


Figure 6. BiPad interaction techniques: a) Taps on buttons. b) Chords with multiple fingers. c) Gestures in multiple directions.

The application defines which BiPad interaction(s) will trigger which function in which zone(s). Applications can specify several interaction techniques for the same function depending upon which BiPad zone (and therefore *Hold*) the user registers. For example, an application might specify that a *Tap* with a finger on the F_{side} zone and a downward *Gesture* with the thumb in the T_{corner} zone will both shift modes for the dominant hand, triggering a pop-up menu rather than selecting an on-screen object.

BiPad Applications

We used BiPad to implement three applications that illustrate how to add bimanual interaction to handheld tablets (Fig. 1).

Quasi-modes and Shortcuts

BiPDF (Fig. 1a) is a PDF reader that uses standard touch gestures to navigate through pages, scroll or zoom the document. A pie menu contains additional commands, e.g. first/last page. As with many tablet applications, the user must touch and dwell to activate the menu instead of executing a gesture. We added a bimanual tap that speeds up interaction: while the user is touching the screen with the dominant hand, a tap on a BiPad button activates the menu immediately.

BiText (Fig. 1b) lets users create custom bimanual shortcuts for text entry, e.g. a button for the '`space`' key and a quasi-mode button for the soft keyboard's '`keypad`' key. Although the dominant hand can also reach these keys, it requires extra movement. The user can also assign any key from the keyboard to a BiPad button by simultaneously pressing the two. Modifier keys, such as the '`keypad`' key become quasi-modes: they activate the mode as long as they are being pressed. Two other BiPad buttons accept or reject the suggestions from the standard text completion engine, reducing movements by the dominant hand.

Menu navigation

BiSketch uses BiPad *Chords* to navigate a tool menu. First-level items, e.g. `color` or `stroke`, appear in the BiPad zone. The user chooses a tool and holds down the corresponding finger in the BiPad zone to trigger the next menu level. The user can then use another finger to select the desired option, e.g., `color` then `red`. Chords can trigger frequently used tools or options while drawing with the dominant hand.

Spatial multiplexing

The previous example refers to two-handed interactions based on temporal multiplexing. BiPad can also handle spatially multiplexed tasks. BiMap (Fig. 1c) lets users zoom in and out by pressing buttons with the support hand. They can select part of the map larger than the view port by (i) selecting with the dominant hand; (ii) simultaneously controlling the zoom factor with the non-dominant hand; and (iii) continuing to change the selection with the dominant hand.

EXPERIMENT

We ran a controlled experiment to determine whether BiPad bimanual interaction techniques outperform a common one-handed technique. We also wanted to see if the BiTouch kinematic chain analysis successfully identifies which bimanual holds are most comfortable and efficient.

We asked participants to stand while holding a multi-touch tablet, using one of the holds identified in the preliminary study. We then asked them to perform a series of bimanual *Taps*, *Gestures* or *Chords*, using the thumb or fingers of the non-dominant support hand to modify the actions of the dominant hand. The key research questions were:

Q1 Are two-handed BiPad techniques faster than a similar one-handed technique?

Q2 What are the trade-offs among the different bimanual holds, orientations and interaction techniques?

Participants. Nine men, three women, all right-handed, aged 22-35. Six own a touch-screen phone, one owns a tablet PC.

Apparatus. iPad1 (display: 9.7" , weight: 680g, dimensions: $190 \times 243 \times 13$ mm), running BiPad.

Procedure. We conducted a $[2 \times 5 \times 3]$ within-subjects design with three factors: ORIENTATION (portrait, landscape), HOLD (F_{side}, F_{top}, T_{bottom}, T_{corner}, T_{side}), corresponding to the five BiPad interaction zones, and TECHNIQUE (tap, chord, gesture), i.e. 30 unique conditions, plus the no-BiPad control, a standard one-handed task. We discarded eight conditions as impossible or impractical:

Chords can only be performed with the F_{side} and F_{top} HOLD (both *Orientations*) since a single thumb cannot perform multi-finger interactions.

Gestures were omitted from the F_{side} and F_{top} landscape conditions, since the short edge of the tablet cannot be held steadily on the forearm.

Trials were organized into blocks of 6 trials according to TECHNIQUE, ORIENTATION, and HOLD. Participants were asked to stand and support the tablet with a specified hold. In each trial, the participant touched four successive 80-pixel circular targets with the index finger of the dominant hand while holding the tablet with the non-dominant hand. Targets were arranged randomly around the center of the screen. The first target of a series was always green and one randomly

chosen target of the following three targets was red. When the red target appeared, the participant was instructed to use the specified technique to turn the target from red back to green before touching it with the dominant hand.

The four techniques for changing red targets to green include the three BiPad techniques: *Tap*, *Chord*, *Gesture*, and the no-BiPad control condition. The three chords use the index finger and one or both of the remaining fingers of the support hand (middle or ring finger). Gestures slide toward the center of the screen, except for T_{corner}, where the thumb slides up-down, down-up or diagonally. In the no-BiPad control condition, the user touches a button at the bottom of the screen with the dominant hand. The task was chosen to support both pointing and bimanual interaction, including mode switches and quasi-modes.

Participants began with the unimanual no-BiPad control condition, followed by the bimanual BiPad conditions (ORIENTATION, HOLD, TECHNIQUE) counter-balanced across subjects using a Latin square. Although this simplifies the experimental design, it does not account for potential order effects between unimanual and bimanual conditions. On the other hand, all of today's tablets are one-handed and it is unlikely that performing a bimanual task prior to a unimanual one would improve performance on the latter. Indeed, the more likely effect would be a drop in performance due to fatigue. To ensure that participants were familiar with the basic task and both conditions, we asked them to perform a three-trial practice block in portrait mode prior to each no-BiPad condition and to each TECHNIQUE×HOLD condition. They were also allowed to perform a one-trial recall prior to each TECHNIQUE×ORIENTATIONS×HOLD so they could find a comfortable position for the assigned hold.

To begin an experimental BiPad block, participants touched the specified BiPad zone to register the support hand. Participants were asked to maintain this hold throughout the block and perform each task as quickly as possible. At the end of each condition, they evaluated how comfortable it was to interact with the support hand using that hold. Each session lasted approximately 45 minutes.

In summary, we presented two orientations for no-BiPad, all 10 holds for bimanual taps, eight for bimanual gestures (no landscape thumb holds) and four for bimanual chords (fingers only). We thus collected 216 trials per participant:

- 6 replications of the *no-BiPad* control condition in both ORIENTATIONS (landscape, portrait): 12 trials;
- 6 replications of the *Tap* technique in all HOLD and ORIENTATION conditions: 60 trials;
- 6 replications of the three *Chord* techniques in both ORIENTATIONS for finger-based HOLDS (F_{side}, F_{top}): 72 trials;
- 6 replications of each of the three *Gesture* techniques:
  - two-finger-based HOLDS (F_{side}, F_{top}) in portrait ORIENTATION: 12 trials;
  - two thumb-based HOLDS (T_{bottom}, T_{side}) in both ORIENTATIONS: 24 trials;
  - one thumb-based HOLD (T_{corner}) in both ORIENTATIONS: 36 trials.

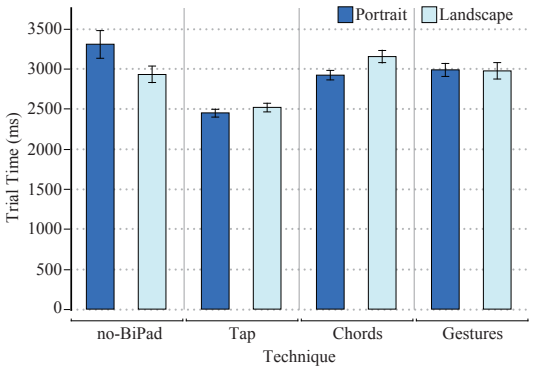


Figure 7. Mean *Trial Time* for each TECHNIQUE by ORIENTATION.

Data Collection. We videotaped each trial and recorded three temporal measures: (i) trial time: from the appearance of the first target to final target selection; (ii) BiPad reaction time: from the appearance of the red target to the first touch in the BiPad area; and (iii) BiPad completion time: from the appearance of the red target to the successful execution of the BiPad interaction. Comfort ratings used a 5-point Likert scale (1 = very uncomfortable; 5 = very comfortable).

RESULTS

We conducted a full factorial ANOVA and handled 'participant' as a random variable, using the standard repeated measures REML technique from the JMP statistical package.

Q1: Bimanual BiPad vs. one-handed interaction

We compared the mean trial time of BiPad techniques to the no-BiPad control condition, using the TECHNIQUE×ORIENTATION ×Random(PARTICIPANT) ANOVA model. We found a significant effect for TECHNIQUE ($F_{3,33} = 16.16$, $p < 0.0001$) but no effect for ORIENTATION ($F_{1,11} = 0.30$, $p = 0.60$). However, we did find a significant interaction effect between TECHNIQUE and ORIENTATION ($F_{3,33} = 8.23$, $p = 0.0003$).

This can be explained by the faster performance in landscape mode for the one-handed no-BiPad condition (Fig. 7): participants performed 11.4% faster ($F_{1,11} = 4.6$, $p = 0.04$) because the distance to reach the button is shorter. Thus, while bimanual taps are significantly faster than the control condition for both orientations (25.9% in portrait and 14% in landscape), bimanual gestures and chords are only significantly faster than no-BiPad in portrait mode (10.4% and 11.7% resp.). In landscape mode, the differences between no-BiPad and bimanual gestures and chords are not significant.

Bimanual taps are significantly faster than bimanual gestures and chords in both device orientations (17.3% and 16.1% in portrait, 14.7% and 19.7% in landscape). Participants significantly preferred bimanual taps (3.5) over bimanual chords (3.3) and gestures (2.7) ($F_{2,22} = 17.5$, $p < 0.0001$). Overall, BiPad techniques were more efficient than the one-handed technique we compared them with.

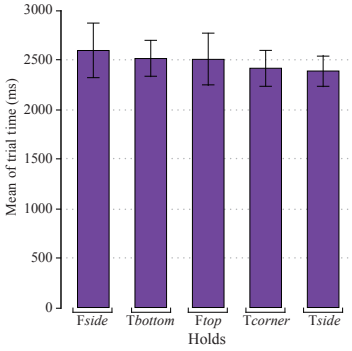


Figure 8. *Tap* performance according to HOLD.

Q2: BiPad tradeoffs: HOLD×ORIENTATION by TECHNIQUE

BiPad Taps

We ran an ANOVA with the model HOLD×ORIENTATION× Random(PARTICIPANT) on trial time for BiPad taps. We found significant effects for HOLD and ORIENTATION ($F_{4,44} = 3.10$, $p = 0.02$ and $F_{1,11} = 5.37$, $p = 0.04$) and no interaction effect ($F_{4,44} = 0.65$, $p = 0.63$).

For HOLD, Tukey post-hoc tests revealed only one significant result: placing the fingers on the right is slower than placing the thumb on the left side of the tablet for right-handed participants (see Fig. 8). For ORIENTATION, a Student's t-test reveals that portrait is significantly faster (LSM $= 2447.31\,ms$) than landscape (LSM $= 2515.99\,ms$).

Performance among bimanual taps is very similar across conditions, making them suitable for all ten holds. The only significant difference is between fingers and thumbs with a side hold. However, although the F_{side} hold is slightly slower, participants preferred it to the T_{side} hold: fingers are more stable than thumbs and cause less fatigue.

BiPad Gestures

As we discarded the two bimanual holds with fingers placed on the right and top of the device in landscape mode, we examined trial Time for each ORIENTATION condition separately for the remaining eight holds. HOLD has a significant effect on the performance time in both portrait ($F_{4,44} = 4.14$, $p = 0.01$) and landscape ($F_{2,22} = 4.75$, $p = 0.02$).

In *Portrait*, post-hoc Tukey HSD tests show that, for a right-handed user, performing gestures with the fingers on the right side of the device is significantly slower than with the thumb on the left side (Fig. 9a). Participants preferred performing gestures with the fingers or with the thumb on the side of the device. In fact, gestures are most difficult to perform when the support hand is placed on the top or bottom of the device when held in portrait mode.

In landscape, where only the *Thumb* placements were tested, performing gestures while supporting the tablet with the thumb on the bottom of the device is significantly faster than in the corner (Fig. 9b). However, since gestures were performed in both ORIENTATION conditions with the thumb, we also compared performance according to thumb holds in both orientation conditions (HOLD×ORIENTATION×Random(PARTICIPANT)).

We found no significant effect of HOLD and ORIENTATION but a significant interaction effect for HOLD×ORIENTATION ($F_{2,22} = 15.08$, $p < 0.0001$). This is because performing gestures with the thumb is significantly faster in portrait, when the support hand is on the side, but significantly slower when the thumb is on the bottom, in which case landscape is faster. The difference between orientations is not significant when the thumb is placed in the corner (Fig. 9c).

The latter effect is interesting and can be explained by the principle of a lever. The greater the distance between the balance point and the most distal support link, the heavier the tablet is perceived. This is considered less comfortable and users find it more difficult to perform gestures. The exception is when the thumb is in the corner: the distal point of the support is equally close to the tablet's balance point in both orientations, thus the two holds are not significantly different. This explanation correlates with the participants' comfort ratings and comments. They preferred to perform gestures with the thumb on the side in portrait and on the bottom in landscape but had no preference for orientation when the thumb is in the corner. Compared to other BiPad techniques, however, gestures were perceived as relatively uncomfortable and practical only for rapid or occasional use.

BiPad Chords

We ran an ANOVA with the model HOLD×ORIENTATION×CHORD TYPE×Random(PARTICIPANT) on *Trial Time*. We found no significant effects of HOLD and ORIENTATION and no interaction effects. For CHORD TYPE, we found a significant effect ($F_{2,22} = 9.09$, $p = 0.01$): holding the index finger down together with the middle finger is significantly faster ($2875\,ms$) than holding down three fingers ($3095\,ms$) or the index and ring finger together ($3131\,ms$).

Participants did not express any significant comfort preferences with respect to chords. However, some participants reported that chords are difficult to perform at the top of the device, especially in landscape mode, due to tension in the arm. Two users could only perform two-finger chords since their third finger could not easily reach the screen.

DISCUSSION

Our results demonstrate not only that hand-held touch tablets can support bimanual interaction, but that it outperforms all tested uni-manual interactions in almost all of our experimental conditions. We created a set of 22 bimanual interaction techniques that combine the ten holds identified in the preliminary study with bimanual taps (10), chords (4) and gestures (8). These offer users trade-offs in performance, comfort and expressive power; BiPad lets users transition smoothly among them.

In the future, we hope to develop the predictive power of the BiTouch design space, building upon our existing understanding of the physical characteristics of the human body and exploring its relationship to hand-held interactive devices. For example, we observed that bimanual taps (in both orientations) and bimanual gestures (in Portrait mode) are significantly faster in holds with thumbs on the side (T_{side}) compared to holds with fingers on the side (F_{side}).

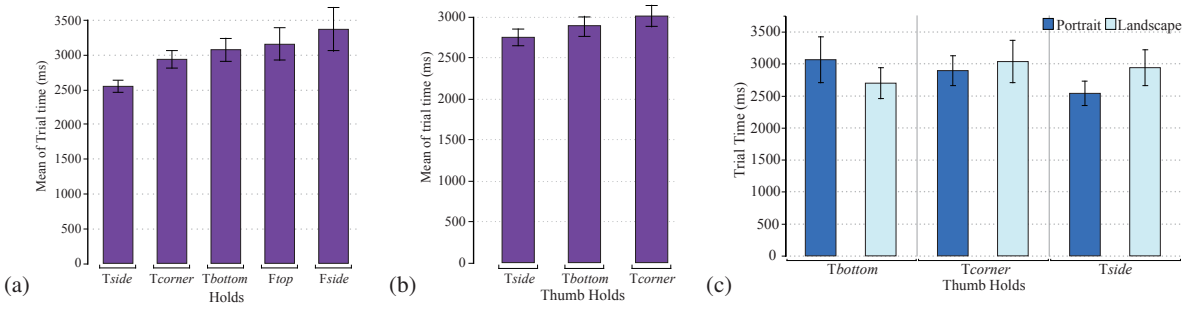


Figure 9. *Gesture* performance according to HOLD (a) in *Portrait*, (b) in landscape, and (c) for the *Thumb* according to HOLD and ORIENTATION.

In contrast, T_{side} is perceived as less comfortable than F_{side}. If we examine thumbs and fingers, we see that the T_{side} hold leaves only two joints available for interaction, whereas the F_{side} hold has three. This suggests that, all other things being equal, performance will be better with interaction techniques that offer a wider range of movement. Additional research is necessary to verify if this prediction obtains for other holds.

We can also use the BiTouch design space to help us understand differences in perceived comfort. One hypothesis is that comfort is correlated with perceived weight, which is determined by both the location of support in the kinematic chain and the orientation of the tablet. If we examine the two holds, we see that the support link for the F_{side} hold, the forearm, is longer than that for the T_{side}, the palm. On the other hand, the former hold restricts movement more than the latter. This suggests two open research questions:

1. Does performance decrease and comfort increase with longer support links?
2. Does performance decrease and comfort increase with increased support link mobility?

We also observed a major effect of tablet orientation in some conditions, such as bimanual gestures. The previously mentioned lever effect plays a role here. If we view the tablet as an extension of the support link, we can estimate its perceived weight based on the distance from the most distal element of the support link to the balance point of the tablet. This raises the question:

3. Do performance and comfort increase as the distance to the balance point decreases?

Finally, multitouch tablets exist in a variety of different shapes, sizes, and weights. We used the popular iPad1 for the first experiment. However, when the iPad2 was released, we replicated the experiment with six participants, and found no significant differences despite the 30% reduction in weight. Of course different tablet designs might affect the performance and comfort of BiPad bimanual interaction. In the future, we plan to extend the BiTouch design space to include device-specific characteristics to increase its predictive power.

SUMMARY AND CONCLUSIONS

We investigated how to introduce effective bimanual interaction into hand-held tablets. We began with a preliminary study that identified support positions while sitting, standing and walking. We found that, although novices found it difficult to come up with effective holds, more experienced users produced ten unique holds that can be adapted to support bimanual interaction. We also found that users do not seek a single, optimal hold, but instead prefer to modify their holds over time, to reduce fatigue and increase comfort. We concluded that the design challenge was not to create a single bimanual technique but rather to create a set of equally comfortable and effective techniques.

