Direct and expressive interaction on the desktop: increasing granularity, extent, and dimensionality

Jonathan Aceituno

Jonathan Aceituno

Direct and expressive interaction on the desktop
Increasing granularity, extent, and dimensionality

Interaction directe et expressive sur le bureau: améliorer la granularité, l'étendue, et la dimensionnalité des actions de l'utilisateur

RÉSUMÉ

L'ordinateur personnel, de bureau ou portable, permet le travail intellectuel et les activités créatives à travers une interaction directe et expressive. Mais l'impression de "directitude" disparaît quand les actions deviennent complexes, et l'expressivité est souvent favorisée au détriment d'une séparation spatiotemporelle, d'une charge cognitive, ou de suites complexes d'opérations. Cette thèse propose de mieux exploiter les périphériques d'entrée standard de façon à augmenter de façon directe la granularité, l'étendue, et la dimensionnalité des actions utilisateur.

Nous montrons d'abord que la granularité des mouvements du pointeur peut être augmentée sans déranger son fonctionnement habituel si la fonction de transfert du pointage prend en compte les périphériques, les capacités de l'utilisateur, et le modèle de données manipulé. Une limite de l'interaction subpixel ainsi permise est la résolution utile (le plus petit déplacement pouvant être produit avec un périphérique), pour laquelle nous proposons un protocole expérimental.

Nous étudions ensuite l'edge-scrolling, une technique qui étend le déplacement d'objets et la sélection au-delà des limites d'une vue en la faisant défiler. Nous montrons plusieurs problèmes d'utilisabilité par l'examen de 33 implémentations, un sondage et des expériences contrôlées. Nous évitons ces problèmes en proposant push-edge et slide-edge, qui montrent des performances similaires aux techniques existantes.

Enfin, nous décrivons trois utilisations réussies de l'ordinateur portable comme instrument de musique permettant le contrôle simultané temps-réel de plusieurs paramètres sonores, en détaillant quelques problèmes de conception.

ABSTRACT

Desktop and laptop personal computers enable knowledge work and creative activities by supporting direct and expressive interaction. But the feeling of directness often disappears when users perform complex actions: expressiveness is ensured at the cost of spatiotemporal separation, cognitive load, or complex operation sequences. This thesis proposes to solve this problem by harnessing the unexploited capabilities of standard input devices and familiar actions performed on them. We investigate how this enables direct increases of the granularity, extent, and dimensionality of user actions.

First, we show that the granularity of pointer movements can be increased a hundred-fold without impeding normal behavior if pointing transfer functions include device characteristics, user capabilities, and the manipulated data model, thus allowing subpixel interaction. This is limited by the useful resolution, the smallest displacement a user can produce using a pointing device, for which we propose an experimental protocol.

Second, we study the design space of a widely used technique, edge-scrolling, that extends dragging actions past a viewport edge by scrolling. We reverse engineer 33 existing implementations, and highlight usability problems through a survey and experiments. We also propose push-edge and slide-edge scrolling, two position control techniques that provide performance comparable to rate control without the shortcomings.

Third, we describe three ways of using a standard laptop as a musical instrument, allowing simultaneous multiparametric control of sound synthesis in real time, together with design considerations and examples of successful uses.

Contents

List of Figures

Publications

The publications listed below are based on work and research-related activities completed during my PhD studies. For those related to the work presented in this dissertation, the corresponding chapters are noted in parentheses.

INTERNATIONAL CONFERENCE PAPERS

1. Sylvain Malacria, **Jonathan Aceituno**[*], Philip Quinn, Géry Casiez, Andy Cockburn, and Nicolas Roussel. 2015. Push-Edge and Slide-Edge: Scrolling by Pushing Against the Viewport Edge. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'15)*. ACM, 2773–2776. (Chapter 7)

2. **Jonathan Aceituno**, Géry Casiez, and Nicolas Roussel. 2013. How Low Can You Go?: Human Limits in Small Unidirectional Mouse Movements. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, 1383–1386. (Chapter 5)

3. Nicolas Roussel, Géry Casiez, **Jonathan Aceituno**[†], and Daniel Vogel. 2012. Giving a Hand to the Eyes: Leveraging Input Accuracy for Subpixel Interaction. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, 351–358. (Chapter 4)

NATIONAL CONFERENCE PAPERS

4. **Jonathan Aceituno** and Nicolas Roussel. 2014. The Hotkey Palette: Flexible Contextual Retrieval of Chosen Documents and Windows. In *Proceedings of the 26th French-speaking Conference on Human-Computer Interaction (IHM '14)*. ACM, 55–59.

DEMOS AND VIDEOS

5. **Jonathan Aceituno** and Ludovic Potier. 2014. L'ordinateur portable comme instrument de musique. In *Extended Abstracts of the 26th French-speaking Conference on Human-Computer Interaction (IHM '14), Démonstrations*. Best demonstration award. ACM, 3–4. (Chapter 8)

6. **Jonathan Aceituno** and Ludovic Potier. 2014. The Secret Life of Computers. In *Extended Abstracts of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14), Video Showcase*. ACM, 151–152. (Chapter 8)

---

[*]Sylvain Malacria led the writing of the paper and motivated the experiment, which he also performed. I designed the techniques, developed the experimental software, analyzed the results, and was a main contributor in the writing, theoretical developments, and experimental design.

[†]Nicolas Roussel and Géry Casiez provided the first theoretical developments and led the writing of the paper. I designed and developed the prototype systems, and was a main contributor in the writing, as well as in theoretical and technical developments.

DOCTORAL CONSORTIUM

7. **Jonathan Aceituno**. 2013. Manipulation directe et accès indirect. In *Extended Abstracts of the 25th French-speaking Conference on Human-Computer Interaction (IHM '13), Rencontres doctorales*. ACM, 25–28.

POPULAR SCIENCE

8. **Jonathan Aceituno** and Nicolas Roussel. 2014. Douglas Engelbart, inventeur et visionnaire. In *Bulletin de la Société Informatique de France* 2, 153–164. Joint publication with the Interstices web site.

CONFERENCE PROCEEDINGS

9. **Jonathan Aceituno**, Renaud Blanch, Stéphane Huot, Martin Hachet, and Nicolas Roussel, editors. 2013. *Proceedings of the 25th French-speaking Conference on Human-Computer Interaction (IHM '13)*. ACM.

Remerciements

Je voudrais d'abord remercier David Gross-Amblard et Rosane Ushirobira, qui sont en quelque sorte mes anges-gardiens académiques. Depuis mes premières années à Dijon, ils m'ont patiemment écouté et conseillé, et ils ont su calmer mon indécision vocationnelle en m'orientant comme par hasard vers le domaine "presque artistique" (et un peu magique) de l'interaction homme-machine. Mais si déterminante que soit leur contribution à ma formation, ils ne sont pas les seuls à en partager la responsabilité. Je pense par exemple à Patrice Roy, sans nul doute le meilleur — et le plus fun — enseignant et programmeur que je connaisse, et à Julien Castet, qui a accompagné mes premiers pas dans la recherche dans la bonne humeur.

Merci à Nicolas Roussel pour m'avoir guidé sur les chemins de la recherche et de l'IHM, tout en me laissant la liberté de les suivre à ma façon. Par-delà les élucubrations, les jambons à roulettes et les moments les plus difficiles que j'ai pu lui imposer, il m'a prêté, sans compter, son soutien, sa confiance et ses bouquins, et j'espère pouvoir les lui rendre un jour. Il est toujours pour moi une source de sagesse et de stimulation intellectuelle inépuisable, et je n'imagine pas de meilleur maître Zen. Je remercie aussi Géry Casiez, qui a contribué à un grand nombre de travaux présentés dans cette thèse, pour ses conseils, sa rigueur scientifique, et pour tout ce qu'il m'a appris. Merci également à Laurent Grisoni pour son engagement et son soutien. Votre dévouement et la diversité de vos points de vue font de Mint et de Mjolnir un environnement épanouissant et enrichissant que je suis bien triste de devoir quitter.

Je suis particulièrement honoré d'avoir pu soutenir devant un jury composé de personnes dont j'admire les travaux. Je ne saurais assez les remercier pour le temps et l'intérêt qu'ils m'ont accordé, ainsi que pour les riches discussions que nous avons pu avoir.

Merci aux membres et aux ex-membres de l'équipe avec qui j'ai passé ces quatre dernières années, pour les discussions, projets, soirées, jam sessions (de bureau ou pas), life coaching, expériences de pointage en environnement virtuel distribué et super cadeaux de thèse. Merci également à tous les amis que j'ai négligé pendant ces années ; les pédoules, gangsters d'amour, pinarchistes, zemblanitiens, scarface-clubbers, mofos ou voyous de Chagny, celles et ceux de Raspatakouet, d'Eiffel, de Mirande ou de Talence. Le souvenir des moments vécus, les trop courtes retrouvailles et la perspective de prochainement revenir vous voir me donnent du courage. Ça, et le jazz-rock un peu smoothy.

Merci à ma famille qui m'a apporté un soutien constant malgré les tempêtes que nous avons traversé. Je dédie ma thèse à Carmen, David et Roland, qui sont partis trop tôt, ainsi qu'à Olivia qui vient de nous rejoindre. Merci aussi à ma belle-famille qui, en plus de me soutenir autant que ma vraie famille, m'a aussi offert le plus beau des cadeaux : la personne avec qui je suis heureux de partager ma vie. Lénou, je te dois tout. *Dal mek creon te shree tal'ma.*

"It is a profoundly erroneous truism, repeated by all copy-books and by eminent people when they are making speeches, that we should cultivate the habit of thinking of what we are doing. The precise opposite is the case. Civilization advances by extending the number of important operations which we can perform without thinking about them. Operations of thought are like cavalry charges in a battle — they are strictly limited in number, they require fresh horses, and must only be made at decisive moments."