We next examined the theoretical basis of the ten observed holds and presented the BiTouch design space, based on Guiard's kinematic chain model. We argue that we can understand bimanual interaction with hand-held devices by examining how three functions – framing, support and interaction – are distributed along the kinematic chain. Our goal is to offer *descriptive*, *predictive* and *generative* power, and BiTouch offers a good start: we can describe all of the unimanual and bimanual interaction techniques observed in the preliminary study; we can make informal predictions about which factors affect performance, comfort and expressive power; and we have generated a set of bimanual interaction techniques that offer different trade-offs with respect to the above:

- *Bimanual Taps*: one finger or thumb taps the screen,
- *Bimanual Chords*: several fingers touch the screen,
- *Bimanual Gestures*: a finger or thumb slides on the screen.

We implemented these techniques in BiPad, a user interface toolkit we made for designing bimanual interaction with off-the-shelf hand-held tablets[1], and developed three working applications in which the non-dominant hand can modify the dominant hand's interaction using taps, chords or gestures.

We tested these interaction techniques in a controlled experiment for each of the five holds and two orientations found in the preliminary study. Bimanual taps are faster than reaching on-screen buttons with the dominant hand only, regardless of tablet orientation or hold. However, they can handle at most three buttons, since the pinky cannot reach the screen and the range of thumb movement is limited. Bimanual chords and gestures offer a richer vocabulary for shortcuts to off-screen functions, but have their own limitations. Chords require multiple fingers and gestures are restricted in landscape to thumb

[1]The BiPad toolkit is freely available at http://insitu.lri.fr/bipad

holds. The BiTouch analysis helps explain why bimanual chords and gestures are faster only in portrait orientation: the position of the support link in the kinematic chain directly affects which fingers or thumbs are available for interaction and the number of available degrees of freedom.

Together, the BiTouch design space and the BiPad toolkit offer developers a richer understanding of bimanual interaction and a practical approach for adding bimanual interaction to hand-held tablets. Future work will explore how we can generate new possibilities for bimanual interaction on a range of devices in different mobile settings.

ACKNOWLEDGMENTS

We wish to thank Michel Beaudouin-Lafon for many fruitful discussions and suggestions for improving this article. We also thank the anonymous reviewers for their helpful comments, as well as the participants for their time and effort.

REFERENCES

1. Balakrishnan, R., and Hinckley, K. The role of kinesthetic reference frames in two-handed input performance. In *UIST '99*, ACM (1999), 171–178.
2. Balakrishnan, R., and Hinckley, K. Symmetric bimanual interaction. In *CHI '00*, ACM (2000), 33–40.
3. Beaudouin-Lafon, M., Mackay, W. E., Andersen, P., Janecek, P., Jensen, M., Lassen, M., Lund, K., Mortensen, K., Munck, S., Ravn, K., Ratzer, A., Christensen, S., and Jensen, K. Cpn/tools: revisiting the desktop metaphor with post-wimp interaction techniques. In *CHI '01*, ACM (2001), 11–12.
4. Benko, H., Wilson, A. D., and Baudisch, P. Precise selection techniques for multi-touch screens. In *CHI '06*, ACM (2006), 1263–1272.
5. Bier, E. A., Stone, M. C., Pier, K., Buxton, W., and DeRose, T. D. Toolglass and magic lenses: the see-through interface. In *SIGGRAPH '93*, ACM (1993), 73–80.
6. Brandl, P., Forlines, C., Wigdor, D., Haller, M., and Shen, C. Combining and measuring the benefits of bimanual pen and direct-touch interaction on horizontal interfaces. In *AVI '08*, ACM (2008), 154–161.
7. Fonville, A., Choe, E. K., Oldham, S., and Kientz, J. A. Exploring the use of technology in healthcare spaces and its impact on empathic communication. In *IHI '10*, ACM (2010), 497–501.
8. Froehlich, J., Wobbrock, J. O., and Kane, S. K. Barrier pointing: using physical edges to assist target acquisition on mobile device touch screens. In *Assets '07*, Assets '07, ACM (2007), 19–26.
9. Guiard, Y. Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model. *J. Motor. Behav. 19* (1987), 486–517.
10. Hinckley, K., Pausch, R., Proffitt, D., and Kassell, N. F. Two-handed virtual manipulation. *ACM TOCHI 5* (1998), 260–302.
11. Huot, S., and Lecolinet, E. Focus+context visualization techniques for displaying large lists with multiple points of interest on small tactile screens. In *INTERACT'07*, Springer-Verlag (2007), 219–233.
12. Kabbash, P., Buxton, W., and Sellen, A. Two-handed input in a compound task. In *CHI '94*, ACM (1994), 417–423.
13. Karlson, A. K., and Bederson, B. B. One-handed touchscreen input for legacy applications. In *CHI '08*, ACM (2008), 1399–1408.
14. Kin, K., Agrawala, M., and DeRose, T. Determining the benefits of direct-touch, bimanual, and multifinger input on a multitouch workstation. In *GI '09*, Canadian Information Processing Society (2009), 119–124.
15. Latulipe, C., Mann, S., Kaplan, C. S., and Clarke, C. L. A. symspline: symmetric two-handed spline manipulation. In *CHI '06*, ACM (2006), 349–358.
16. Leganchuk, A., Zhai, S., and Buxton, W. Manual and cognitive benefits of two-handed input: an experimental study. *ACM TOCHI 5* (1998), 326–359.
17. Mackay, W. E. Which interaction technique works when?: floating palettes, marking menus and toolglasses support different task strategies. In *AVI '02*, ACM (2002), 203–208.
18. Moscovich, T., and Hughes, J. F. Indirect mappings of multi-touch input using one and two hands. In *CHI '08*, ACM (2008), 1275–1284.
19. Oulasvirta, A., and Bergstrom-Lehtovirta, J. Ease of juggling: studying the effects of manual multitasking. In *Proc.*, CHI '11, ACM (2011), 3103–3112.
20. Pascoe, J., Ryan, N., and Morse, D. Using while moving: Hci issues in fieldwork environments. *ACM TOCHI 7* (2000), 417–437.
21. Roth, V., and Turner, T. Bezel swipe: conflict-free scrolling and multiple selection on mobile touch screen devices. In *CHI '09*, ACM (2009), 1523–1526.
22. Roudaut, A., Huot, S., and Lecolinet, E. Taptap and magstick: improving one-handed target acquisition on small touch-screens. In *AVI '08*, ACM (2008), 146–153.
23. Schwesig, C., Poupyrev, I., and Mori, E. Gummi: a bendable computer. In *CHI '04*, ACM (2004), 263–270.
24. Scott, J., Izadi, S., Rezai, L. S., Ruszkowski, D., Bi, X., and Balakrishnan, R. Reartype: text entry using keys on the back of a device. In *MobileHCI '10*, ACM (2010), 171–180.
25. Sugimoto, M., and Hiroki, K. Hybridtouch: an intuitive manipulation technique for pdas using their front and rear surfaces. In *MobileHCI '06*, ACM (2006), 137–140.
26. Wigdor, D., Forlines, C., Baudisch, P., Barnwell, J., and Shen, C. Lucid touch: a see-through mobile device. In *UIST '07*, ACM (2007), 269–278.
27. Wimmer, R., and Boring, S. Handsense: discriminating different ways of grasping and holding a tangible user interface. In *TEI '09*, ACM (2009), 359–362.
28. Wobbrock, J. O., Myers, B. A., and Aung, H. H. The performance of hand postures in front- and back-of-device interaction for mobile computing. *Int. J. Hum.-Comput. Stud. 66* (2008), 857–875.
29. Wu, M., and Balakrishnan, R. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. In *UIST '03*, ACM (2003), 193–202.
30. Yang, X.-D., Mak, E., Irani, P., and Bischof, W. F. Dual-surface input: augmenting one-handed interaction with coordinated front and behind-the-screen input. In *MobileHCI '09*, ACM (2009), 5:1–5:10.

Using Rhythmic Patterns as an Input Method

Emilien Ghomi ghomi@lri.fr
Guillaume Faure gfaure@lri.fr
Stéphane Huot huot@lri.fr
Olivier Chapuis chapuis@lri.fr
Michel Beaudouin-Lafon mbl@lri.fr

Univ Paris-Sud (LRI)
F-91405 Orsay, France

CNRS (LRI)
F-91405 Orsay, France

INRIA
F-91405 Orsay, France

ABSTRACT

While interaction techniques that use the temporal dimension have been used for a long time, such as multiple clicks or spring-loaded widgets, more advanced uses of *rhythmic patterns* have received little attention in HCI. Using such temporal structures to convey information can be particularly useful in situations where the visual channel is overloaded or even not available. In this paper we introduce *Rhythmic Interaction* as the use of rhythms for input. We report the results of two experiments that show that (i) rhythmic patterns can be efficiently reproduced by novice users and recognized by computer algorithms, and (ii) rhythmic patterns can be memorized as efficiently as traditional shortcuts when associating them with visual commands. Overall, these results demonstrate the potential of Rhythmic Interaction and open the way to a richer repertoire of interaction techniques.

ACM Classification Keywords

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

Author Keywords

Rhythm; taping; hotkeys; learning; patterns; Morse code

INTRODUCTION

Rhythm plays an important role in our everyday life. Temporal patterns are of course critical in experiencing music, but they also underlie periodic actions such as walking, breathing or chewing, and they are even necessary for understanding speech prosody. Rhythm is so deeply embedded in our experience of living that it can be used to cure some diseases such as stress or sleep disorders [29].

While perceiving and reproducing rhythm is recognized as a fundamental human ability by physiologists and neuropsychologists, it is still underused as an interactive dimension in HCI. In common desktop environments, interaction relies heavily on manipulating graphical widgets, simple mouse clicks and keyboard shortcuts. This basic vocabulary, however, is often extended by using *spatial* or *temporal* features, as with mouse gestures or multiple clicks.

E. Ghomi, G. Faure, S. Huot, 0. Chapuis, and M. Beaudouin-Lafon. Using Rhythmic Patterns as an Input Method. In CHI '12: Proceedings of the SIGCHI Conference on Human Factors and Computing Systems, 1253-1262, ACM, 2012. http://doi.acm.org/10.1145/2207676.2208579

Although the spatial dimension has been the focus of much HCI research on interaction techniques based on hand postures or gestures, e.g. [1, 3, 16], the temporal dimension has received little attention so far. We propose to use rhythm as an input method and introduce *Rhythmic Interaction* as a complementary way to control interactive systems. Rhythmic Interaction can be used in any event-driven environment for a variety of input modalities: clicking the mouse, hitting keyboard keys or a touch-sensitive surface, moving a motion-sensing device, etc. However, it has competitive advantages for tactile screens, since it requires less screen space than gestural interaction and no visual attention [33]. This article presents a first exploration of the design space of Rhythmic Interaction in order to address the following questions:

- *Feasibility*. Even if perceiving and performing rhythm is quite natural, are users able to reproduce, learn and memorize patterns? Can they use them to trigger commands?
- *Interaction design*. The number of possible rhythmic patterns is virtually infinite and they can be presented in several ways. Which patterns make sense for interaction and how to design a vocabulary? What feedback helps executing and learning patterns?
- *Technical issues & Integration*. Like most continuous high-level input methods, e.g. voice, marks, gestures, Rhythmic Interaction relies on a recognizer to segment and interpret user input. How to design effective recognizers that do not require training?

In the rest of this paper, we survey related work and then define a framework for Rhythmic Interaction, narrowing the scope of our study to vocabularies of rhythmic patterns that are relevant in the context of HCI. Then, we report on two experiments where the patterns are rhythmic sequences of taps performed on a tactile trackpad to trigger commands. The first one tests the ability of novice users to reproduce individual patterns, while the second one compares the ability of users to memorize the association of commands to rhythmic patterns vs. keyboard shortcuts. We also describe the recognizers that we created for these two experiments, and draw some conclusions regarding the design of pattern vocabularies and appropriate feedback.

BACKGROUND & RELATED WORK

The literature in cognitive science has studied the perception, reproduction and use of rhythm from several perspectives: physiology, e.g., perception and action [13, 19], knowledge and learning, e.g., language [24], artistic applications, e.g., music [23], etc. Two major studies on the psychology

of rhythm in music are reported by Fraisse [12] and by Clarke [7]. In this section, we give an overview of the literature in cognitive science that is most relevant to Rhythmic Interaction and then focus on the few studies of using time and rhythm to control interactive systems.

Rhythm in Music and Games

For musicians, rhythm is one of the most important features in music, together with melody and harmony. Many traditional forms of music combine simple rhythmic structures played in parallel to build up higher level aggregated rhythms, called "polyrhythms" [2]. From a cognitive point of view, these practices suggest that the cognitive load required to deal with an elementary rhythmic pattern is light enough to allow their combination in more complex structures.

Highly trained musicians are able to create and play an incredible amount of rhythmic variations. Simple patterns, however, can be reproduced by everyone, as illustrated by the success of popular musical games such as Guitar Hero, TapTap or Donkey Kong, where rhythmic structures are recognized and reproduced by non-musicians of all ages.

Rhythm in Cognitive Sciences

In physiology and neuropsychology, numerous studies report that humans have a natural perception of rhythm, thanks to rhythmic mechanisms that are involved in the internal functioning of our organism (heart beat, sleep cycles, etc.) [13] and their relation with periodic external phenomena (day/night, seasons, etc.). When listening to music, we constantly try to infer the beat, i.e., to perceive regular and salient events, or to group events into rhythms [18]. In fact, we systematically try to perceive rhythm even when none is present [25] or when being told to avoid it [12].

Rhythm perception is deeply related to the motor system [5]. Since childhood, humans are used to tap their feet, clap their hands, snap their fingers and move in synchrony with music. These activities are common and seem simple, even though they involve complex rhythmic structures. Outside music, periodic activities of different frequencies are pervasive in our everyday life. For example, chewing or walking are known to have universally preferred rates [13, 19].

While these studies attempt to explain how and why we perceive and produce periodicities, they rarely deal with the reproduction and memorization of rhythmic patterns associated to tasks that we address in this article.

Rhythm as an Input Method in HCI

Rhythm is built on the temporal dimension, which is commonly used in interactive software. For example, long clicks are often distinguished from short clicks to trigger different commands based on temporal criteria. The concept of "dwelling"—freezing the interaction for a short amount of time—is also used to segment gestural interaction [15] or to explicitly switch mode [10]. Rhythmic Menus [22] successively highlight items at a given rate while the mouse button is pressed. When the user releases the button, the current item is selected.

Some techniques are based on the temporal grouping of events. Double click is the simplest and most common case, but some studies also explored rhythmic motion: Motion Pointing [11] assigns different periodic motions to graphical objects in a scene or items in a pie menu; The user selects the object or menu item of interest by performing the corresponding motion. In Cyclostar [20], the user controls continuous parameters, such as the speed of zooming, by performing elliptical oscillatory gestures. The rate of the circling motion controls a parameter of the resulting command.

In the above cases, rhythmic aspects are reduced to periodicity. To the best of our knowledge, only a few techniques involve the reproduction of rhythmic patterns. Five-key [31] is a text entry system based on rhythmic sequences, where letters can be entered with only five keys. However, efficiency and learning were not studied systematically. In [9], tempo reproduction is used to select a particular song in a music library by tapping on a mobile device or shaking it. But relying only on tempo raises some scalability issues that were not assessed. Finally, Tapsongs [33] is an alternative to textual passwords where users tap a rhythmic pattern that they have registered with the system for authentication.

MOTIVATION

Our goals are more general than Five-key and Tapsongs: we want to design vocabularies of rhythmic patterns that users can learn easily and perform reliably in order to trigger commands. This approach is somewhat similar to the use of Morse code for encoding characters. However, the design of Morse code was driven by information theoretic issues rather than usability, and while early computers were able to decode human-produced Morse code [4], it has rarely been used in HCI [6]. Our objective is to propose a comprehensive framework to design rhythmic patterns for interaction, with efficient recognizers that do not need training.

Advantages of Using Rhythms for Input

The design of new techniques based on Rhythmic Interaction is not the main focus of the present article. However, we have identified a number of potential advantages of using rhythm to interact with computer systems. First, as evidenced by research in Cognitive Science, there is a direct correspondence between performing a rhythm (action) and listening to a rhythm (potential audio stimulus and feedback). Second, rhythms can be performed in a variety of situations: while performing a rhythm requires as little as a single degree of freedom of one finger, many movements can be performed rhythmically and captured using different sensors, e.g., tapping fingers, tapping feet, or nodding the head.

Gestural interaction typically requires space to perform the gestures, and often interferes with the display space on a small touchscreen. By contrast, Rhythmic Interaction only uses temporal features. Rhythms can be performed on a small area of a tactile device, even in an eye-free context.

Finally, rhythmic structures can be designed in a hierarchical way. By using common prefixes among different patterns, a natural hierarchy emerges that can be internalized by users, facilitating memorization and recall.

Applications of Rhythmic Patterns

Rhythmic patterns are not meant to replace more conventional command input methods. Instead, it is an alternative that may be more adapted to specific situations, such as eye-free operation. It is also a way to enhance existing methods with a richer vocabulary. For example rhythmic patterns could give access to a restricted set of commands such as speed-dialing a phone number, navigating an e-book or switching mode in an application.

In some situations, rhythmic patterns can simplify interaction. For example, bookmarks, menu items or contacts are often organized hierarchically. Rhythmic patterns could match this hierarchy or provide an alternate hierarchy such as organizing contacts by their first name. Also, since rhythmic patterns can be performed without visual attention, they can be used with a tactile device in the pocket or while driving, or in the dark, e.g. to shut down an alarm clock, or even with devices that do not have a display.

Rhythmic Interaction also offers novel solutions to well-known problems. Tapping on the back of a hand-held device can be captured without extra sensors, thanks to built-in accelerometers or microphones [28]. For example, a rhythmic pattern performed while receiving a phone call could add the caller to the contact list, or display extra information such as battery life or signal level. Patterns could also be performed with the non-dominant hand or another part of the body such as the feet [30], to switch mode or ignore an incoming call.

RHYTHMIC PATTERNS FOR INTERACTION

Our definition of a rhythmic pattern comes from music: The elementary structure in music is called a *motif*, which is defined as a "melodic, rhythmic, or harmonic cell" [21]. A *rhythmic motif* represents the temporal structure of a set of notes and consists of the relative durations of notes and silences. Notes and silences can have eight different durations in standard musical pieces, and motifs can contain many notes, leading to a huge number of possible rhythmic motifs.