— Alfred North Whitehead, *An Introduction to Mathematics*.

Introduction

> Imagine having your own self-contained knowledge manipulator in a portable
> package the size and shape of an ordinary notebook. Suppose it had enough power
> to outrace your senses of sight and hearing, enough capacity to store for later
> retrieval thousands of page-equivalents of reference materials, poems, letters, recipes,
> records, drawings, animations, musical scores, waveforms, dynamic simulations, and
> anything else you would like to remember and change. We envision a device as small
> and portable as possible which could both take in and give out information in
> quantities approaching that of human sensory systems. Visual output should be, at
> the least, of higher quality than what can be obtained from newsprint. Audio output
> should adhere to similar high-fidelity standards. There should be no discernible
> pause between cause and effect. One of the metaphors we used when designing such
> a system was that of a musical instrument, such as a flute, which is owned by its user
> and responds instantly and consistently to its owner's wishes. Imagine the absurdity
> of a one-second delay between blowing a note and hearing it!
>
> — Alan Kay and Adele Goldberg, *Personal Dynamic Media (1977)*.

Surely, there is no need to recourse to imagination to picture that magical personal notebook
when you can probably find one just by looking around you, at your desk, or on your lap. The
future Kay and Goldberg (1977) tried to invent some 40 years ago now exists as the modern
laptop, a computer that can "be owned by everyone and [has] the power to handle virtually all of
its owner's information-related needs" (Kay and Goldberg, 1977), and that fits, even exceeds, the
above technical description of their DynaBook. Is that to say the quest for a personal medium
for creative thought, one that assists thinking, is over? At a time where computing has bled from
our desks into our pockets, our houses, and now pervades the public space in a myriad of forms,
there are several reasons to believe that laptops and desktops remain relevant in human-computer
interaction (HCI) research.

First, this form of computing has stood the test of time, it addresses a wide variety of application
domains, and it is unlikely to disappear anytime soon. The familiar keyboard has only slightly
evolved compared to the QWERTY layout invented in 1878 (Noyes, 1983), and it is firmly
entrenched in users' core computing skills. The mouse dates back to the sixties and consistently
outperforms other pointing devices like joysticks and trackballs in various tasks (e.g., English
et al., 1967; Card et al., 1978; MacKenzie et al., 1991). The touchpad replaced the mouse in
laptops, but their operation remains similar in graphical user interfaces: they allow to move a
pointer on the display, the primary means to manipulate on-screen content. This is how users
have interacted with desktops (and laptops) for almost 30 years, and the input and output devices
they use have proven well adapted to graphical manipulation, command selection, and text entry.

Second, little is known about the basic mechanisms encountered daily by millions of desktop and laptop users. Animated transitions are now heavily used to smoothly convey changes in graphical interfaces, but how to best design them is still not well understood (Chevalier et al., 2014). The state of affairs is somewhat similar for the design of velocity-dependent transfer functions used in modern systems for pointing (Casiez and Roussel, 2011), of the automatic scrolling functionality (studied in this work) that activates once a dragged object approaches the border of a window, or of the visual indications that promote the use of command shortcuts (Malacria et al., 2013). Research on laptop and desktop interaction relies on, but also nourishes, our understanding of more fundamental issues.

One of these issues is the apparent tradeoff between ease of use and power (Beaudouin-Lafon and Mackay, 2000; Beaudouin-Lafon, 2005): complex and featureful systems often offer complicated, indirect interfaces, that remain hard to use even after extensive learning, but simplistic ones offer poor expressive power, making them less useful. Arguably, one cannot expect to be able to do complex things more easily than simple ones: compare a child drawing of a house and a technical drawing of the same house with complete dimensions at various levels of detail. But the interfaces that allow complex things to be done are often more complicated than needed. For example, to create a line with accurate dimensions in an architectural design application, one does not just draw it accurately: one has to first draw it using the mouse, then reach the keyboard, hit a command shortcut to edit one of the dimensions, and type in an exact numerical value. Even though the expert architect can perform this sequence of operations without even thinking about it, the large number of steps and the time it takes to see the results would certainly discourage her to engage in ambitious "what-if" explorations. More generally, the expression of complex things is often less direct than it could be.

We believe that harnessing the unexploited capabilities of our standard input devices and of familiar actions we can perform on them, such as pointing, can improve the expressiveness of users of laptops and desktops with minimal indirectness. In this work, we focus on increasing the range of levels of detail (*granularity*), the covered area (*extent*), and the useful degrees of freedom (*dimensionality*) accessible through such actions. Our goal is to determine the degree to which these improvements can be obtained without threatening the directness of interaction, and without recourse to non-standard input devices. The rest of this chapter will set the scene with an overview of the related research, detail the main problem, goals, and contributions of this research, and provide an overview of the structure of the dissertation.

1.1. Background

The vision of the computer as a medium that can be used to help us think has been floating around since the dawn of interactive computing (Bush, 1945; Licklider, 1960; Engelbart, 1962; Kay and Goldberg, 1977). The computer can not only perform computations for us, but also help us make dynamic representations of thoughts and manipulate them easily. When we manipulate external representations, we can "think bigger thoughts", but the advanced computer user should

be able to easily interact with these representations. The system need not be necessarily "easy to learn", in the sense that the methods that make such interactions possible can be learned easily. But acting on external representations should be "easy to do", in the sense that the user's resources are not expended in interactional details, or else she may quickly learn to downgrade her ambitions.

To consider that the world around us helps us think is to accept the view that cognitive processes are not limited to mental activity, and that technology and culture play an important role in the development of human cognition (Steiner, 2010). This view is expressed in theories such as distributed cognition (Hollan et al., 2000), which considers that cognition encompasses not only an individual's mental activities, but that it is also embodied and social: individuals are interacting with their environment, and are shaped by sociocultural processes. Cognition can then occur "in the head" or "in the world", depending on where it is less costly to perform. An example of cognition in the world is the physical actions we perform that are not *pragmatic*—they do not contribute to our advancement towards a goal—but *epistemic* (Kirsh and Maglio, 1994): they help us simplify mental operations. We use epistemic actions to obtain information through the environment if it is faster than obtaining it by mental operations alone (Gray and Fu, 2001; Destefano et al., 2011).

More generally, Kirsh (2013) found seven benefits in using external representations (*externalizing*):

- External representations can be used to reify a thought or a mental structure, to share it with others, and to offload computations to the physical world.
- Because objects can be rearranged, the properties they represent and their relationships can be directly perceived.
- While thoughts could change over time and disappear, things are stable and persistent: the person has enough time and remaining mental resources to build and manage more complex structures than what can fit "in the head" alone.
- Representations can be reformulated in forms that more explicitly evoke their meanings, making the represented concepts easier to grasp.
- The person can use and combine multiple representations to explore different aspects and implications.
- Once represented, a mental structure can be checked for consistency: the fact that one can build a structure by rearranging and reformulating persistent external representations suggests that the structure is viable and consistent.
- Some thoughts are better explored when externalized in their source medium: for example, making sense of musical ideas can be better and more accurate using musical instruments producing actual sounds than by mentally simulating them.

However, these benefits hold only if rearrangement, reformulation, combination, and other operations on external representations are not more difficult to perform externally than internally.

Our ambitions in thinking, our ability to experiment different ideas to solve a problem, and the extent to which we are augmented by the world around us, thus seem to be determined by how these operations are made possible. In his seminal paper, Licklider (1960) made the following comment about his own work activity:

> My choices of what to attempt and what not to attempt were determined to an embarrassingly great extent by considerations of clerical feasibility, not intellectual capability.

This is connected to the "Neo-Whorfian hypothesis" assumed by Engelbart (1962), which he illustrates in the following thought experiment: if the best writing tool ever invented was as difficult to write with as a brick, the speed at which people write and represent thoughts on paper would have been much lower, the form of their written language would certainly be more economical, which would have impacted the language itself; but most importantly, people's limited ability to manipulate representations would have severely discouraged them to "think bigger thoughts".

There are two implications for the design of interactive systems. First, the manipulation of external representations should be facilitated by the system: when doing so, the user should be able to focus on their meaning, not on the actual manipulation. Second, as a prerequisite, a system should give its user enough leeway so that she can convey the proper meanings through the system. The first implication points to *directness* in human-computer interaction, while the second one points to the user's *expressiveness* with the system. In sum, for the desktop (or laptop) to be successful as a "tool for thought", it should support both directness and expressiveness.

The notion of directness appeared in HCI after Shneiderman (1983) observed the direct manipulation interfaces that result from the availability of a bitmapped display and an on-screen pointer. Such interfaces were remarkable because, contrary to language-based ones, objects of interest were continuously displayed before the user's eyes, responding immediately to her physical actions, and providing graphical cues that allowed novice users to comprehend interaction possibilities and the effects of prospective actions. Further research clarified the concept and outlined some properties responsible for the feeling of directness, which are either related to close spatial and temporal relationships between action and effect (Hutchins et al., 1985; Beaudouin-Lafon, 2000), to low cognitive overhead (Hutchins et al., 1985; Bardram and Bertelsen, 1995), or to the absence of perceived intermediaries concerning agency (Hutchins et al., 1985; Hutchins, 1987) or the amount of interaction needed to achieve a given effect (Frohlich, 1993, 1997).