Considering the number of commands and actions often used when interacting with computers, such an expressive power is not required. Therefore we propose a restricted definition of *rhythmic pattern* (or simply *pattern*) more adapted to HCI. A rhythmic pattern is a sequence of taps[1] and breaks whose durations are counted in *beats*. The beat is the basic unit of time and its duration is defined below). We define the complete set of possible patterns with the following rules:

- Taps can be of three types: *impulse* (a hit on a touch device or a click), *short* tap (one beat) or *long* tap (two beats). A tap starts at the beginning of a beat, and there cannot be more than one tap per beat.
- Breaks can be of two types: *short* (one beat) or *long* (two beats). A pattern cannot begin or end with a break, and there cannot be two successive breaks.

This definition of taps and breaks is based on our empirical observation that computer users are familiar with the distinction between instantaneous and long clicks or taps. By adding a third duration and by taking breaks into account, we offer designers more possibilities for selecting a set of patterns among the possible combinations. In comparison to Morse code, we do not need the "intra-character", "inter-character" and "inter-word" breaks that are specific to the coding of language, and we do not allow more than one tap per beat.

[1]The word "tap" reflects our focus on using a tactile device for input.

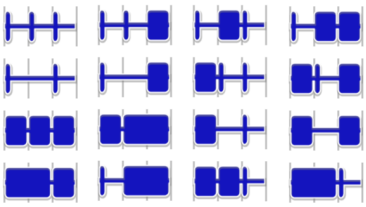


Figure 1. The 16 three-beat patterns defined by our rules. Each rectangle represents a tap. The thin gray lines show the beats.

The *length* of a pattern is the sum of the durations of its taps and breaks. To simplify reproduction and memorization, we focus on patterns between two and six beats long. The rules above define 5 two-beat patterns, 16 three-beat patterns (Figure 1), 53 four-beat patterns, 171 six-beat patterns and 554 six-beat patterns. By comparison, the total number of patterns with n taps is 3^{2n-1}, i.e. 199,290 patterns with up to six taps.

In this entire study, beats occur at the tempo of 120 BPM (2Hz). Thus, the onsets of two consecutive taps are separated by at least 500 ms, i.e. a beat is half a second. This corresponds to a common tempo of human motor actions, e.g. walking [19], and of contemporary music [23].

As a first step, we only consider rhythmic patterns performed by tapping on a touch-sensitive surface. While keyboards, accelerometers [17, 9] or eye blinks [32] can probably be used for Rhythmic Interaction, it is out of the scope of this article. We also do not address the segmentation of patterns from other input. Simple solutions that should be tested include segmenting in time by preceding each pattern with a specific short sequence of taps, or segmenting in space by performing patterns on a specific location of a device.

A key aspect of this research is to design a recognizer that can reliably identify the patterns produced by users. In a first experiment, we used a *structural* recognizer to assess users' ability to produce patterns accurately. Based on the results, we designed a *pattern classifier* that accounts for user inaccuracies while still discriminating the patterns in the vocabulary. This classifier was used in a second experiment where we assessed users' ability to memorize associations between patterns and commands in an applicative context.

EXPERIMENT 1: RHYTHMIC PATTERN REPRODUCTION

In order to assess the potential of Rhythmic Interaction, we conducted a first experiment where novice users were asked to replicate patterns presented to them in visual and/or audio form by tapping on a touch surface. The goal was to assess the accuracy of the reproduction and to compare the effects of several feedback mechanisms while performing patterns.

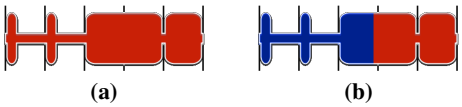


Figure 2. The stimulus in Experiment 1. A static representation is displayed (a) and fills up in synchrony with the audio playback (b).

Figure 3. Visual feedback while tapping a pattern.

Recognizer

The recognizer that we designed for this experiment is based on the above rules for defining the patterns. It first extracts the rhythmic structure as a list of taps and breaks and infers their respective types (impulse, short and long) using autonomous heuristics. The reconstructed pattern is then checked against the vocabulary used for the study.

In order to identify the type of every tap and break in the sequence, the recognition algorithm uses k-means clustering iterated 500 times on duration values. The algorithm builds clusters corresponding to the possible types of taps and breaks: impulse, short and long[2]. A minimum distance of 200ms between the duration clusters is enforced, corresponding to a maximum tempo for the pattern to be recognized. If two clusters are closer than that distance, they are merged and will be recognized as a single tap type. Thus, if the pattern is performed too fast, events of different types may be confused by the recognizer. For cluster identification, the reference durations for short and long taps or breaks are set to 500ms and 1000ms respectively, and the maximum duration of an impulse or release is set to 180ms.

After clustering, breaks that correspond to the rest of a beat after an impulse are removed from the reconstructed pattern. The resulting pattern is then looked up in the vocabulary to check if it matches the stimulus provided to the participant.

Note that this recognizer is intentionally very strict, in order to assess the participants' ability to precisely reproduce the patterns. In particular, if the reconstructed pattern is not in the vocabulary, the recognizer will systematically return an error. With minimal knowledge about our definition of rhythmic patterns, the algorithm is able to identify the type of every tap and break in a sequence even in tricky situations, such as when there is just one type of events. Thanks to clustering, the recognizer adapts to the tempo (the overall tempo can be inferred by comparing the centroids of the clusters.)

Apparatus and Participants

The experiment was implemented in Java and conducted on a 13" Apple MacBook (Intel processor). Participants tapped the rhythmic patterns on the embedded multitouch trackpad.

Twelve unpaid volunteers (six female) participated in this experiment, with age ranging from 23 to 53 (mean 29, median 27). Five of them had never practiced music.

Stimulus

The pattern to reproduce is presented to the participant with a stimulus combining a static graphical representation of the pattern, visual animation and audio (Figure 2). The visual stimulus is a stationary shape depicting the whole rhythmic pattern, where each rectangle represents an event (Figure 2a). This shape is then progressively filled (Figure 2b) in synchrony with audio playback. Beats are marked with thin gray lines to visualize the durations of events.

For audio playback and animation, impulses last 125ms and the tempo is set to 120 BPM or 2Hz (500ms period). This value is above the "synchronization threshold" [27] for both visual and auditive stimulus, ensuring that participants can perceive and perform it accurately. The audio stimulus is a 440Hz A, played by the General MIDI Instrument "English Horn" and held at a constant sound level. We chose this sound as it is soft enough for the subjects to endure during the experiment, but has clear onset and release.

Feedback

Participants are presented with 4 input FEEDBACK conditions while reproducing rhythmic patterns. The $Audio$ feedback plays the same sound as the stimulus as long as the participant is touching the surface of the trackpad. The $Visual$ feedback is based on the graphical representation of the stimulus. The rectangles representing the events appear dynamically while the subject is tapping on the trackpad (Figure 3). The $AudioVisual$ feedback combines the two previous methods, and there is no feedback at all in the $None$ condition.

The $Audio$, $Visual$ and $AudioVisual$ feedback methods are expected to help learning, e.g. in novice mode. Conversely, the $None$ condition corresponds to the situation where an expert user is performing patterns in an eyes-free manner without audio feedback.

Vocabulary

For this experiment, we selected 30 rhythmic patterns among the 799 patterns with two to six beats generated by the rules described earlier. This vocabulary (Figure 4) contains 4 two-beats patterns, 8 three-beats patterns, 8 four-beats patterns, 6 five-beats patterns and 4 six-beats patterns. We explicitly featured fewer patterns for the extreme situations (two, five and six beats). Among the patterns with the same duration, we tried to balance the number of events. For example, for the 8 four-beat patterns, 2 contain two events, 3 contain three events and 3 contain four events.

Task

A trial consists in reproducing a rhythmic pattern according to the FEEDBACK condition, right after it is presented twice in a row. The participant performs the pattern by tapping on the trackpad with the index finger of her dominant hand. The recognizer then computes the temporal structure of the input and matches it with that of the stimulus. At the end of the trial, the participant is notified about the success or the failure of the match before advancing to the next trial.

[2]A break with a duration of an impulse is called a *release*. It occurs between two adjacent taps.

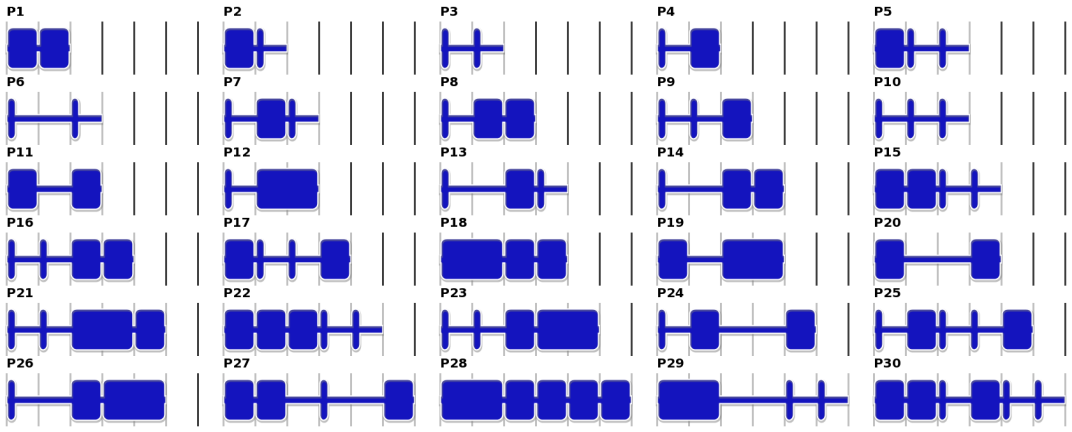


Figure 4. Vocabulary used in the first experiment.

Design and Procedure

The experiment is a 2×30 within-subject design with factors: (i) FEEDBACK: *Audio*, *Visual*, *AudioVisual* and *None*; and (ii) PATTERN: P1 – P30 (Figure 4).

At the beginning of the session, each FEEDBACK condition is introduced to the participant with a short block of 15 random trials. Then, the participant is asked to perform two warm-up blocks of 15 trials in the *AudioVisual* feedback condition, which we hypothesize provides the best feedback to become familiar with the task. The 3 first trials of the first warm-up block are performed by the experimenter to demonstrate the feedback condition to the participant. The second warm-up block is interrupted if the participant reports to be confident enough to start the experiment.

During the main session, measured trials are grouped into blocks according to the FEEDBACK factor. The presentation order for FEEDBACK is counterbalanced across participants with a Latin square. Within each block, the 30 patterns are repeated twice in randomized order. A practice block of 15 randomly selected patterns is performed prior each measured block and participants are allowed to have breaks between and in the middle of each block. Thus, we collected 12 participants $\times$ 4 FEEDBACK $\times$ 30 PATTERN $\times$ 2 repetitions $=$ 2880 measured trials. Participants were instructed to be as accurate as possible by paying attention to the discrimination of different types of taps and breaks. Each participant took about one hour to complete the sessions, after which they were asked to rank the feedback methods according to the difficulty of the task on a 5-point Likert's scale.

Quantitative Results

The overall success rate is 64.3%. This may seem low, but recall that our recognizer is deliberately very strict regarding the temporal structure of patterns, and that it can recognize all 799 patterns with two to six beats, not just the 30 patterns in the study. The precise reproduction of the rhythmic patterns in the study is similar to playing a percussion instrument, a task that musicians can take years to master.

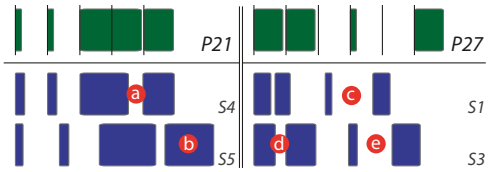


Figure 5. Two stimuli (P21 and P27) with reproductions errors by subjects of Experiment 1. S4: the last break is too long (a); S5: the last tap is too long (b); S1: the last break is too short (c); S3: the first break is too long (d), the last break is too short (e).

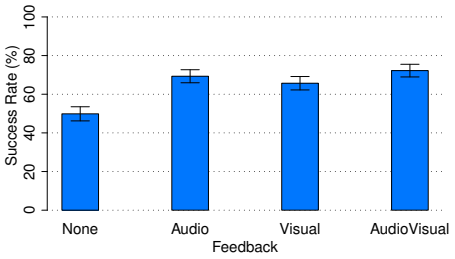


Figure 6. Success rate for each FEEDBACK condition.

Figure 5 shows typical reproduction errors by study participants, such as release breaks that are too long and recognized as short breaks, or breaks or taps that are too similar to be separated during clustering. Interestingly, errors seem more frequent with breaks than with taps, which is consistent with the finding that users tend to be more precise when performing notes than pauses [26].

A one-way ANOVA for FEEDBACK (with participant as a random variable) reveals a significant effect on success rate ($F_{3,33} = 15.4$, $p < 0.0001$). This effect can be observed in Figure 6[3]. Post-hoc t-tests with Bonferroni correction show that the *None* condition is significantly worse than all other feedback conditions. It is not surprising that the absence of feedback while performing the pattern significantly degrades the accuracy of rhythm reproduction.

[3] In all figures, error bars show the 95% confidence interval.

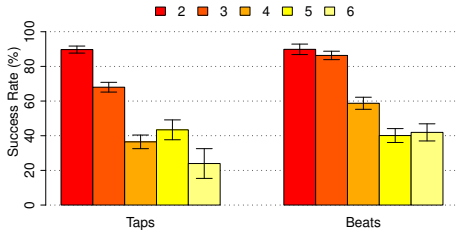


Figure 7. Success rate by number of taps and by length in beats.

Regarding PATTERN, a one-way ANOVA reveals a significant effect on success rate ($F_{29,319} = 25.1, p < 0.0001$). We observe a large deviation of the success rate for some patterns: from 16% with P27 to 98% for P10. Fifteen patterns have a success rate of at least 70%: P1–P7, P9–P13, P19, P20, and P29. All have at least 3 taps and all but P29 are less than four-beats long. However, some three-tap patterns have a low success rate (below 50%): P14 and P18 (both with 4 beats).

We could not identify similarities among the patterns that were difficult to reproduce. However, the number of taps and beats are the most obvious characteristics that can influence the ease of reproduction. In fact, we found a significant effect on success rate for taps ($F_{4,44} = 54.1, p < 0.0001$) and beats ($F_{4,44} = 85.5, p < 0.0001$), without significant interaction with FEEDBACK. Post-hoc t-tests support this hypothesis since, in most cases, the highest recognition rates were achieved for patterns with a small number of taps or beats (Figure 7).

Qualitative Results

Six participants out of 12 preferred the *Audio* feedback, 3 the *Visual* feedback, 2 the *AudioVisual* feedbacks and 1 no feedback. Moreover, 6 participants ranked *AudioVisual* second and 8 ranked the *None* condition last. Note that many participants pointed out that *AudioVisual* was confusing, providing too much information. They explained that in most cases, they chose one feedback (visual or auditive) and tried to ignore the other. Half of them preferred the *Audio* feedback because it was more related to rhythm than graphics.

We assessed the subjective difficulty of the task with the statement "I found it difficult to reproduce rhythmic patterns". Seven participants disagreed or strongly disagreed, 4 neither disagreed nor agreed, and only one agreed, but at the same time disagreeing for the *None* and *Visual* feedbacks.

Overall, both quantitative and qualitative results are encouraging and support our hypothesis that rhythmic patterns, as defined by our framework, is a viable input technique for interactive tasks. While quantitative results support the need to provide feedback while performing input, qualitative results inform on the type of appropriate feedback. Finally, an analysis of recognition errors gives insights on how to create a recognizer that would be more suitable for real applications.

A PATTERN CLASSIFIER

The goal of the structural recognizer in Experiment 1 was to assess how accurately participants could reproduce a stimulus pattern. This recognizer is deliberately strict, accounting only for variations in the overall tempo of the pattern, and

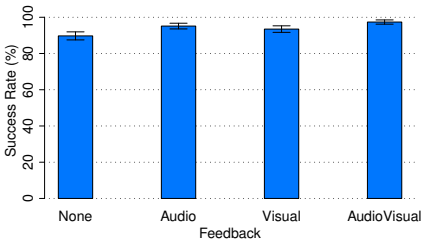


Figure 8. Revised success rate by FEEDBACK for the pattern classifier.

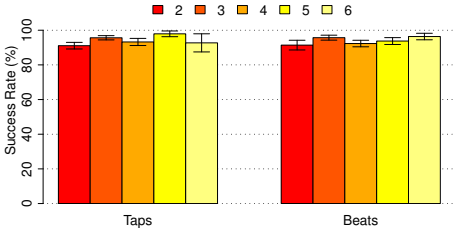


Figure 9. Revised success rate by number taps and length in beats for the pattern classifier.

it does not take advantage of the fact that the input patterns are assumed to be part of a known vocabulary. We designed a second recognizer for use in actual applications, that classifies an input pattern against a vocabulary.

In order to recognize a sequence of taps, this *pattern classifier* first counts the number of taps in the sequence and considers the subset of the vocabulary with that number of taps. Then, it calculates a score for each candidate pattern. First, it infers the duration of a beat by considering the duration of the sequence of taps and the number of taps of the candidate. Using this value, it scales the pattern to match the duration of the input sequence and sums the temporal differences of events onsets and durations. A duration of a quarter beat is used for impulses and releases between consecutive events (when lifting the finger from the device). Finally, the score is weighted by the ratio between the inferred beat duration and the 120 BPM reference (500ms).

This classifier is less strict than the structural recognizer because it will always match an input pattern to a pattern in the vocabulary if it is the only one with the same number of taps, unless a threshold is set on the lowest acceptable score. Moreover, normalization makes the recognizer match patterns that are homothetic of each other. This is the reason for weighing the score by the relative beat durations.

We tested this classifier with the data and vocabulary of Experiment 1. The overall success rate rose to 93.9%, more in line with the expectations of an applicative context. As with the previous recognizer, a one-way ANOVA for FEEDBACK reveals a significant effect on success rate ($F_{3,33} = 7.2, p = 0.0007$) (Figure 8).

Figure 9 shows that unlike the structural recognizer, success rate does not decrease with pattern "complexity": there is no significant effect of the number of taps or the length on

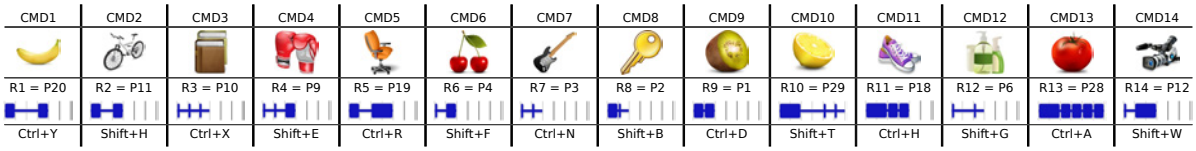


Figure 10. Commands used in Experiment 2. Pxx refers to the patterns of Experiment 1 (see Figure 4).

success rate[4]. Instead, we observe that success rates are affected by the similarity between patterns: a complex pattern can be recognized quite reliably provided that it is sufficiently different from other patterns with the same number of taps. For example, P30 is the only pattern made of 6 taps in our set, making recognition failure occur only when the subject tapped a wrong number of taps. By contrast, P17 seems to be more "complex" than the "simple" pattern P20 but the former has a 100% success rate and the latter 82%. In fact, the recognizer sometimes confuses P20 with P11. However, a post-hoc t-test with Holm correction reveals no significant difference between patterns for success rates.