This knowledge facilitates the design of interactive systems with which the user can interact in a direct way, which is relatively easy as long as their complexity stays within the limits of what can be displayed on the screen and of the basic actions that can be performed using the input devices. But the explosion of computing and storage capabilities has led to a commensurate

increase in the complexity of task users would like to perform on their computers (Beaudouin-Lafon, 2004). The corresponding software improvements allow users to express more complex intentions, but the directness of interaction does not scale as easily. For instance, controlling an arrangement of 2D objects at different levels of scale can be done, but requires multiple panning and zooming operations to be interleaved with the user's actions immediately pertaining to the task (Shoemaker and Gutwin, 2007).

Many interaction techniques have been introduced to address such tasks (e.g., Perlin and Fox, 1993; Shoemaker and Gutwin, 2007) and other specific contexts, like direct manipulation across windows (e.g., Dragicevic, 2004; Faure et al., 2009). Alternatively, some works focused on more direct and efficient ways to increase the granularity (e.g., Masui et al., 1995; Appert and Fekete, 2006) and the extent (e.g., Malacria, 2011; Ramos and Balakrishnan, 2003) of specific actions like controlling an on-screen slider, such that they can be performed in a manner more compatible with the normal way of doing so; that is, when everything fits on the display. However, they mostly introduce new graphical objects or actions that must be handled during interaction. In addition, there exist tasks, like 3D manipulation (e.g., Balakrishnan et al., 1997; Perelman et al., 2015) and live musical performance (Hunt and Kirk, 2000), for which standard input devices do not seem appropriate because their control lacks dimensionality—there are too few degrees of freedom simultaneously controllable.

1.2. Thesis statement and goals

In this thesis, we argue that it is possible to develop fundamental ways to increase the expressiveness of desktop and laptop users that are inherently more direct than other solutions by considering the unused possibilities of standard input devices and of the basic and familiar actions that can be performed on them. We investigate this hypothesis in three specific areas—increasing the granularity of pointing and dragging actions, increasing the extent of dragging actions, and increasing the dimensionality of musical expressions—by determining the necessary conditions for its realization, as well as the potential gains in expressiveness.

We specifically restrict the scope of the research to standard input devices for two reasons. First, as we will try to demonstrate, their possibilities are considerably underused, and because they have changed relatively slowly, they can be considered as stable elements to which interaction technique designs can be more tightly coupled. Second, we believe that the portability and self-sufficiency of the laptop is an important asset for a personal medium that assists in doing and thinking, because it can be used everywhere, at any time, without prior setup. The portability requirement of the DynaBook, for example, was certainly not incidental ("Need we add that it be usable in the woods?", Kay, 1972).

To achieve our goals, we first need to understand the implications of directness and expressiveness for the design of interactive systems. From existing accounts of directness, it is unclear how the different properties they have identified are connected, and the notion of expressiveness is rarely

used directly in the HCI literature. Successful completion of this goal will result in a definition of directness, with supporting properties, as well as in a description of strategies for supporting expressiveness in interactive systems.

Concerning the granularity of pointing and dragging actions, the respective roles of the underlying parameters to be controlled, the device, and the user who controls them must be determined, and a way to increase granularity with minimal impact on directness must be devised. This goal will be successful if these roles are theoretically or empirically quantified, and if the proposed method demonstrably leads to an increase of granularity.

Concerning the extent of dragging actions, we need to assess the potential of a widely used family of techniques, *edge-scrolling*[1]. Successful completion of this goal will yield a characterization of the factors influencing its design, a description of current system and user behaviors regarding it, and an empirical assessment of the effectiveness of existing and/or proposed designs.

Concerning the dimensionality of musical expressions, we need to show whether standard desktop or laptop input can be successfully used for the simultaneous control of multi-dimensional musical parameters in real time. This goal will be met when some of the factors governing the mapping of user gestures on standard input to musical output are elucidated, and when prototype examples can demonstrate effective scenarios.

1.3. Research contributions

This research makes six original contributions to HCI:

1. A definition of directness in three dimensions, clearly separating the spatiotemporal, cognitive, and mediational aspects of the concept, and providing several properties to support each of them.

2. The design and implementation of transfer functions compatible with *subpixel interaction*, a fundamental way to increase the granularity of basic pointing and dragging actions by leveraging the sensitivity of input devices and the fine motor capabilities of users, while retaining the normal behavior of the on-screen pointer. Subpixel interaction is also compatible with any existing method that relies on pointer control.

3. An experimental protocol and measure of the *useful resolution* of a user with a particular device, which quantifies the smallest displacement she is able to reliably produce using it. The useful resolution provides an alternative measure of users' fine motor skills with any input device.

4. A conceptual framework of the factors governing the design of edge-scrolling techniques, including a detailed description of the techniques implemented in common desktop software, and a set of guidelines derived from empirical studies.

---

[1]Edge-scrolling allows users to pan a viewport by pointing inside a dedicated area near its edges. This interaction is typically used to reach a distant target while performing selection or drag-and-drop tasks that necessitate scrolling.