In summary, we found that this classifier was well adapted to actual applications. In particular, a designer can create a vocabulary that minimizes the risk of patterns being confused.

EXPERIMENT 2: RHYTHMIC PATTERNS MEMORIZATION

In order to further validate Rhythmic Interaction, we conducted a second experiment to test whether patterns can be memorized and recalled in order to be used as an alternative to standard techniques for triggering commands. We compared rhythmic patterns with standard hotkeys in a "learn and recall" experiment similar to Appert and Zhai's comparison of gesture shortcuts with hotkeys [1], itself inspired by Grossman et al's study of hotkeys [14].

Variables

We compare two techniques for triggering commands (TECH factor): $Hotkey$ and $Rhythm$. A third condition, $Free$, lets participants choose the technique they prefer.

Each command C_i is a triplet associating an image I_i, used as a stimulus for this command, and two triggering techniques: a rhythmic pattern R_i and a hotkey K_i. The command set (CMD factor) has 14 commands: C_1, ..., C_{14}. We chose the images symbolizing the commands in a set of common objects and fruits (Figure 10).

For the rhythmic patterns, we selected 14 patterns of varying complexity from Experiment 1 and randomly assigned each pattern to a command. For the hotkeys, we created combinations of a modifier (`Shift` or `Ctrl`) and a letter. The letters were chosen so that they did not match the first letter of the name of the object representing the command, as in [1]. The goal is to avoid giving an unfair advantage to hotkeys, since there is no similar mnemonic association between rhythmic patterns and command names. Furthermore, the mapping between commands and hotkeys often varies by application and language. Figure 10 shows the resulting assignment.

[4]This could be due to the fact that in the vocabulary, there were few patterns with five taps or beats.

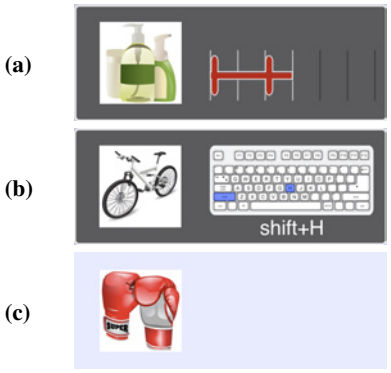


Figure 11. Stimulus in the learning phase for the $Rhythm$ (a) and $Hotkey$ (b) conditions, and in the testing phase for both conditions (c).

Task

The primary task of the experiment is to activate a command (C_i), presented by its stimulus image (I_i), with the triggering technique corresponding to the current TECH condition (R_i or K_i). The experiment has two phases: *learning* and *testing*.

During the learning phase, both the image I_i and the corresponding triggering technique (R_i or K_i) are shown to the participant. For rhythmic patterns, the static graphical representation is displayed next to the image (Figure 11a) and the audio stimulus is played twice. Hotkeys are presented with a short animation of the corresponding key-press sequence, also repeated twice, and text (Figure 11b).

In the testing phase, participants are presented with the image I_i only (Figure 11b). According to the current TECH condition, they must perform the corresponding hotkey K_i or rhythmic pattern R_i. If they forgot which trigger to perform, they are strongly encouraged to invoke a help screen by pressing the SPACE key. The task then switches to the learning mode, presenting the shortcut to perform as described above.

In both phases, the participant must perform the rhythmic pattern or the hotkey. For rhythmic patterns, we use the Audio-only feedback since Experiment 1 showed that it was effective and participants preferred it. Also, this avoids interference with the visual interface. For hotkeys, participants receive the usual kinesthetic feedback while pressing keys.

After entering each hotkey or pattern, the participants are asked to indicate which trigger they were trying to perform (Figure 12). Then, participants are notified of the correctness of their answer. If the answer is correct, they are given the result of the recognition. If not, the correct trigger is presented before moving to the next trial. The reason

Figure 12. Confirmation in the $Rhythm$ (a) and $Hotkey$ (b) conditions. Feedback for a wrong answer in the $Rhythm$ condition (c)).

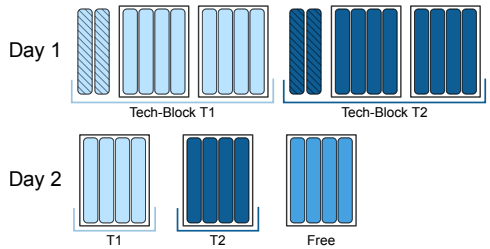


Figure 13. A sample session. Hatched sub-blocks are learning trials, boxed sub-blocks are sub-sessions.

for this procedure is that we are primarily interested in the memorization of the associations, not the participants' ability to perform the triggers. For rhythmic patterns, it also allows us to test the recognition rate of the classifier.

Apparatus & Participants

We used the same apparatus as in Experiment 1. We recruited 14 participants (5 female), aged between 22 and 33 (mean 26, median 26). Five of them had participated in Experiment 1.

Design & Procedure

The experiment is a within-subject design with technique (TECH) and command (CMD) as primary factors. The experiment is split into two sessions held on two consecutive days. The first day, all participants are presented with rhythmic patterns in a 5 minutes practice session based on Experiment 1. We use TECH as a blocking factor, counterbalanced across participants. The second day, a $Free$ block is added at the end of the testing phase. In this block, participants can choose to use $Rhythm$ or $Hotkey$ for each trial, but cannot get help.

Each TECH-block is divided into several sub-blocks of 15 trials: (i) 2 learning sub-blocks with 4 testing sub-blocks each on the first day; (ii) 4 testing sub-blocks on the second day. Thus, the testing phase of the experiment is split into SUBSESSIONS of 60 trials each: two on the first day to evaluate immediate memorization of triggering commands and one on the second day to test mid-term recall (Figure 13).

In order to simulate a more realistic setup, where some commands are more frequently used than others, we assign an apparition frequency to each of the 14 commands following a Zipf distribution [14, 1]. For the learning phase we use the frequencies (6, 6, 3, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1) and for the testing phase (12, 12, 6, 6, 4, 4, 3, 3, 2, 2, 2, 2, 1, 1). The 14 commands are combined with these frequencies using 7 different permutations, and each frequency assignment is

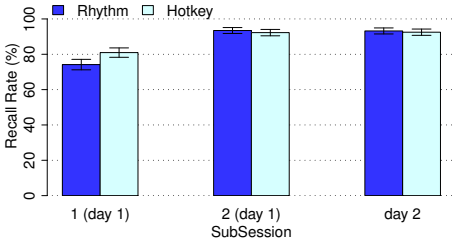


Figure 14. Recall rate for both techniques by sub-session.

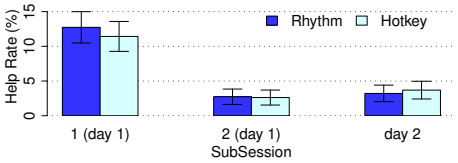


Figure 15. Help usage rate for both techniques for each sub-session.

counterbalanced across participants, resulting in the same number of trials for each command overall. The presentation of the trials is randomized across consecutive pairs of sub-blocks.

The experiment takes about one hour on day 1 and 30 minutes on day 2, after which participants are given a questionnaire to collect subjective observations and preferences.

Quantitative Results

Our main measures are (i) *recall rate*, the percentage of correct answers in the testing phase without help; and (ii) *help rate*, the percentage of trials where the participants used help in the testing phase. We analyze the results according to TECH and the three sub-sessions of the experiment by considering these measures in the model TECH $\times$ SUBSESSION $\times$ Rand(PARTICIPANT).

We find a significant effect of SUBSESSION on the recall rate ($F_{2,26} = 103$, $p < 0.0001$). A post-hoc t-test with Bonferroni correction shows that the recall rate is significantly lower only between the first sub-session and the two following ones (Figure 14). There is no significant effect of TECH on recall rate ($F_{1,13} = 0.61$, $p = 0.4474$), but the ANOVA reveals a significant interaction effect TECH $\times$ SUBSESSION ($F_{2,26} = 5.36$, $p = 0.0113$). Post-hoc t-tests with Bonferroni correction show a significant difference between $Rhythm$ and $Hotkey$ for the first sub-session (74% and 81% respectively). For the remaining sub-sessions, the results are extremely close between the two techniques with a recall rate of about 93% (Figure 14).

For the use of help, an ANOVA reveals a significant effect of SUBSESSION ($F_{2,26} = 17.3$, $p < 0.0001$), no effect of TECH ($F_{1,13} = 0.04$, $p = 0.8532$), and no TECH $\times$ SUBSESSION interaction effect ($F_{2,26} = 0.62$, $p = 0.545$). We find only one significant difference among sub-sessions: help was used more often in the first sub-session than in the two subsequent ones (see Figure 15).

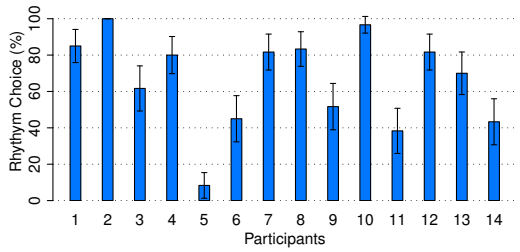


Figure 16. Percentage use of $Rhythm$ by participant ($Free$ condition).

Results for rhythmic patterns and hotkeys are quite similar, suggesting that rhythmic patterns can be memorized as successfully as hotkeys without mnemonics. This is a remarkable result considering how widespread hotkeys are.

Recall rates are consistent across commands. Considering only the $Rhythm$ condition, we build the model CMD × SUBSESSION × Rand(PARTICIPANT) for recall rate and help and see a significant effect of CMD on recall rate ($F_{13,169} = 1.17$, $p = 0.025$). A post-hoc t-test with Holm corrections shows significant differences only between R_3 and R_{13} (recall rate about 97%) and R_{10}, R_{11} and R_{14} (∼80%).

To test our classifier, we compare the pattern recognized by the classifier with the answer selected by the participant using the model TECH × SUBSESSION × Rand(PARTICIPANT). We find a significant effect of TECH ($F_{1,13} = 5.34$, $p = 0.038$), with $Rhythm$ having a significant lower success rate than $Hotkey$: 85.2% vs. 91.8%. The success rate for $Hotkey$ is surprisingly low, as we expect few if any errors when entering hotkeys. This may be due to participants changing their mind as to which was the right hotkey when they see the answer sheet. For $Rhythm$, the rate is also lower than expected, but the same phenomenon may have occurred. Indeed, the success rate of $Rhythm$ relatively to $Hotkey$ is 92.8%, close to the rate obtained on the data for Experiment 1 (94%).

Qualitative Results

Figure 16 shows the percentage of trials where participants used rhythmic patterns in the $Free$ condition, on the second day of the experiment. Ten participants (out of 14) used rhythmic patterns more often than hotkeys. Seven participants used rhythmic patterns more than 80% of the time, while only one participant used rhythmic patterns less than 20% of the time.

The answers to the questionnaire were generally positive, confirming the previous results. Out of the 14 participants, 9 preferred using the rhythmic patterns, 3 the hotkeys, 2 had no preference. Those who preferred using rhythmic patterns did so mostly because of the "fun factor" of tapping rhythms, but also because it could be performed "in place" on the trackpad, even for a novice user, without having to visually search the keys on the keyboard. On the other hand, several participants noticed that hotkeys are faster to perform and preferred to use hotkeys when the corresponding pattern is too long.

Regarding memorization, some participants reported using mnemonics related to the rhythm itself in order to help memorization. For instance, a subject linked the "boxing gloves" command and the corresponding pattern P9 (Figure 4) to a "pif paf boom" onomatopoeia that, for him, echoed the "short short medium" structure of the pattern. Another participant also reported linking the pattern structure with the pronunciation of the object's name, e.g., "toma-to-to-to-to" for command 13 and pattern P28. Subjects also used the graphical representation of patterns to memorize them, which supports our design for this representation. For example, one participant stated that "the rhythmic pattern's visual representation for the cherry looks like a cherry".

These comments suggest that users elaborate efficient strategies for the memorization of rhythmic patterns, based on the rhythm itself or its visualization. Since commands were assigned to rhythmic patterns randomly, we did not expect such associations, but this finding opens the way to studying ways to reinforce these associations. This is commonly done for gestures, e.g., a question mark for help, and hotkeys, e.g. Ctrl-*S* for *S*ave. In particular, various strategies could be explored to create visual "cheatsheets" for rhythmic patterns or display them next to menu commands, like hotkeys.

In addition, the complexity of performing rhythmic patterns can be turned into an advantage for memorization. Since deeper and greater numbers of levels of encoding and processing help memory [8], combining motor and auditive perception of rhythmic patterns may help users memorize, i.e., encode, their associations with commands.

SUMMARY AND PERSPECTIVES

In this paper we studied the use of rhythmic patterns in HCI. We explored Rhythmic Interaction as an opportunity to generalize the primitive use of rhythm in existing techniques, e.g., long click and double click, as well as to promote a new input modality. Since Rhythmic Interaction relies on the time dimension instead of the spatial and visual dimensions used by most input methods, it is well suited when space is limited or when visual attention is not available.

We presented a grammar for creating rhythmic patterns as well as two recognizers that do not require training. A first experiment evaluated the ability of casual users to reproduce rhythmic patterns very precisely with different feedback conditions. We found that some complex patterns can be difficult to reproduce in such a precise way, but that audio and/or visual feedback improve accuracy. After analyzing recognition errors, we designed a different recognizer that reached 94% recognition rate for the 30-pattern vocabulary of Experiment 1. We ran a second experiment to investigate the memorization of associations between rhythmic patterns and commands, i.e., *rhythmic shortcuts*. The results suggest that rhythmic patterns are recalled as efficiently as traditional hotkeys and that users create effective mnemonic strategies to associate rhythms with commands.

This work demonstrates the potential of rhythmic patterns as an input method, and contributes a 14-pattern vocabulary that has proven usable by novice users. Beyond triggering commands and switching modes in standard desktop environments, rhythmic patterns could be used in many contexts: eye-free control of a mobile device, such as a cellular phone

or a mobile player; remote control of interactive environments such as wall-size displays by tapping on wearable sensors without the need for visual attention; selection of an object on a tabletop when it is not easily reachable, etc.

Our future work will address issues such as the segmentation of patterns, the scalability of the vocabularies and the speed of execution, which are important for the design of Rhythmic Interactions. Another area for future work is the use of multiple fingers or both hands to tap patterns and to combine rhythmic interaction with other interaction techniques. More complex actions than tapping should also be explored to enter rhythmic structures, such as performing sequences of gestures or keyboard taps, as well as the use of the temporal dimension to convey additional information. Furthermore, rhythmic *output*, such as vibration patterns on mobile devices, seems worth studying since perception and performance of rhythmic patterns are tightly linked. Finally, the power of rhythmic interaction could be expanded by exploiting syntactic features used in music such as performing sequential or parallel combinations of patterns.

ACKNOWLEDGEMENTS

We thank Caroline Appert for feedback on earlier versions of this paper, our participants and the anonymous reviewers.

REFERENCES

1. Appert, C., and Zhai, S. Using strokes as command shortcuts: cognitive benefits and toolkit support. In *Proc. CHI '09*, ACM (2009), 2289–2298.
2. Arom, S., Thom, M., Tuckett, B., and Boyd, R. *African polyphony and polyrhythm: musical structure and methodology*. Cambridge university press, 1991.
3. Baudel, T., and Beaudouin-Lafon, M. Charade: Remote control of objects using free-hand gestures. *Comm. ACM 36* (1993), 28–35.
4. Blair, C. R. On computer transcription of manual morse. *JACM 6*, 3 (1959), 429–442.
5. Brochard, R., Touzalin, P., Despres, O., and Dufour, A. Evidence of beat perception via purely tactile stimulation. *Brain Res. 1223* (2008), 59 – 64.
6. Chen, S.-C., Chien, C.-Y., Chang, W.-M., and Lin, S.-W. A new assistive communication system for the serious disabled. In *Proc. iCREATe '08*, START Centre (2008), 59–64.
7. Clarke, E. Rhythm and timing in music. *The Psychology of Music 2* (1999), 473–500.
8. Craik, F. I., and Lockhart, R. S. Levels of processing: A framework for memory research. *J. Verbal Learning and Verbal Behavior 11*, 6 (1972), 671–684.
9. Crossan, A., and Murray-Smith, R. Rhythmic interaction for song filtering on a mobile device. In *Proc. HAID '06*, Springer (2006), 45–55.
10. Faure, G., Chapuis, O., and Roussel, N. Power tools for copying and moving: useful stuff for your desktop. In *Proc. CHI '09*, ACM (2009), 1675–1678.
11. Fekete, J.-D., Elmqvist, N., and Guiard, Y. Motion-pointing: target selection using elliptical motions. In *Proc. CHI '09*, ACM (2009), 289–298.
12. Fraisse, P. Rhythm and tempo. In *The Psychology of Music*, Academic Press (1982), 149–180.
13. Glass, L. Synchronization and rhythmic processes in physiology. *Nature 410*, 6825 (2001), 277–284.
14. Grossman, T., Dragicevic, P., and Balakrishnan, R. Strategies for accelerating on-line learning of hotkeys. In *Proc. CHI '07*, ACM (2007), 1591–1600.
15. Hinckley, K., Baudisch, P., Ramos, G., and Guimbretiere, F. Design and analysis of delimiters for selection-action pen gesture phrases in scriboli. In *Proc. CHI '05*, ACM (2005), 451–460.
16. Kurtenbach, G., and Buxton, W. User learning and performance with marking menus. In *Proc. CHI '94*, ACM (1994), 258–264.
17. Lantz, V., and Murray-Smith, R. Rhythmic interaction with a mobile device. In *Proc. NordiCHI '04*, ACM (2004), 97–100.
18. Large, E. W. Periodicity, pattern formation, and metric structure. *J. New Music Research 30*, 2 (2001), 173 – 185.
19. MacDougall, H. G., and Moore, S. T. Marching to the beat of the same drummer: the spontaneous tempo of human locomotion. *J. Applied Physiology 99*, 3 (2005), 1164–1173.
20. Malacria, S., Lecolinet, E., and Guiard, Y. Clutch-free panning and integrated pan-zoom control on touch-sensitive surfaces: the cyclostar approach. In *Proc. CHI '10*, ACM (2010), 2615–2624.
21. Manuel, R. *Histoire de la Musique*. Encyclopédie de la Pléiade, Gallimard, Paris, France, 1960.
22. Maury, S., Athénes, S., and Chatty, S. Rhythmic menus: toward interaction based on rhythm. In *Proc. CHI '99 EA*, ACM (1999), 254–255.
23. Moelants, D. Preferred tempo reconsidered. In *Proc. ICMPC '02*, AMPS (2002), 580–583.
24. Petitto, L. A., Holowka, S., Sergio, L. E., Levy, B., and Ostry, D. J. Baby hands that move to the rhythm of language: hearing babies acquiring sign languages babble silently on the hands. *Cognition 93*, 1 (2004), 43 – 73.
25. Potter, D. D., Fenwick, M., Abecasis, D., and Brochard, R. Perceiving rhythm where none exists: Event-related potential (erp) correlates of subjective accenting. *Cortex 45*, 1 (2009), 103–109.
26. Rammsayer, T., and Lima, S. Duration discrimination of filled and empty auditory intervals: Cognitive and perceptual factors. *Perception and Psychophysics 50* (1991), 565–574.
27. Repp, B. H. Rate limits in sensorimotor synchronization. *Advances in Cognitive Psychology 2*, 2 (2006), 163–181.
28. Robinson, S., Rajput, N., Jones, M., Jain, A., Sahay, S., and Nanavati, A. Tapback: towards richer mobile interfaces in impoverished contexts. In *Proc. CHI '11*, ACM (2011), 2733–2736.
29. Sacks, O. *Musicophilia: Tales of Music and the Brain*, 2 ed., vol. 1. Vintage Books, New York, USA, 2008.
30. Scott, J., Dearman, D., Yatani, K., and Truong, K. N. Sensing foot gestures from the pocket. In *Proc. UIST '10*, ACM (2010), 199–208.
31. Szentgyorgyi, C., and Lank, E. Five-key text input using rhythmic mappings. In *Proc. ICMI '07*, ACM (2007), 118–121.
32. Westeyn, T., and Starner, T. Recognizing song-based blink patterns: Applications for restricted and universal access. In *Proc. FGR '04*, IEEE (2004), 717–722.
33. Wobbrock, J. O. Tapsongs: tapping rhythm-based passwords on a single binary sensor. In *Proc. UIST '09*, ACM (2009), 93–96.

A Body-centric Design Space for Multi-surface Interaction

Julie Wagner[1,2,3] **Mathieu Nancel**[2,1] **Sean Gustafson**[4] **Stéphane Huot**[2,1] **Wendy E. Mackay**[1,2]
wagner@telecom-paristech.fr {nancel, huot, mackay}@lri.fr sean.gustafson@hpi.uni-potsdam.de
[1]Inria – [2]Univ. Paris-Sud & CNRS (LRI) F-91405 Orsay, France
[3]Télécom ParisTech Paris, France
[4]Hasso Plattner Institute Potsdam, Germany

ABSTRACT
We introduce *BodyScape*, a body-centric design space that allows us to describe, classify and systematically compare multi-surface interaction techniques, both individually and in combination. BodyScape reflects the relationship between users and their environment, specifically how different body parts enhance or restrict movement within particular interaction techniques and can be used to analyze existing techniques or suggest new ones. We illustrate the use of BodyScape by comparing two free-hand techniques, *on-body touch* and *mid-air pointing*, first separately, then combined. We found that touching the torso is faster than touching the lower legs, since it affects the user's balance; and touching targets on the dominant arm is slower than targets on the torso because the user must compensate for the applied force.

Author Keywords
Multi-surface interaction, Body-centric design space

ACM Classification Keywords
H.5.2. Information Interfaces and Presentation: User Interfaces.: Theory and methods

INTRODUCTION

Multi-surface environments encourage users to interact while standing or walking, using their hands to manipulate objects on multiple displays. Klemmer et al. [19] argue that using the body enhances both learning and reasoning and this interaction paradigm has proven effective for gaming [32], in immersive environments [26], when controlling multimedia dance performances [21] and even for skilled, hands-free tasks such as surgery [35].

Smartphones and devices such as Nintendo's Wii permit such interaction via a hand-held device, allowing sophisticated control. However, holding a device is tiring [27] and limits the range of gestures for communicating with co-located users, with a corresponding negative impact on thought, understanding, and creativity [13]. Krueger's VIDEOPLACE [20] pioneered a new form of whole-body interaction in which users stand or walk while pointing to a wall-sized display. Today, off-the-shelf devices like Sony's Eyetoy and Microsoft's Kinect let users interact by pointing or moving their bodies, although most interaction involves basic pointing or drawing. Most research in complex device-free interaction focuses on hand gestures, e.g. Charade's [2] vocabulary of hand-shapes that distinguish between "natural" and explicitly learned hand positions, or touching the fore-arm, e.g. Skinput's [16] use of bio-acoustic signals or PUB's [23] ultrasonic signals.

However, the human body offers a number of potential targets that vary in size, access, physical comfort, and social acceptance. We are interested in exploring these targets to create more sophisticated body-centric techniques, sometimes in conjunction with hand-held devices, to interact with complex data in multi-surface environments. Advances in sensor and actuator technologies have produced a combinatorial explosion of options, yet, with few exceptions [31, 27], we lack clear guidelines on how to combine them in a coherent, powerful way. We argue that taking a *body-centric* approach, with a focus on the sensory and motor capabilities of human beings, will help restrict the range of possibilities in a form manageable for an interaction designer.

This paper introduces *BodyScape*, a design space that classifies body-centric interaction techniques with respect to multiple surfaces according to input and output location relative to the user. We describe an experiment that illustrates how to use the design space to investigate atomic and compound body-centric interaction techniques, in this case, compound mid-air interaction techniques that involve pointing on large displays to designate the focus or target(s) of a command. Combining on-body touch with the non-dominant hand and mid-air pointing with the dominant hand is appealing for interacting with large displays: both inputs are always available without requiring hand-held devices. However, combining them into a single, compound action may result in unwanted interaction effects. We report the results of our experiment and conclude with a set of design guidelines for placing targets on the human body depending on simultaneous body movements.

BODYSCAPE DESIGN SPACE & RELATED WORK

Multi-surface environments require users to be "physically" engaged in the interaction and afford physical actions like pointing to a distant object with the hand or walking towards a large display to see more details [3]. The body-centric paradigm is well-adapted to device- or eyes-free interaction techniques because they account for the role of the body in the interactive environment. However, few studies and designs take this approach, and most of those focus on large displays [22, 31, 27].

J. Wagner, M. Nancel, S. Gustafson, S. Huot, W. E. Mackay.
A Body-centric Design Space for Multi-surface Interaction.
In CHI'13: Proceedings of the 31st International Conference on Human Factors in Computing Systems, ACM, April 2013.
Authors Version

Today's off-the-shelf technology can track both the human body and its environment [17]. Recent research prototypes also permit direct interaction on the user's body [16, 23] or clothes [18]. These technologies and interaction techniques suggest new types of body-centric interaction, but it remains unclear how they combine with well-studied, established techniques, such as free-hand mid-air pointing, particularly from the user's perspective.

Although the literature includes a number of isolated point designs, we lack a higher-level framework that characterizes how users coordinate their movements with, around and among multiple devices in a multi-surface environment. Previous models, such as the *user action notation* [11], separate interaction into asynchronous tasks and analyze the individual steps according to the user's action, interface feedback, and interface internal state. Loke et al. investigated users' input movements when playing an Eyetoy game [24] and analyzed their observations using four existing frameworks. These models do not, however, account for the body's involvement, including potential interaction effects between two concurrent input movements. Our goal is to define a more general approach to body-centric interaction and we propose a design space that: (i) assesses the adequacy of specific techniques to an environment or use context; and (ii) informs the design of novel body-centric interaction techniques.

We are aware of only two design spaces that explicitly account for the body during interaction. One focuses on the interaction space of mobile devices [7] and the other offers a task-oriented analysis of mixed-reality systems [28]. Both consider *proximity* to the user's body but neither fully captures the distributed nature of multi-surface environments. We are most influenced by Shoemaker et al.'s [31] pioneering work, which introduced high-level design principles and guidelines for body-centric interaction on large displays.

BodyScape

BodyScape builds upon Card et al.'s morphological analysis [5], focusing on (i) the relationships between the user's body and the interactive environment; (ii) the involvement of the user's body during the interaction, i.e. which body parts are involved or affected while performing an interaction technique; and (iii) the combination of "atomic" interaction techniques in order to manage the complexity of multi-surface environments. These in turn were inspired by early research on how people adjust their bodies during coordinated movements, based on constraints in the physical environment or the body's own kinematic structure [25]. They help identify appropriate or adverse techniques for a given task, as well as the impact they may have on user experience and performance, e.g., body movement conflicts or restrictions.

Relationships Between the Body and the Environment

Multi-surface environments distribute user input and system visual output[1] on multiple devices (screens, tactile surfaces, handheld devices, tracking systems, on-body sensors, etc.). The relative location and body positions of the user thus play a central role in the interactions she can perform. For example, touching a tactile surface while looking at a screen on your back is obviously awkward. This physical separation defines the two first dimensions of *BodyScape*: User *Input* (**D1**) and System *Visual Output* (**D2**). Using a body-centric perspective similar to [7, 28], we identify two possible cases for input and output: *Relative to the body* and *Fixed in the world*. Such body-environment relationships have been considered in Augmented Reality systems [12], but never applied to a body-centric description of interaction techniques.

D1*: Input* – User input may be *relative to the body* and thus performed at any location in the environment, e.g. on-body touch, or *fixed in the world*, which restricts the location and body position of the user, e.g. standing next to an interactive table. Different technologies offer users greater or lesser freedom of movement. Some interaction techniques require no devices, such as direct interaction with the user's body [23] or clothes [18]. Others require a hand-held device, constraining interaction, but not the orientation of the body. Others further restrict movement, such as mid-air pointing at a wall display, in which the user holds the arm, which is tracked in 3d, in a fixed position relative to an external target.

D2*: Visual Output* – Multi-surface environments are inevitably affected by the separation of visual output over several devices [33, 34]. Users must adjust their gaze, switching their attention to output devices that are relevant to the current task by turning the head and – if that is not sufficient – turning the torso, the entire body, or even walking. Visual output *relative to the body* is independent of the user's location in the environment, e.g., the screen of a hand-held device. It does not constrain the user's location or body position, except if a limb must hold the device. Conversely, visual output *fixed in the world* requires users to orient the head towards the target's physical location, e.g., where it is projected on a wall. Users' location and body positions are constrained such that they can see the visual output effectively.

The *BodyScape* design space differentiates between *Touch*-based and *Mid-air* user input, since these can affect performance and restrict the position of the body. Body movements and their coordination depends upon the physical connection with the environment [10]. For example, Nancel et al. [27] showed that touch-based pan-and-zoom techniques are faster on large displays than mid-air gestures because tactile feedback helps guide input movements. Multi-surface environments may add additional constraints, such forcing users to walk to an interactive tabletop in order to touch it.

Body Restriction in the Environment – We define this as a qualitative measure of how a given interaction technique constrains the user's body position, as determined by a combination of the *Input* and *Visual Output* dimensions above, from free to restricted (horizontal axis in Fig. 1). The *Input* dimension clearly restricts body movement more than *Visual Output*, and *Touch* is more restrictive than *Mid-air* gestures, when the input is fixed in the world. For example, one can watch a fixed display from a distance, at different angles, whereas touch input devices require physical proximity.

[1] We do not consider auditory feedback since sound perception does not depend upon body position, except in environments featuring finely tuned spatial audio.

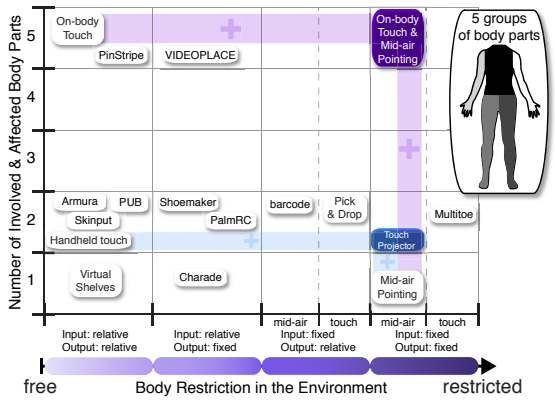


Figure 1. Atomic body-centric interaction techniques in BodyScape, according to the level of *Body Restriction in the Environment* and number of *Involved and Affected* limbs. Compound techniques (colored background) are linked to their component atomic techniques.

Together, *Input* and *Visual Output* dictate the body's remaining degrees of freedom (translation and rotation) available for other potential interactions or body movements. Note that *Body Restriction* is not necessarily negative. For example, assigning each user their personal display area in a collaborative multi-surface environment restricts their movement, but can prevent common problems that arise with interactive tables [30] such as visual occlusions, collisions, conflicts and privacy concerns. Figure 1 shows various atomic interaction techniques in terms of their level of body restriction and the total number of involved and affected body parts, and shows how combining them into a compound technique further restricts body movement.

***D3**: Body Involvement* – *BodyScape* offers a finer grained assessment of body restriction by considering which parts of the user's body are *involved* in an interaction technique. Every interaction technique involves the body with varying degrees of freedom, from simple thumb gestures on a hand-held device [27], to whole-body movements [21]. We define a group of limbs involved in a technique as the *involved body parts*. For example, most mid-air pointing techniques involve the dominant arm, which includes the fingers and hand, the wrist, the forearm, the upper arm and the shoulder.

A technique may *involve* a group of limbs and also *affect* other limbs. For example, on-body touch interaction involves one hand and the attached arm, and the limb touched by the hand is the *affected body part*. This implies further restrictions on the body, since affected body parts are unlikely to be involved in the interaction and vice versa, especially when interaction techniques are combined. We define five groups of *involved body parts*: the *dominant arm*, the *non-dominant arm*, the *dominant leg*, the *non-dominant leg* and the *torso*.

We omit the head when considering *involved* and *affected* body parts, since the location of the visual output is the primary constraint. Although head orientation has been used to improve access to data on large displays [8], this is only a "passive" approach in which the system adapts itself to the user's head orientation.

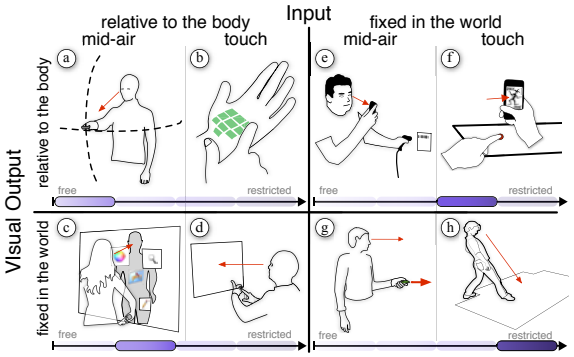


Figure 2. BodyScape presents a taxonomy of atomic body-centric interaction techniques, organized according to *Input* and *Visual Output*. a) Virtual Shelves [22]; b) Skinput [16]; c) Body-centric interaction techniques for large displays [31]; d) PalmRC [9]; e) Scanning objects with feedback on a device; f) Pick and Drop [29]; g) Mid-air pointing [27]; and h) Multitoe [1].

Classification of Body-centric Interaction Techniques

Figure 2 lays out atomic body-centric interaction techniques from the literature along the *Input* and *Visual Output* dimensions, illustrating their impact on body restrictions in the environment. Each technique involves performing an elementary action, e.g. moving a cursor or selecting a target.

Relative Input / Relative Output – The least restrictive combination lets users move freely in the environment as they interact and obtain visual feedback. VirtualShelf [22] is a mid-air example in which users use the dominant arm to orient a mobile phone within a spherical area in front of them to enable shortcuts (Fig.2a). *Armura* [15] extends this approach with wearable hardware that detects mid-air gestures from both arms and projects visual feedback onto the user's body. Skinput [16] (Fig. 2b) is a touch example that accepts touch input on the user's forearm and provides body-relative visual output from a projector mounted on the shoulder. The dominant arm is involved and the non-dominant arm is affected by the pointing.

Relative Input / Fixed Output – A more restrictive combination constrains the user's orientation and, if the distance to the display matters, the user's location. Shoemaker's [31] mid-air technique involves pointing to a body part and pressing a button on a hand-held device to select a command. Visual output consists of the user's shadow projected on the wall with the available commands associated with body locations. Only the pointing arm is involved and users must remain oriented towards the screen (Fig. 2c). PalmRC [9] (Fig. 2d) allows free-hand touch operations on a TV set. Users press imaginary buttons on their palm [14] and see visual feedback on the fixed TV screen. One arm is involved in the interaction; the other is affected.

Fixed Input / Relative Output – The next most restrictive approach requires users to stand within a defined perimeter, limiting movement. Here, touch is more constrained than mid-air gestures: standing within range of a Kinect device is less restrictive than having to stand at the edge of an interactive table. A simple mid-air example involves a user

who scans a barcode while watching feedback on a separate mobile device (Fig. 2e). Pick and Drop [29] uses touch to transfer an object from a fixed surface to a mobile device (Fig. 2f). Both examples involve the dominant arm and affect the non-dominant arm, which carries the handheld device.

Fixed Input / Fixed Output – The most restrictive combination constrains both the user's location and visual attention. A common mid-air technique uses the metaphor of a laser pointer to point to items on a wall-sized display. Although the interaction is performed at a distance, the user must stand at a specified location in order to accurately point at a target on the wall, making it "fixed-in-the-world" (Fig. 2g). Conventional touch interaction on a tabletop or a large display is highly restrictive, requiring the user to stand in a fixed location with respect to the surface. Multitoe [1] is even more constrained, since both touch input and visual output appear on the floor, next to the feet (Fig. 2h).

Body Involvement – Figure 1 shows that most body-centric techniques only involve and affect one or two groups of body parts, usually the arms. We know of only a few "whole-body" techniques that involve or affect the entire body: VIDEOPLACE [20] and its successors for games and entertainment and PinStripe [18], which enables gestures on the users' clothing.

Compound Techniques in Multi-surface Environments

Complex tasks in multi-surface environments combine several interaction techniques: (i) in *series*, e.g., selecting an object on one touch surface and then another; or (ii) in *parallel*, e.g., simultaneously touching one object on a fixed surface and another on a handheld device.