PART I: DIRECTNESS AND EXPRESSIVENESS

Chapter 2
Directness

Chapter 3
**Expressiveness
on the desktop**

PART II: INCREASING GRANULARITY

Chapter 4
**Subpixel
interaction**

Chapter 5
Useful resolution

PART III: INCREASING EXTENT

Chapter 6
**Design and use of existing
edge-scrolling designs**

Chapter 7
**Generating and evaluating
edge-scrolling designs**

PART IV: INCREASING DIMENSIONALITY

Chapter 8
Office music

FIGURE 1.1: Structure of the dissertation. Arrows can be interpreted as *"X informs Y"*.

5. The design and evaluation of push-edge and slide-edge scrolling, two novel edge-scrolling techniques operated by "pushing against the viewport edge". They decouple the pointer and the input device when activated, thus ensuring a constant increase of extent in all situations, with performance comparable to the state of the art.
6. The design of three "office music" instruments that demonstrate successful increases of dimensionality for musical expressions. They enable users to "play the laptop" as if it was a polyphonic piano-like instrument, a monophonic wind instrument, or a modern drum kit, while allowing them to quickly alternate between real-time musical control and graphical interaction during live performance.

1.4. Structure of the dissertation

This thesis is divided into four parts (Figure 1.1). The first part is a preliminary discussion of the notions of directness and expressiveness, which informs and motivates our contributions to increasing granularity, extent, and dimensionality, presented in the three ensuing parts. The four parts and their corresponding chapters are outlined in the following.

Part I provides a literature overview of directness and expressiveness, two notions that are central to our work. Chapter 2 discusses the work on directness, as well as related notions, and tries to provide a unified perspective by defining directness in three dimensions: spatiotemporal, cognitive, and mediational. Chapter 3 details the notion of expressiveness. Based on the literature on semiotics and semiotic engineering, it outlines the ways a system can support the user's expressiveness and increase its expressive power, discusses existing strategies to do so on desktop and laptop systems and their impact on directness, and introduces the contributions of the three next parts.

Part II is concerned with the granularity of basic pointing and dragging actions. In Chapter 4, we present the concept of subpixel interaction, which allows users to interact "in-between pixels" and attain a level of granularity suitable for a wide range of tasks by simply controlling the pointer. We show how pointing transfer functions can support subpixel interaction by incorporating three critical factors: device characteristics, human capabilities, and data granularity. We also describe various examples that benefit from it, and determine the limitations of this approach. In Chapter 5, we focus on human capabilities and introduce the useful resolution, a measure of human limits in the production of small unidirectional device displacements. We propose an operational definition of the useful resolution, as well as an associated experimental protocol to measure it. We then conduct a controlled experiment suggesting that the useful resolution is dependent on movement direction, and that our users have a useful resolution between 200 and 400 counts per inch (CPI) when using a high-end 6400 CPI mouse.

Part III focuses on the extent of dragging actions. In Chapter 6, we present the first investigation of the factors governing the design of edge-scrolling. We propose a design space of edge-scrolling techniques highlighting four main design dimensions and two task-related factors. We then use a novel reverse-engineering method to show that 33 existing edge-scrolling implementations present substantially different design approaches. We also investigate the use of edge-scrolling through an online survey, which confirms that edge-scrolling is widely used and links particular perceived usability problems back to the dimensions of the design space. In Chapter 7, we focus on the design of novel edge-scrolling designs and their evaluation. We present push-edge and slide-edge scrolling, two position control techniques whose design is motivated by the pitfalls of existing implementations. A first controlled experiment based on a text selection task using a mouse and a touchpad suggests that these techniques outperform the standard OS X implementation, and that push-edge is best combined with a mouse, while slide-edge is best combined with a touchpad. The results of a second experiment comparing more diverse behaviors examined in the previous chapter show that push-edge performance is comparable to the best designs tested, and highlights several problematic design choices in existing techniques. The chapter also concludes this part by discussing further evaluation criteria and deriving general design guidelines from empirical results.

Part IV examines the remaining issue: the dimensionality of musical expressions. In Chapter 8, we discuss the possibility of real-time musical performance similar to traditional instruments on an unequipped laptop. After outlining the literature relevant to musical expression and the design of digital musical instruments, we discuss several design considerations relative to standard laptop input devices and musical gestures, physical constraints and posture, as well as the properties of computer keyboard layouts for selecting musical tones. We then describe the design and use of three successful "office music" instruments, the Pianotant, the Soufflant and the Tapant, as well as mechanisms to switch between instrumental control and graphical interaction. We conclude by discussing how physical gestures similar to those employed in our designs can be applied in other contexts than musical performance through three examples.

Finally, Chapter 9 concludes the dissertation by discussing the main findings of our research and providing directions for further investigations.