Serial Combination – a temporal sequence of interaction techniques. The combined techniques can be interdependent (sharing the same object, or the output of one as the input of the other), but the first action should end before the second starts. For example, the user can select an object on a tactile surface (touch and release) and then apply a function to this object with a menu on a mobile device. Serial compound techniques do not increase the restrictions imposed by each atomic technique in the sequence, nor the involved or affected body parts. However, one must still design serial combinations to avoid awkward movements, such as having to constantly walk back and forth, move a device from one hand to another or repeatedly switch attention between fixed and relative displays.

Parallel Combination – performing two techniques at the same time. The techniques may be independent or dependent. For example, the user might touch two tactile surfaces simultaneously in order to transfer an object from one to the other [36]. Unlike serial combinations, these compound techniques may significantly restrict the body's movement and raise conflicts between involved and affected body parts.

The constraint on body movement is determined by the more restrictive of the combined techniques. Thus, combining a "fixed-in-the-world" with a "relative-to-the-body" technique will be as restrictive as "fixed-in-the-world". Touchprojector [4] illustrates this well (see Fig. 1). The user uses one device as a lens to select objects on a distant display, orienting it towards the target (mid-air fixed input and fixed output) while simultaneously touching the device's tactile screen to select the target (touch relative input + relative output). Touchprojector is thus considered a "touch fixed input and fixed output" technique in *BodyScape*. The advantage of minimizing body restrictions with relative-to-the-body technique is overridden by requiring a fixed input. Even so, Touchprojector offers other advantages, since users can interact directly with a remote display without having to move to the display or use another interaction device.

BODYSCAPE EXPERIMENT: COMBINING ON-BODY TOUCH AND MID-AIR POINTING

Our work with users in complex multi-surface environments highlighted the need for interaction techniques that go beyond simple pointing and navigation [3]. Users need to combine techniques as they interact with complex data spread across multiple surfaces. The *BodyScape* design space suggests a number of possibilities for both atomic and compound interaction techniques that we can now compare and contrast.

This section illustrates how we can use the *BodyScape* design space to look systematically at different types of body-centric interaction techniques, both in their atomic form and when combined into compound interaction techniques. We chose two techniques, illustrated in Figure 2d, ON-BODY TOUCH input, and 2g, MID-AIR POINTING input, both with visual output on a wall display, which is where our users typically need to interact with their data. Although the latter has been well-studied in the literature [27], we know little of the performance and acceptability trade-offs involved in touching one's own body to control a multi-surface environment. Because it is indirect, we are particularly interested in on-body touch for secondary tasks such as confirming a selection, triggering an action on a specified object, or changing the scope or interpretation of a gesture. We are also interested in how they compare with each other, since MID-AIR POINTING restricts movement more than ON-BODY TOUCH (Fig. 2g vs. 2d), while ON-BODY TOUCH affects more body parts than MID-AIR POINTING (Fig. 1).

Finally, we want to create *compound* interaction techniques, so as to increase the size of the command vocabulary and offer users more nuanced control. However, because this involves coordinating two controlled movements, we need to understand any potential interaction effects. The following experiment investigates the two atomic techniques above, which also act as baselines for comparison with a compound technique that combines them. The two research questions we are addressing are thus:

1. *Which on-body targets are most efficient and acceptable?* Users can take advantage of proprioception when touching their own bodies, which enables eyes-free interaction and suggests higher performance. However, body targets differ both in the level of motor control required to reach them, e.g., touching a foot requires more balance than touching a shoulder, and in their social acceptability, e.g., touching below the waist [18].

2. *What performance trade-offs obtain with compound body-centric interaction techniques?* Users must position themselves relative to a target displayed on the wall and stabilize the body to point effectively. Simultaneously selecting on-body targets that force shifts in balance or awkward movements may degrade pointing performance. In addition, smaller targets will decrease pointing performance, but may also decrease ON-BODY TOUCH performance.

Method

Participants

We recruited sixteen unpaid right-handed volunteers (13 men, average age 28); five had previous experience using a wall-sized display. All had good to excellent balance (median 4 on a 5-high Likert scale) and practiced at least one activity that requires balance and body control. All wore comfortable, non-restrictive clothing.

Apparatus

Participants stood in front of a wall-sized display consisting of 32 high-resolution 30" LCD displays laid out in an 8×4 matrix (5.5m ×1.8m) with a total of 20480×6400 pixels (100.63 ppi). Participants wore passive infra-red reflective markers that were tracked in three dimensions by ten VICON cameras with sub-millimeter accuracy at a rate of up to 200 Hz. Markers were mounted on a wireless mouse held in the user's dominant hand to track pointing at a target on the wall, on the index finger of the non-dominant hand to track on-body touches, and on protective sports gear – belt, forearms, shoulders and legs – to track on-body targets. The latter were adjustable to fit over the participants' clothing. VICON data was filtered through the 1Euro filter [6].

Based on pilot studies, we defined 18 body target locations distributed across the body (Fig. 3), ranging in size from $9cm$ on the forearm to $16cm$ on the lower limbs, depending upon location and density of nearby targets, grouped as follows:

Dominant Arm: 4 targets on dominant arm (D ARM = upper arm, elbow, forearm, wrist)
Dominant Upper Body: 4 targets on dominant side of upper body (D UPPER = thigh, hip, torso, shoulder)
Non-dominant Upper Body: 4 targets on non-dominant side of upper body (ND UPPER = thigh, hip, torso, shoulder)
Dominant Lower Leg: 3 targets on dominant side of lower leg (D LOWER = knee, tibia, foot)
Non-dominant Lower Leg: 3 targets on non-dominant side of lower leg (ND LOWER = knee, tibia, foot)

In ON-BODY TOUCH conditions, participants wore an IR tracked glove on the non-dominant hand with a pressure sensor in the index finger. The system made an orthogonal projection from the index finger to the touched limb segment using a skeleton-based model to calculate the closest body target.

Wall pointing tasks varied in difficulty from *easy* (diameter of the circular target was $1200px$ or $30cm$) to *medium* ($850px$ or $21.25cm$) to *hard* ($500px$ or $12.5cm$). Wall targets were randomly placed $4700px$ ($117.5cm$) from the starting target.

Data Collection

We collected timing and error data for each trial, as follows:

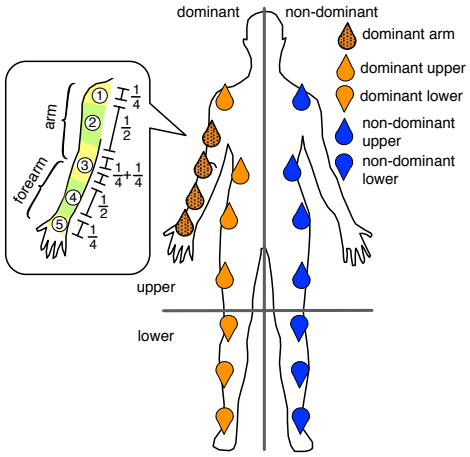


Figure 3. 18 body targets are grouped into five categories.

TRIAL TIME: from trial start to completion.
POINTING REACTION TIME: from appearance of on-screen target to cursor displacement of more than $1000px$.
POINTING MOVEMENT TIME: from initial cursor movement to entry into goal target.
CURSOR READJUSTMENT TIME: from leaving goal target to relocating cursor onto goal target.
BODY REACTION TIME: from appearance of trial stimulus to leaving starting position.
BODY POINTING TIME: from leaving start position to touching on-body target.
BODY ERRORS: number of incorrect touches detected on body target[2]; includes list of incorrect targets per error.

We debriefed participants at the end of the experiment and asked them to rank on a Likert scale: (i) perceived comfort of each body target according to each MID-AIR POINTING condition ('1=very uncomfortable' to '5=very comfortable'); and (ii) social acceptability of each on-body target:"Would you agree to touch this body target in a work environment with colleagues in the same room?" ('1=never' to '5=certainly').

Procedure

Each session lasted about 60 minutes, starting with a training session, followed by blocks of trials of the following conditions, counter-balanced across subjects using a Latin square.

BODY ONLY: Non-dominant hand touches one of 18 on-body targets (atomic technique − 18×5 replications = 90 trials)
POINTING ONLY: Dominant hand points to one of three target sizes (atomic technique − 3×5 replications = 15 trials)
POINTING+BODY: Combines touching an on-body target with selecting a wall target (compound technique − (18×3)×5 replications = 270 trials)

Participants were thus exposed to 75 unique conditions, each replicated five times, for a total of 375 trials. BODY ONLY and POINTING+BODY trials were organized into blocks of six, with

[2]Includes both system detection and user errors.

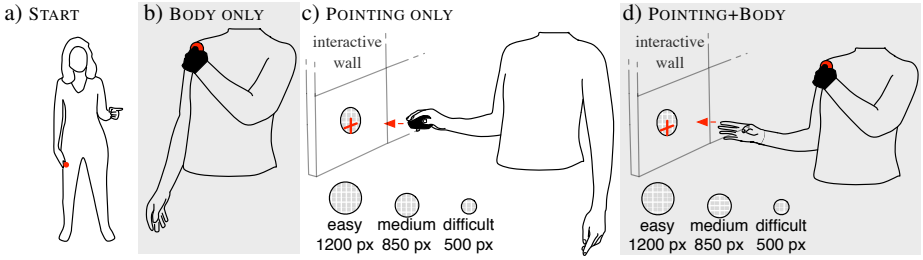


Figure 4. a) Starting position b) BODY ONLY **c)** POINTING ONLY **d)** POINTING+BODY

Starting position: non-dominant hand at the hip and/or dominant hand points to a starting target on the wall display. BODY ONLY and POINTING ONLY are atomic conditions; POINTING+BODY is compound: a body touch triggers the selected wall target.

the location of body targets randomized and no two successive trials involved the same body target group. POINTING ONLY trials were organized into blocks of five and all wall pointing trials were counterbalanced across difficulty. The two atomic interaction techniques, BODY ONLY and POINTING ONLY serve as baseline comparisons for performance with the compound interaction technique, POINTING+BODY.

TASK: Participants were asked to perform trials as quickly and accurately as possible. They were asked to point and select on-body targets using their non-dominant hand's index finger in the BODY ONLY condition, and to point and select wall-targets using a mouse device held in the dominant hand in the POINTING ONLY condition. The compound POINTING+BODY condition asked users to point to the wall-target and keep the cursor inside before selecting an on-body target.

BODY ONLY (Fig. 4b): The starting position involves standing comfortably facing the wall display, with the non-dominant hand at the thigh (Fig. 4a). The trial begins when an image of a body silhouette appears on the wall, with a red circle indicating the location of the on-body target to acquire. The participant touches that target with the index finger of the non-dominant hand as quickly and accurately as possible. Participants were asked to avoid crouching or bending their bodies, which forced them to lift their legs to reach lower-leg targets. The trial ends only when the participant selects the correct target; all intermediate incorrect selections are logged.

Figure 5 shows how different body parts interact for different on-body targets. The non-dominant arm is always involved, since it is responsible for pointing at the target. However, some on-body targets also affect other body parts, which may have adverse effects, such as shifting one's balance to touch the foot (Fig. 5c).

POINTING ONLY (Fig. 4c): The starting position involves standing comfortably facing the wall display and using the dominant hand to locate a cursor within a circular target displayed in the center of the wall. The trial begins when the starting target disappears and the goal target appears between $0.5s$ and $1s$ later, to reduce anticipatory movements and learning effects. The participant moves the dominant hand to move the cursor to the goal target and selects by pressing the left button of the mouse bearing the optical marker used for pointing. The trial ends only when the participant successfully clicks the mouse button while the cursor is inside the goal target.

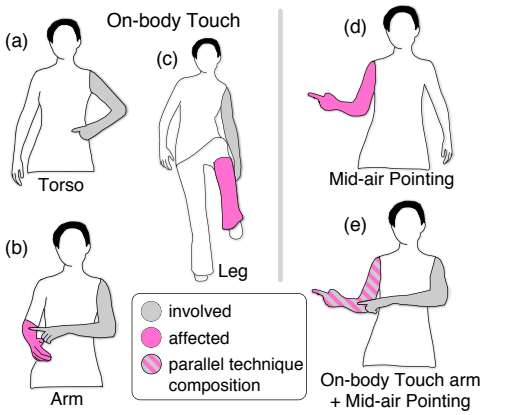


Figure 5. Body parts involved when touching the (a) torso, (b) arm, (c) leg; (d) mid-air pointing; and (e) in parallel, when the dominant hand points in mid-air and non-dominant hand touches the dominant arm.

POINTING+BODY (Fig. 4d): The starting position combines the above, with the non-dominant hand at the thigh and the dominant hand pointing to the starting target on the wall. The trial begins with the appearance of a body-target illustration and the goal target on the wall display. The participant first points the cursor at the goal target, then completes the trial by touching the designated on-body target. The trial ends only when the on-body touch occurs while the cursor is inside the goal target on the wall. As in the BODY ONLY condition, multiple body parts may be involved, sometimes with adverse effects. Fig. 5e shows the interaction between the dominant arm, which is trying to point to a target on the wall and the non-dominant arm, which is pointing at the dominant arm.

Training

Participants began by calibrating the system to their bodies, visually locating, touching and verifying each of the 18 body targets. They were then exposed to three blocks of six BODY ONLY trials, with the requirement that they performed two on-body touches in less than five seconds. They continued with three additional blocks to ensure they could accurately touch each of the targets. Next, they were exposed to all three levels of difficulty for the POINTING ONLY condition: easy, medium and hard, in a single block. Finally, they performed three additional blocks of the compound POINTING+BODY technique.

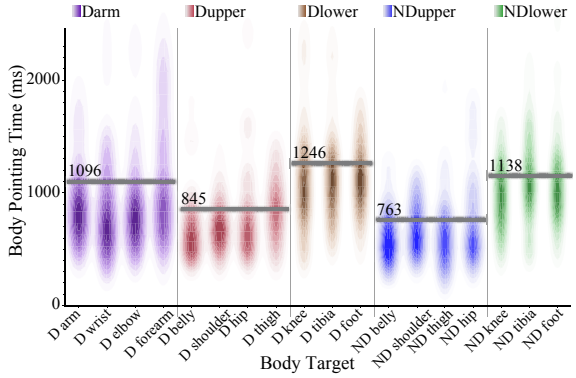


Figure 6. Mean BODY POINTING TIME is faster for both upper body targets (D UPPER and ND UPPER) compared to other targets. Horizontal lines indicate group means; performance within groups is consistent.

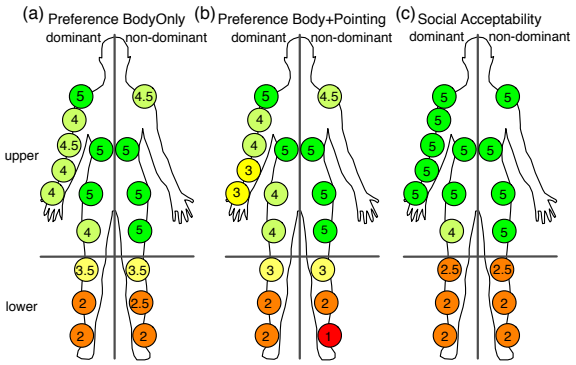


Figure 7. Median preference and acceptability rankings of on-body targets (from green = acceptable to red = unacceptable).

Results

Q1: Efficiency & acceptability of on-body targets

Our first research question asks which on-body targets are most efficient and which are socially acceptable. We conducted a full factorial ANOVA on the BODY ONLY condition, with PARTICIPANT as a random variable based on the standard repeated measures (REML) technique from the JMP 9 statistical package. We found no fatigue or learning effects.

Figure 6 shows the times for touching all 18 on-body targets, grouped into the five body areas. We found significant effects of BODY TARGET on BODY POINTING TIME: touching lower body targets is slower. Since BODY POINTING TIME is consistent for targets within a given target group, we report results according to target group, unless otherwise stated.

Overall, we found a main effect of BODY TARGET GROUP on TRIAL TIME ($F_{4,60} = 21.2$, $p < 0.0001$). A post-hoc Tukey test revealed two significantly different groups: body targets located on the upper torso required less than $1400ms$ to be touched whereas targets on the dominant arm and on the lower body parts required more than $1600ms$. Results are similar for BODY POINTING TIME with a significant effect of BODY TARGET GROUP only for the D UPPER group ($F_{3,45} = 5.07$, $p = 0.004$), specifically, targets on the dominant thigh are touched more slowly than those on the shoulder or torso. For BODY REACTION TIME, despite a significant effect, values are very close for each BODY TARGET GROUP ($530ms \pm 20ms$).

Participants were able to quickly touch on-body targets with an accuracy of 92.4% on the first try. A post-hoc Tukey test showed that targets on the dominant arm were more prone to errors than other body target areas (14.8% vs. 6% for dominant and non-dominant upper body and 2.9% for non-dominant lower body targets). Most errors obtained when targets were close to each other, i.e. when the participant's hand touched the boundary between the goal and a nearby target or when the dominant arm was held close to the torso, making it difficult to distinguish between the torso and arm targets. Touching lower body parts is, not surprisingly, slower, since these targets are further from the starting position and require more complex movements. However, the difference is small, about $200ms$ or 12% of global trial time.

Qualitative measures of Preference and Social Acceptance

Figure 7a shows that participants' preferences (median values of Likert-scale) for and within each BODY TARGET GROUP were consistent with performance measures: targets on the upper body parts were preferred over lower body parts (consistent with [18]) and the torso were slightly more preferred than on the dominant arm.

Interestingly, preferences for non-dominant foot and the dominant arm decrease when on-body touch interaction is combined with mid-air pointing (Fig. 7b). The latter is surprising, given that the most popular location for on-body targets in the literature is on the dominant arm. This suggests that interaction designers should explore alternative on-body targets as well. Social acceptability varies from highly acceptable (upper body) to unacceptable (lower body) (Fig. 7c).

Q2: Performance Trade-offs for compound techniques

The second research question examines the effect of combining two atomic interaction techniques, in this case BODY ONLY and POINTING ONLY, into a single compound technique. We treat these atomic techniques as baseline values to help us better evaluate the compound task.

Pointing Only task

Not surprisingly, hard pointing tasks are significantly slower (TRIAL TIME of $1545ms$ avg., $F_{2,30} = 40.23$, $p < 0.0001$) than medium ($1216ms$) or easy ($1170ms$) tasks, which are not significantly different from each other (Fig. 8a). POINTING REACTION TIME is also significantly slower for difficult ($498ms$) as opposed to medium ($443ms$) or easy ($456ms$) tasks. POINTING MOVEMENT TIME is significantly different for all three levels of difficulty: hard ($708ms$), medium ($511ms$) and easy ($435ms$).