Directness and expressiveness on the desktop

Directness

The term "directness" is mostly associated in human-computer interaction with *direct manipulation*, a set of principles defined by Shneiderman (1983), who analyzed the novel possibilities afforded by graphical bitmapped displays that were observed in many contemporary systems, from display editors to video games. The four principles of direct manipulation are:

Continuous representation Instead of requiring the user to issue a `display` command to the system in order to see the results of a change, the object of interest is represented continuously and in context.

Physical action Instead of using a complex syntax, the user performs physical actions on display objects, such as a moving a pointer, or pressing labeled buttons that remind her of possible actions.

Incrementality and reversibility Instead of requiring long and complex typed sentences, operations are performed rapidly and their effect is immediately visible. The user can quickly repeat them and verify if they are furthering her goals. If not, inverse operations ease recovery from errors.

Spiral approach to learning Instead of learning from manuals, the user can operate with minimal knowledge and learn basic operation through observation of other users. Using system feedback in response to exploratory actions, the user progresses towards mastery.

Shneiderman defined the principles of direct manipulation in opposition to language-based interaction (Frohlich, 1993). He argued that systems designed according to these principles are easier to learn, improve performance, and increase satisfaction by making interaction comprehensible, natural, rapid, and enjoyable (Shneiderman, 1983). Direct manipulation has also frequently been promoted as a way to facilitate the externalization of cognition (van Oostendorp and Walbeehm, 1995). As they were formulated, the principles of direct manipulation seem related to the idea of being able to physically manipulate visible objects, but the benefits may not hold if the designer chooses the wrong graphical representations or operations, uses ambiguous icons or misleading metaphors.

It is unclear, then, whether the benefits of direct manipulation originate from a particular mode of interaction—acting on and perceiving a virtual world, rather than conversing with an virtual partner (Shneiderman, 1997)—or from other qualities that could exist independently of manipulation-based behavior. Subsequent work on direct manipulation has provided much clearer definitions of directness, even though we will see that some of the notions they defined remain obscure.

In this chapter, we try to detail these notions and provide a unified perspective on directness. We will first review various explanations of directness and the associated considerations raised by students of direct manipulation. We will then incorporate frameworks that are not explicitly related to direct manipulation, but can provide a complementary understanding of some aspects of directness. In addition, we will discuss works that focus on the concept of mediation through the lens of activity theory. Finally, based on this review, we will propose a more comprehensive and detailed definition of directness in three dimensions.

2.1. Accounts of directness related to direct manipulation

Since the paper of Shneiderman (1983), various authors have expanded on the origins, the constituents, and the implications of the notion of directness. Their work, which we will review in chronological order, has drawn on a number of theories to uncover the different ideas behind direct manipulation.

2.1.1. LOW SYNTACTIC VARIABILITY AND HIGH-LEVEL SEMANTIC KNOWLEDGE

The claimed benefits of direct manipulation were examined by its coiner (Shneiderman, 1983) using the syntactic/semantic model (Shneiderman and Mayer, 1979), an information processing framework initially proposed to describe programming behavior. The model can describe basic tasks, such as writing or debugging a program, in terms of cognitive structures that are manipulated by cognitive processes. The cognitive structures describe the organization in long-term memory of task-related knowledge, which can be *semantic* or *syntactic*.

Semantic knowledge consists of general, abstract concepts formed and structured hierarchically during learning and problem solving. Higher-level knowledge related to the task domain is decomposed into lower-level concepts closer to the individual commands used in interactive systems. However, semantic knowledge does not depend on the system.

In contrast, syntactic knowledge is tied to the arbitrary representations of a particular system. It is more concrete, and also more volatile in memory. The acquisition of new syntactic information does not involve integration into existing knowledge.

The syntactic/semantic analysis of direct manipulation (Shneiderman, 1983) offers two main hypotheses. First, a direct manipulation system exploits well-developed action and perception skills, consequently there is lower syntactic variability compared to systems using language. And second, operations on graphical objects are directly related to the higher-level knowledge of the task domain, relieving the user from the cognitive burden of dealing with low-level concepts. The first hypothesis pertains to the manipulation-based aspect, but the second is related to the idea of "cognitive directness": a close mapping between the task domain and the commands offered by the system.
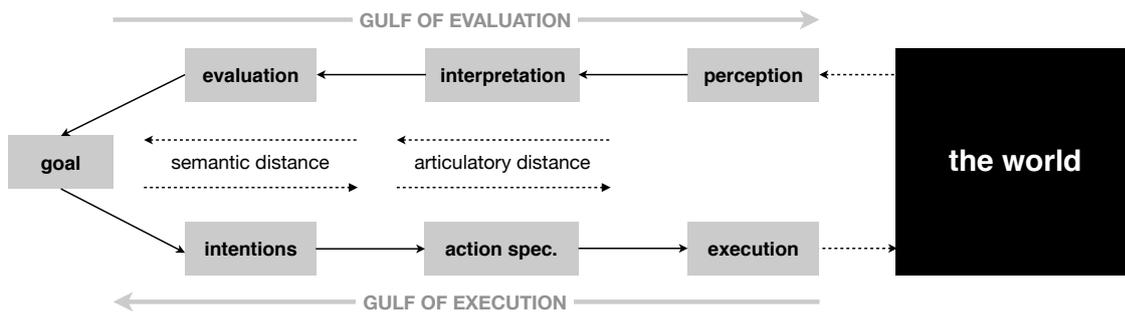
FIGURE 2.1: The stages-of-action model of Norman (1987) outlines the user's cognitive and sensorimotor activity (solid arrows) when interacting with the world (black box) in seven stages (gray boxes). Dashed arrows represent two types of distances in the gulfs of execution and evaluation: semantic and articulatory distance.