Participants made few errors but occasionally had to relocate the cursor inside the goal target before validating the selection with the mouse. This occurred rarely (1.8% of all trials), but was significantly more likely for difficult pointing tasks (15%) ($F_{2,30} = 8.02$, $p = 0.0016$) and accounts for the differences in TRIAL TIME and POINTING MOVEMENT TIME.

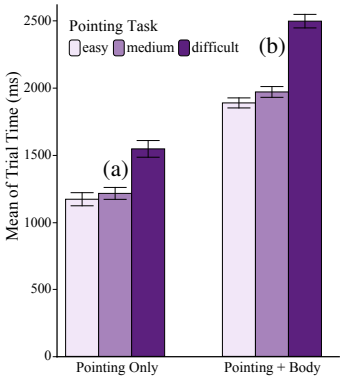


Figure 8. TRIAL TIME **for (a) Pointing Only and (b) Pointing + Body, by pointing difficulty.**

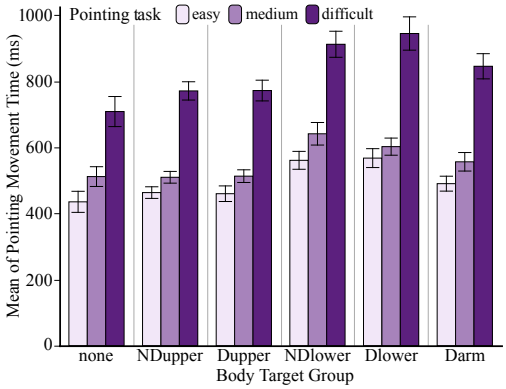


Figure 9. Interaction Pointing×Body on POINTING MOVEMENT TIME**.**

Compound Pointing plus Body task

Figure 8b shows that the combined MID-AIR POINTING and ON-BODY TOUCH task is significantly slower than MID-AIR POINTING alone for all levels of difficulty. TRIAL TIME is significantly slower for difficult MID-AIR POINTING ($2545ms$) than both medium ($1997ms$) and easy ($1905ms$) tasks. In fact, the easiest compound task is significantly slower that the hardest POINTING ONLY task.

BODY TARGET GROUP also has an effect on TRIAL TIME ($F_{4,60} = 34.1$, $p < 0.0001$) with the same significant groups as for BODY ONLY: TRIAL TIME is significantly faster when touching upper body targets (ND UPPER $= 1794ms$, D UPPER $= 1914ms$) than lower body targets (ND LOWER $= 2267ms$, D LOWER $= 2368ms$) or the dominant arm (D ARM $= 2401ms$). BODY REACTION TIME is faster than POINTING REACTION TIME, regardless of pointing difficulty.

Although we can see that the individual techniques are both more efficient than the compound technique, the question is why? Just how does ON-BODY TOUCH affect MID-AIR POINTING? Figure 9 shows interaction effects between the two elements of the compound tasks, by both BODY TARGET GROUP and pointing difficulty. While POINTING MOVEMENT TIME is close to the pointing baseline for all difficulties when MID-AIR POINTING is

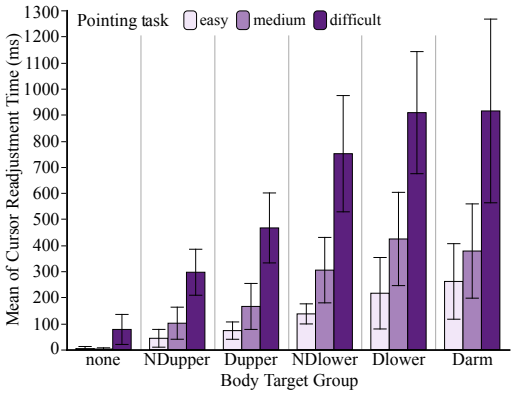


Figure 10. Effect of Pointing difficulty and BODY TARGET GROUP **on** CURSOR READJUSTMENT TIME**.**

combined with ON-BODY TOUCH on the upper body parts, we observe a stronger negative effect for the lower body parts and the dominant arm, especially for difficult pointing tasks.

This impact of ON-BODY TOUCH on the MID-AIR POINTING task does not only relate to the movement phase but also cursor readjustments. For the combined POINTING+BODY task, 31% of the trials required the participants to relocate the cursor inside of the target before validating the selection with a body touch, compared to only 6% for POINTING ONLY. Thus, we found significant effects of MID-AIR POINTING ($F_{2,30} = 59.64$, $p < 0.0001$), BODY TARGET GROUP ($F_{5,75} = 23.03, p < 0.0001$) and MID-AIR POINTING×BODY TARGET GROUP ($F_{10,150} = 8.45$, $p < 0.0001$) on CURSOR READJUSTMENT TIME. As shown in Figure 10, CURSOR READJUSTMENT TIME increases significantly for each level of difficulty of MID-AIR POINTING but selecting body targets on some BODY TARGET GROUP, especially in D LOWER and D ARM, affects the body configuration and requires even more time to relocate the cursor inside of the on-screen target.

This result reveals two important things: (1) touching the dominant arm while pointing affects the precision of pointing and requires "force-balance" (targets on D ARM); (2) touching targets on the lower body parts affects the precision of pointing and requires "movement-balance" (targets on ND LOWER and D LOWER). Overall, since the impact of both D LOWER and D ARM is similar, we observe that maintaining force-balance is as difficult as maintaining movement-balance during the pointing task, and that the difficulty in movement-balance is not only caused by standing on one leg, but by simultaneously crossing the body's sagittal plane (difference between D LOWER and ND LOWER).

Similarly, we studied the effect of MID-AIR POINTING on ON-BODY TOUCH by performing an ANOVA with the model MID-AIR POINTING[none /easy/medium/difficult]×BODY TARGET GROUP. We did not find any effect on BODY REACTION TIME. On BODY POINTING TIME, we did find a significant effect of BODY TARGET GROUP ($F_{4,60} = 38.69, p < 0.0001$), of MID-AIR POINTING ($F_{3,45} = 78.15$, $p < 0.0001$) and a significant MID-AIR POINTING×BODY TARGET GROUP interaction ($F_{12,180} = 2.28$, $p = 0.01$). The main effect of BODY TARGET GROUP is similar to the baseline (with ND UPPER and D UPPER significantly faster than all other

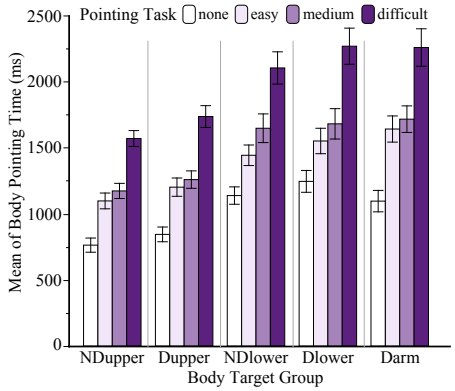


Figure 11. Interaction Pointing×Body on BODY POINTING TIME.

groups). The main effect of MID-AIR POINTING is also similar to those observed before, showing that difficult pointing tasks make simultaneous body touching slower than medium or easy pointing task. Obviously, these are all significantly slower than the BODY ONLY baseline.

More interesting, the MID-AIR POINTING×BODY TARGET GROUP interaction effect reveals the actual impact of MID-AIR POINTING on ON-BODY TOUCH. As shown in Figure 11: (i) the increasing difficulty of the pointing task increases BODY POINTING TIME. In fact, despite the fact that our task required body target selection as the final action, the reaction times indicate that both tasks start almost simultaneously (ON-BODY TOUCH even before MID-AIR POINTING). (ii) The increase in difficulty does not change the difference between the groups of targets, but rather amplifies them. ND UPPER and D UPPER remain the groups of targets that require less time to be touched.

In summary, the compound POINTING+BODY task involves interaction effects between the two atomic techniques, which not only incur a time penalty when the tasks are performed simultaneously, but also degrades pointing performance for MID-AIR POINTING (fixed in the world) when combined with a body-relative technique that involves and affects multiple limbs. However, our results also reveal that ON-BODY TOUCH on the lower parts of the body significantly impair the movement phase of pointing, and that the overall negative impact increases with the difficulty of the pointing task, especially when targeting on the pointing arm.

CONCLUSION

The BodyScape design space uses a body-centric approach to classify both existing and novel interaction techniques. The distributed nature of multi-surface environments highlights the need for combining interaction techniques, in series or in parallel, to accomplish more complex tasks. A body-centric approach can help predict possible interaction effects of body movements by (i) analyzing the spatial body-device relationship and (ii) proposing ways to decompose individual techniques into groups of body parts that are either *involved in* or *affected by* the interaction. We argue that studying compound interaction techniques from a body-centric perspective will lead to powerful guidelines for interaction design, both with and without physical devices.

We illustrate BodyScape by examining the effects of combining two multi-surface interaction techniques: *mid-air pointing* and *on-body touch*. This novel combination enables an eyes-free interaction with on-body targets while offering a rich set of mid-air pointing commands to access a remote virtual target on a large display. We ran a controlled experiment to study both techniques individually and in combination, investigating performance and acceptability of 18 on-body targets, as well as any interaction effects that obtain when the two techniques are combined. Participants were most efficient with targets on the torso and least efficient with targets on the lower body and on the dominant arm, especially in the combined condition: Reaching targets on the lower legs requires additional balance and touching the dominant arm impairs the precision of mid-air pointing because of the force applied on the pointing arm. Users consistently preferred targets located on the upper body.

Our results suggest three guidelines for designing on-body interaction:

G1 *Task difficulty*: On-body targets should be placed on stable body parts, such as the upper torso, when tasks require precise or highly coordinated movements.

G2 *Body balance*: Anticipatory movements, such as shifts in balance, can be detected to accommodate corresponding perturbations in a primary task, e.g. freezing an on-screen cursor. The precision of a pointing task can be adversely affected if users must also touch on-body targets that require a shift in balance or coordination, in particular, touching the dominant arm while it is performing a separate task.

G3 *Interaction effects*: Designers should consider which body parts negatively affect users' comfort while touching on-body targets as well as side effects of each task, such as reduced attention or fatigue that may lead to unexpected body positions or increases in errors.

Future work will develop more detailed descriptions of each limb's involvement in the interaction. We also plan to increase the predictability of BodyScape, following Card et al. [5], such as developing a Fitts-style pointing model for on-body touch.

ACKNOWLEDGEMENTS

We wish to thank the participants for their time and effort, as well as the anonymous reviewers for their helpful comments.

REFERENCES

1. Augsten, T., Kaefer, K., Meusel, R., Fetzer, C., Kanitz, D., Stoff, T., Becker, T., Holz, C., and Baudisch, P. Multitoe: high-precision interaction with back-projected floors based on high-resolution multi-touch input. In *Proc. UIST* (2010), 209–218.
2. Baudel, T., and Beaudouin-Lafon, M. Charade: remote control of objects using free-hand gestures. *CACM 36* (July 1993), 28–35.
3. Beaudouin-Lafon, M., Huot, S., Nancel, M., Mackay, W., Pietriga, E., Primet, R., Wagner, J., Chapuis, O., Pillias, C., Eagan, J., Gjerlufsen, T., and Klokmose, C.

Multi-surface Interaction in the WILD Room. *IEEE Computer 45*, 4 (2012), 48–56.

4. Boring, S., Baur, D., Butz, A., Gustafson, S., and Baudisch, P. Touch projector: mobile interaction through video. In *Proc. CHI* (2010), 2287–2296.
5. Card, S., Mackinlay, J., and Robertson, G. A morphological analysis of the design space of input devices. *ACM Trans. Inf. Syst. 9*, 2 (Apr. 1991), 99–122.
6. Casiez, G., Roussel, N., and Vogel, D. 1€ filter: a simple speed-based low-pass filter for noisy input in interactive systems. In *Proc. CHI* (2012), 2527–2530.
7. Chen, X. A., Marquardt, N., Tang, A., Boring, S., and Greenberg, S. Extending a mobile device's interaction space through body-centric interaction. In *Proc. MobileHCI* (2012), 151–160.
8. de Almeida, R., Pillias, C., Pietriga, E., and Cubaud, P. Looking behind bezels: french windows for wall displays. In *Proc. AVI* (2012), 124–131.
9. Dezfuli, N., Khalilbeigi, M., Huber, J., Müller, F., and Mühlhäuser, M. PalmRC: imaginary palm-based remote control for eyes-free television interaction. In *Proc. EuroiTV* (2012), 27–34.
10. Dickstein, R., and Laufer, Y. Light touch and center of mass stability during treadmill locomotion. *Gait & Posture 20*, 1 (2004), 41–47.
11. Dievendorf, L., Brook, D., and Jacob, R. J. K. Extending the user action notation (UAN) for specifying interfaces with multiple input devices and parallel path structure. Tech. rep., Naval Research Laboratory, 1995.
12. Feiner, S., MacIntyre, B., Haupt, M., and Solomon, E. Windows on the world: 2D windows for 3D augmented reality. In *Proc. UIST* (1993), 145–155.
13. Goldin-Meadow, S., and Beilock, S. L. Action's influence on thought: the case of gesture. *Perspectives on Psychological Science 5*, 6 (2010), 664–674.
14. Gustafson, S., Holz, C., and Baudisch, P. Imaginary Phone: learning imaginary interfaces by transferring spatial memory from a familiar dvice. In *Proc. UIST* (2011), 283–292.
15. Harrison, C., Ramamurthy, S., and Hudson, S. On-body interaction: armed and dangerous. In *Proc. TEI* (2012), 69–76.
16. Harrison, C., Tan, D., and Morris, D. Skinput: appropriating the body as an input surface. In *Proc. CHI* (2010), 453–462.
17. Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., and Fitzgibbon, A. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proc. UIST* (2011), 559–568.
18. Karrer, T., Wittenhagen, M., Lichtschlag, L., Heller, F., and Borchers, J. Pinstripe: eyes-free continuous input on interactive clothing. In *Proc. CHI* (2011), 1313–1322.
19. Klemmer, S., Hartmann, B., and Takayama, L. How bodies matter: five themes for interaction design. In *Proc. DIS* (2006), 140–149.
20. Krueger, M., Gionfriddo, T., and Hinrichsen, K. VIDEOPLACE—an artificial reality. In *Proc. CHI* (1985), 35–40.
21. Latulipe, C., Wilson, D., Huskey, S., Word, M., Carroll, A., Carroll, E., Gonzalez, B., Singh, V., Wirth, M., and Lottridge, D. Exploring the design space in technology-augmented dance. In *CHI Extended Abstracts* (2010), 2995–3000.
22. Li, F., Dearman, D., and Truong, K. Virtual shelves: interactions with orientation aware devices. In *Proc. UIST* (2009), 125–128.
23. Lin, S., Su, Z., Cheng, K., Liang, R., Kuo, T., and Chen, B. PUB - Point Upon Body: exploring eyes-free interactions and methods on an arm. In *Proc. UIST* (2011), 481–488.
24. Loke, L., Larssen, A. T., Robertson, T., and Edwards, J. Understanding movement for interaction design: frameworks and approaches. *Personal Ubiquitous Comput. 11*, 8 (Dec. 2007), 691–701.
25. Massion, J. Movement, posture and equilibrium: interaction and coordination. *Progress in Neurobiology 38*, 1 (1992), 35–56.
26. Mine, M., Brooks Jr, F., and Sequin, C. Moving objects in space: exploiting proprioception in virtual-environment interaction. In *Proc. SIGGRAPH* (1997), 19–26.
27. Nancel, M., Wagner, J., Pietriga, E., Chapuis, O., and Mackay, W. Mid-air pan-and-zoom on wall-sized displays. In *Proc. CHI* (2011), 177–186.
28. Pederson, T., Janlert, L.-E., and Surie, D. Towards a model for egocentric interaction with physical and virtual objects. In *Proc. NordiCHI* (2010), 755–758.
29. Rekimoto, J. Pick-and-drop: a direct manipulation technique for multiple computer environments. In *Proc. UIST* (1997), 31–39.
30. Scott, S., Carpendale, S., and Inkpen, K. Territoriality in collaborative tabletop workspaces. In *Proc. CSCW* (2004), 294–303.
31. Shoemaker, G., Tsukitani, T., Kitamura, Y., and Booth, K. Body-centric interaction techniques for very large wall displays. In *Proc. NordiCHI* (2010), 463–472.
32. Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. Real-time human pose recognition in parts from single depth images. In *Proc. CVPR* (2011), 1297–1304.
33. Su, R., and Bailey, B. Put them where? towards guidelines for positioning large displays in interactive workspaces. In *Proc. Interact* (2005), 337–349.
34. Tan, D., and Czerwinski, M. Effects of visual separation and physical discontinuities when distributing information across multiple displays. In *Proc. Interact* (2003), 252–255.
35. Wachs, J., Stern, H., Edan, Y., Gillam, M., Handler, J., Feied, C., and Smith, M. A gesture-based tool for sterile browsing of radiology images. *Journal of the American Medical Informatics Association 15*, 3 (2008), 321–323.
36. Wilson, A., and Benko, H. Combining multiple depth cameras and projectors for interactions on, above and between surfaces. In *Proc. UIST* (2010), 273–282.

B

CURRICULUM VITAE

Stéphane Huot

Maître de conférences – Univ. Paris-Sud
on leave at Inria (2011-2013)
↳ *in|situ| group (LRI - UMR 8623 &* Inria *Saclay–Ile-de-France)*

Laboratoire de Recherche en Informatique
Université Paris-Sud - Bât. 650 (PCRI)
91405 ORSAY Cedex, France
+33 6 76 04 51 81
+33 1 69 15 42 29
Stephane.Huot@lri.fr
http://www.lri.fr/~huot

Curriculum Vitae

Education

Doctorate (Ph.D.) in Computer Science (HCI)

Jan. 2001–July 2005 **Université de Nantes & École des Mines de Nantes**, *Nantes*, Computer Science Dpt. of École des Mines de Nantes, Interactive Design and Modeling group (CMI), *defended on July 12, 2005.*

Title *Une nouvelle approche pour la conception créative : De l'interprétation du dessin à main levée au prototypage d'interactions non-standard.*
A New Approach for Creative Design : From the Interpretation of Freehand Drawings to the Prototyping of Alternative Interaction Techniques.

Supervision Gérard Hégron (Professor, head of the CERMA–UMR CNRS 1563 laboratory) and Cédric Dumas (Associate Professor, École des Mines de Nantes).

Jury Referees : Pierre Leclercq (Professor, Université de Liège, Belgium), Philippe Palanque (Professor, Université Paul Sabatier Toulouse 3) ; President : Henry Briand (Professor, Université de Nantes) ; Examiner : Claudie Faure (Research Scientist, CNRS, ENST Paris) ; Invited : Jean-Daniel Fekete (Research Director, INRIA Saclay–Ile-de-France).

Keywords Human-Computer Interaction, Creative Design, 3D Modeling, Pen-based Interfaces, Input Devices, Interaction Techniques, Multimodal Interaction, Post-WIMP Interfaces, GUI Toolkits.

DEA (M.Sc.) in Computer Science

Sept. 1999–Sept. 2000 **Université de Nantes**, *Nantes*, Computer Science Dpt. (IRIN), Computer Science Dpt. of École des Mines de Nantes, Interactive Design and Modeling group (CMI), *defended on September 2000.*

Title *Reconstruction de bâtiments 3D à partir d'images.*
3D Modeling of Buildings from Images.

Supervision Christian Colin (Associate Professor, École des Mines de Nantes).

Undergraduate & Graduate Education '*DEUG*' MIAS (Mathematics, Physics and Computer Science), '*Licence*' and '*Maîtrise*' (Computer Science) at Université de Limoges.

Employment

Sept. 2011–present **on leave at Inria**, *Inria Saclay–Ile-de-France*, Saclay.
in|situ| group.

Sept. 2007–present **Associate Professor**, *IUT d'Orsay – Université Paris-Sud*, Orsay.
Member of the 'Laboratoire de Recherche en Informatique'–UMR CNRS 8623, in|situ| group (joint project between LRI & Inria Saclay–Ile-de-France).
Lecturer at the Computer Science Department of IUT d'Orsay.

Feb. 2007–Aug. 2007 **Postdoctoral Fellow**, *LRI-CNRS & Inria*, Orsay.
AVIZ group (Inria Saclay–Ile-de-France), with Jean-Daniel Fekete.
Research topics : Information Visualization.

Feb. 2006–Feb. 2007 **Postdoctoral Fellow**, *Telecom ParisTech*, Paris.
I3 group (Information, Interaction, Intelligence), Computer Science and Networks Dpt.–UMR CNRS 5141, with Eric Lecolinet, in collaboration with Alcatel-Lucent.
Research topics : Interaction in mobility conditions.

Awards

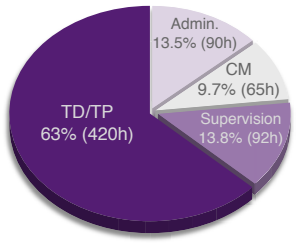
2013 ACM *CHI 2013* "Honorable mention" (top 5%) [I.1].

2012 ACM *CHI 2012* "Best paper" (top 1%) [I.2].

ACM *CHI 2012* "Honorable mention" (top 5%) [I.3].

2009 *IHM 2009* "Best paper" [D.2].

Teaching



Before being on leave at Inria for two years, I was teaching at the Computer Science Department of IUT d'Orsay (the Institute of Technology of the Université Paris-Sud which delivers 2- and 3-years undergraduate degrees). I have taught GUI Programming and Object Oriented Programming courses, and supervised students' projects and internships. Between 2008 and 2011, my yearly teaching load was of an average 200 hours, with about 35 additional hours of administrative duties : courses management, exams supervision, in charge of *Apogee* for the department (students registration and grades management system). The chart on the left gives an overview by activity (2008-2011).

Note : CM=Lecture, TD=Tutorial and TP=Practical. Total teaching load is counted in TD hours, with 1h CM=1.5h TD/TP.

Université Paris-Sud, Computer Science Department

2010–2011 *9h eq. TD*

Master (M2R IAC) **Human-Computer Interaction**, 6h CM, *Lecturer*.
Two lectures for the HCI course in the IAC Master's degree program : "Graphical Interaction : Interaction Styles and Elementary Tasks" (3h) and "Graphical Interaction : Engineering of Interactive Systems, Software Architectures and Toolkits" (3h).

Université Paris-Sud, IUT d'Orsay

2010–2011 *203,5h eq. TD*

DUT S2 (1st year) **Graphical User Interface Programming**, 16h CM + 67,5h TD/TP, *Lecturer*.
Introduction to web programming (XHTML/css), event-driven programming, GUIs (WIMP), basics of engineering of interactive systems and UI-Data connection.
↳ *In charge of this course for 3 years, I revisited its contents by adding basics of interactive systems programming as well as an introduction to web technologies.*

DUT S3 (2nd year) **Object Oriented Programming – Java**, 49h TD, *Co-Lecturer*.
Basics of OOP, Java language, MVC, GUI programming (AWT and Swing).
↳ *From 2008 to 2010, I was in charge of the lectures on "MVC" & "GUI Programming".*

DUT S4 (2nd year) **Synthesis Project**, 27h TD.
Last projects before end studies internship. The project is conducted by a group of 13 to 15 students during 7 weeks and they have two tutorial sessions of 3h each week.
↳ *My objective is to help students in practicing and mastering theoretical concepts with attractive topics and novel technologies : mobile devices, mutlitouch, novel input devices.*

DUT S2 (1st year) **Tutored Projects**, 16h TD.
Supervision of groups of 3-4 1st year students for their first Computer Science project.

DUT S4 (2nd year) **Internship Supervision**, 8h TD.
Supervision of students during their end studies internship (April-June).

Licence Pro. SRSI **Apprenticeship Supervision**, 8h TD.
Supervision of an apprentice student.

2009-2010 *213h eq. TD* (see details above).

DUT S2 **Graphical User Interface Programming**, 15h CM + 67,5h TD, *Lecturer*.

DUT S3 **Object Oriented Programming – Java**, 6h CM + 51h TD, *Co-Lecturer*.

DUT S4 **Synthesis Project**, 27h TD.

DUT S2 **Tutored Projects**, 16h TD.

DUT S4 **Internship Supervision**, 8h TD.

Licence Pro. SRSI **Apprenticeship Supervision**, 8h TD.

2008-2009 *192,5h eq. TD* (see details above).

DUT S2 **Graphical User Interface Programming**, 15h CM + 60h TD, *Lecturer*.

DUT S3 **Object Oriented Programming – Java**, 7h CM + 49h TD, *Co-Lecturer*.

DUT S4 **Synthesis Project**, 22,5h TD.

DUT S2 **Tutored Projects**, 16h TD.
DUT S4 **Internship Supervision**, 12h TD.

2007-2008 *150h eq. TD, unload of 42h for the 1st year of service* (see details above).
DUT S2 **Graphical User Interface Programming**, 30h TD.
DUT S3 **Object Oriented Programming – Java**, 49h TD.
DUT S4 **Synthesis Project**, 45h TD.
DUT S2 **Tutored Projects**, 16h TD.
DUT S4 **Internship Supervision**, 10h TD.

Professional Service

Administrative Activities

Université Paris-Sud

– Member of the University Consultative Specialists Committee in Computer Science (hiring and promotions at the department level), CCSU 27 (since 2010).

IUT Orsay

– In charge of the registration and grading management system (Apogée) for the Computer Science Department (2008–2011).

Inria - in|situ|

– in|situ| representative at the Inria "Mobile Services Initiative".
– in|situ| correspondent (alternate) for the Inria-Saclay "Equipment Committee".

LRI

– Member of the "Equipment Committee".
– Member of the "Web Committee".

Evaluation of Research

Program Committees

International Conferences

– ACM CHI'13 (Associate Chair, subcommittee "*Interaction Techniques and Devices*").
– IFIP TC13 Interact 2009 and 2013.

Domestic Conferences

– IHM 2010 (Papers co-Chair), 2011 (Demonstrations co-Chair) and 2013 (Program co-Chair).
– UbiMob 2008 and 2009.
– Workshop 'Visual Data-Mining', EGC 2011.

Reviewing

– Journal d'Interaction Personne-Systèmes (JIPS).
– International Conferences : ACM CHI (since 2004), ACM UIST (since 2005), IFIP TC13 Interact (since 2009), ACM EICS (2013), TEI (2013), Mobile HCI (2012), Intelligent User Interfaces (2012), 3D User Interfaces (2011), NordiCHI (2006–2008).
– Domestic Conferences : IHM (since 2003), UbiMob (2008–2009).

Evaluation Committees and Invited Expertise

– Expert reviewer for ANR (French National Research Agency) – AAP JCJC - SIMI 2, 2012.
– Hiring Committees at Univ. Paris-Sud – Orsay, in 2009, 2010 and 2011.
– Hiring Committee at Univ. Paul Sabatier – Toulouse, in 2012.
– Hiring Committee at Univ. Bordeaux 1 – Bordeaux, in 2013.

Ph.D. Juries

– Mathias Baglioni. *Nouvelles interactions physiques pour dispositifs mobiles.* Télécom ParisTech, Ph.D. Thesis, April 25, 2012. Examiner.
– Julie Wagner. *A Body-centric Framework for Generating and Evaluating Novel Interaction Techniques.* Université Paris-Sud & Inria, Ph.D. Thesis, December 6, 2012. Examiner (co-advisor).
– Émilien Ghomi. *Designing Expressive Interaction Techniques for Novices Inspired by Expert Activities : The case of musical practice.* Université Paris-Sud, Ph.D. Thesis, December 17, 2012. Examiner (co-advisor).

Conferences Organization

IHM 2011 — Conference of the Francophone Association in HCI, Demonstrations co-Chair.

RJC-IHM 2006 — French Symposium for Junior Researchers in HCI, co-Chair.

Invited Talks and Seminars

RWTH Aachen, Aachen, Germany — "Novel Interaction Techniques and Engineering of Interactive Systems", Media Computing Group seminar, October 2011.

Institut Farman, ENS Cachan, Cachan, France — "Designing Advanced Interaction Techniques for Interactive High-Resolution Visualization Platforms", Institut Farman annual seminar, November 2012.

Societies

– Member of the AFIHM Scientific Board (CPPMS), the Francophone Association in Human-Computer Interaction (since 2012).
– Member of the AFIHM Council, the Francophone Association in Human-Computer Interaction (2005–2008).
– Member of the ACM SIGCHI Paris Chapter (since 2010).

Funded Projects & Technology Transfer

Funded Projects

since 2012 — **Digipods : Distant Collaborative Interaction Between Heterogeneous Visualization Platforms**, *"Équipement mi-lourd SESAME 2012" of Région Île-de-France*, 850K€ (total : 1.9M€).

Partners — FCS Campus Paris-Saclay, Université Paris-Sud, Inria, CNRS, CEA-LIST and Télécom ParisTech.

Role — **Coordinator and principal investigator**.

since 2011 — **DIGISCOPE (High-Performance Infrastructure for Collaborative Interactive Visualization)**, *ANR-EQUIPEX 2010*, coordinator : Michel Beaudouin-Lafon, LRI–in|situ|, 6.7M€ (total : 22M€).

Partners — FCS Campus Paris-Saclay, Univ. Paris-Sud, CNRS, CEA, Inria, Télécom-ParisTech, Ecole Centrale, UVSQ, ENS Cachan and Maison de la Simulation.

Role — **Co-Chair of the Technical Committee** (management of hiring and purchasing processes, supervision of project engineers), **node-manager for WILD** (Université Paris-Sud platform), **specification of WILDER** (Inria platform), **coordination and participation** in related future research projects.

since 2009 — **WILD (Wall-sized Interaction with Large Datasets)**, *Région Ile-de-France, Digiteo, CNRS, Inria, Inria-MSR, Université Paris-Sud, ANR*, (coordinator : Emmanuel Pietriga, Inria–in|situ|), 429K€.

Partners — Université Paris-Sud, Inria, CNRS.

Role **Manager of the platform since July 2012, coordination and participation** in related research projects.

2008-2011 **iStar**, *ANR*, (coordinator LRI : Michel Beaudouin-Lafon, LRI–in|situ|).

Partners LRI, ENAC, IntuiLab, Anyware Technologies.

Role **Participant**.

Technology Transfer

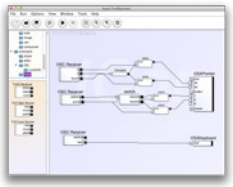
since 2011 **VCoRE (Visual COmputing Runtime Environment)**, *Inria/IGD Fraunhofer Institute*, (coordinators : Bruno Raffin & Jean-Christophe Lombardo, Inria).

Partners Inria (Grenoble, Lille, Rennes, Saclay, Sophia Antipolis), IGD Fraunhofer Institute.

Role **Coordinator and investigator for in|situ|/Inria Saclay–Île-de-France**.

Software Development & Distribution

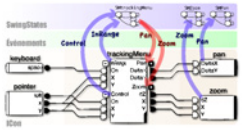
Input Configurator



`http://inputconf.sourceforge.net/`

A Java toolkit and a visual editor for the support and the dynamic configuration of a large range of input devices and advanced interaction techniques [I.13].
- Principal developer and maintainer (formerly by P. Dragicevic & J.-D. Fekete).
- Distributed under the BSD license and used in several projects as the core component for input and interaction management.
- Will be refactored and extended (see FlowStates and WILDInputServer below) as part of the VCoRE technological development action of Inria (see p.6).

FlowStates



`http://www.lri.fr/~appert/FlowStates/`

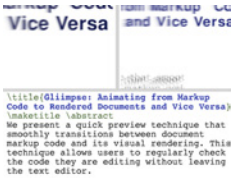
A Java toolkit that combines state machines and data flow paradigms (ICon) for prototyping and configuring advanced interaction techniques. [D.2].
- Principal maintainer (developed with C. Appert).
- Will be integrated into ICon distribution.

WILDInputServer

`http://inputconf.sourceforge.net/`

A standalone application based on ICon, that improves the control of interactions in large multi-surface environments (wall-sized displays, VR platforms, etc.) [I.7].
- Developer and maintainer (part of the jBricks framework, with E. Pietriga & R. Primet).
- Add support for specific input devices (e.g., trackers) and protocols (OSC, TUIO, vrpn) to ICon, as well as a remote control protocol and a dynamic plugins mechanism.
- Will be integrated into ICon distribution as part of the VCoRE technological development action of Inria (see p.6).

Gliimpse



`http://www.aviz.fr/gliimpse/`

A Markup Language editor that allows to smoothly transition from the code to the rendered document [I.5].
- Developer and maintainer, with P. Dragicevic.
- Looking for opportunities to integrate Gliimpse in a real editor.

TorusDesktop

`http://insitu.lri.fr/TorusDesktop`

A MacOS X application that implements a new pointing technique based on 'cursor-wrapping' in a real desktop environment [I.6].
- Developer and maintainer.

WikipediaViz `http://reactivitiz.lri.fr/mediawiki/index.php`

A set of visualizations designed for casual users of Wikipedia, that reveal some important data about an article and help to assess potential quality problems. [I.9].
- Former designer and developer (with J.-D. Fekete & F. Chevalier).
- Implemented on a Wikipedia mirror site as a proof of concept and for evaluation purposes.

SpiraClock & **HeliCal** `http://www.emn.fr/z-info/spiraclock/` & `http://www.lri.fr/~dragice/helical/`

An interactive technique for the continuous and non intrusive visualization of temporal events. [I.15].
- Design and development, with P. Dragicevic.
- Several applications were developed for use with different calendar systems (Outlook, Google, etc.). Downloaded more than 5000 times.

Supervision of Research

Ph.D. Students

Jan. 2013 — **Justin Mathew** (advisor, 80%)
with Brian F.G. Katz (LIMSI-CNRS) & Hervé Roux (Digital Media Solutions)

Funding — ANRT CIFRE with Digital Media Solutions.

Title — New Visualization and Interaction Techniques in Spatial Composition for Mixing Interfaces in the Context of 3D Spatial Audio.

Keywords — Interaction Techniques, Visualization, Audio Mixing Interfaces, 3D Spatial Audio.

Sept. 2010–Dec. 2012 — **Julie Wagner** (joint advisor, 50%)
with Wendy E. Mackay (Inria) – Ph.D. started in 2009, joint advisor since 2010

Funding — Inria CORDI.

Defense — December 6, 2012.

Title — A Body-centric Framework for Generating and Evaluating Novel Interaction Techniques.

Keywords — Body-centric Interaction, Kinematic Chain, Proprioception, Multi-surface Environments.

Publications — "BiTouch and BiPad : Designing Bimanual Interaction for Hand-held Tablets" [I.4] – "Left-over Windows Cause Window Clutter... But What Causes Left-over Windows ?" [D.1] – "A Body-centric Design Space for Multi-surface Interaction" [I.1].

Sept. 2008–Dec. 2012 — **Émilien Ghomi** (joint advisor, 60%)
with Michel Beaudouin-Lafon (Université Paris-Sud)

Funding — MENRT & ATER.

Defense — December 17, 2012.

Title — Designing Expressive Interaction Techniques for Novices Inspired by Expert Activities : The case of musical practice.

Keywords — Expert practices, Direct Interaction, Multitouch Interaction, Rhythmic Interaction, Interaction Techniques, Music Practice and Experience.

Publications — "Conception et apprentissage des interactions tactiles : le cas des postures multi-doigts" [WS.2] – "Using Rhythmic Patterns as an Input Method" [I.2].

Engineers

Nov. 2012–Oct. 2014 — **Software Engineer**, INRIA ("Ingénieur Jeune Diplômé"), ADT VCoRE.
Improving and extending the ICon toolkit and the WILDInputServer software for their integration into the VCoRE framework.

since June 2012 — **Software Engineer**, CNRS (IR), Equipex Digiscope.
Technology watch, feasibility studies and software developments in System and Networks for the Digiscope platform.

since Dec. 2011 — **Software Engineer**, CNRS (IE), Equipex Digiscope.
Technology watch, feasibility studies and software developments in Computer Graphics and Interaction for the Digiscope platform.

M.Sc. Students

Dec. 2010–July 2011 — **Can Liu**, RWTH Aachen University, Germany (80%)
with Jonathan Diehl and Jan Borchers, RWTH Aachen University
Six months research internship under my supervision at in|situ|.

Title — Exploring Mobile Augmented Reality Instructions to Assist Operating Physical Interfaces.

Keywords — Mobile Interaction, Mobile Augmented Reality, Context-aware, Reminder, Physical Interaction, PIM.

Publications "Mobile Augmented Note-taking to Support Operating Physical Devices" [WS.1] – "Evaluating the Benefits of Real-time Feedback in Mobile Augmented Reality with Hand-held Devices" [I.3].

Feb. 2010–July 2010 **Quentin Roy**, Univ. Lyon 2 and École Polytechnique Univ. Nantes (70%)
with James Eagan, postdoc at LRI, now Assistant Professor at Télécom ParisTech
Six months research internship under my supervision at in|situ|.

Title Transformation and Adaptation of Interaction Techniques as Part of the GUI Teleportation Process.

Keywords Interfaces "Deconstruction" and Teleportation, Adaptability, Heterogeneous and Multi-device Environments, Engineering of Interactive Systems.

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of May 16, 2013 (`Stuf's HDR` version 0.9).