2.1.2. COGNITIVE DIRECTNESS

The next discussion of direct manipulation builds upon Norman's stages-of-action model (Norman, 1987). In this model, human-computer interaction is a dynamic process in which the user furthers her psychological representation of the state of a task through action on and perception of the physical state of the system. This process is characterized by cycles of action/execution and perception/evaluation, which can be roughly approximated by seven stages (Figure 2.1):

Goal The user first forms a *goal*; i.e., the psychological state that is meant to be reached through interaction ("*what do I want?*").

Intentions The discrepancy between the current state and the desired goal motivates its translation into several levels of increasingly detailed *intentions*; i.e., decisions to take action in order to advance towards the goal ("*what needs to be done?*").

Action specification The user then selects an appropriate action sequence matching the intentions. She does so by determining a desired system state and how to manipulate the system to attain this state ("*how is it done?*").

Execution The selected sequence of actions must then be carried out physically through the system. It is at the action specification and execution stages that the differences between language-based and manipulation-based action operate (Norman, 1984).

Perception The user perceives the physical state of the system ("*what is to be seen?*").

Interpretation The perceived physical state of the system is interpreted in order to determine its relationship to the user's goal ("*what is the state of the things I am interested in?*").

Evaluation The interpretation of the perceived system state is compared to the goal ("*have I got what I want?*").

This model exposes the separation between the user's goals and the physical state of the system as two unidirectional gulfs, the *gulf of execution* and the *gulf of evaluation*, that the user bridges by developing a mental model of the system from her interpretation of the system's interface. But the interface can also take part in bridging the gulfs by providing adequate representations at

a level of description similar to the user's model of the task, thereby reducing the user's cognitive effort. Hutchins et al. (1985) identify the *distance* remaining to be spanned by the user alone as one of the components of directness: the larger the distance, the more cognitive effort is required to span the gulfs, and the less interaction feels direct.

There are two distances in the gulfs of execution and evaluation (Hutchins et al., 1985): the *semantic distance* between the user's goals and the concepts provided by the interface, and the *articulatory distance* between these concepts and their physical form. In the gulf of execution, semantic distance is crossed at the intention stage, where the user's goal is translated into intentions ultimately formulated using the interface's concepts. The closer these concepts to the user's thoughts about the task domain, the smaller the semantic distance. Then, the action specification stage spans the articulatory distance, which is determined by the effort required to formulate an action sequence from intentions. In the gulf of evaluation, the articulatory distance is bridged at the interpretation stage, where meaning is derived from the form taken by the system's output. The amount of effort needed afterwards to relate the interpreted meaning to the goal determines the semantic distance in the gulf of evaluation. Note that in all cases, semantic and articulatory distance are defined independently of the mode of interaction. The designer's efforts to reduce semantic and articulatory distances in the gulfs of execution and evaluation contribute to the feeling of *cognitive directness* (Frohlich, 1993), but so does the user's adaptation to the system by learning and skill acquisition.

Note that there are two levels of articulatory distance that Hutchins et al. (1985) do not clearly separate. At the first level, which we will call *referential distance* (Hutchins, 1987), or referential gap between meaning and form, there is a match between the nature of a physical manifestation and its conceptual effects. For example, to move an object, one moves the mouse instead of typing a command. At the second level, which we will rather refer to as *movement compatibility* (Sanders and McCormick, 1992), there is a match between the characteristics of a physical manifestation and the characteristics of its conceptual effects. For example, having to move the mouse to the left in order to move an on-screen object to the right is a case of poor movement compatibility, but referential distance is still low.

2.1.3. ENGAGEMENT

In the view of Hutchins et al. (1985), the other important aspect of directness is *engagement*, the degree to which the user feels she is directly engaged and involved in interaction. The feeling of engagement is related to the idea of *first-personness*, which is characterized, according to Laurel (1987), by continuous real-time input (*interactive frequency*), a wide range of choices (*interactive range*), and opportunity for significant impact on the result of interaction (*interactive significance*). The notions of engagement and first-personness pervade Shneiderman's formulation of the principles of direct manipulation (continuous representation, physical action, and incrementality and reversibility).

The discussion of engagement involves two major "mode of interaction metaphors" (Hutchins, 1987) that both assume the existence of a world where actions take place. With the conversation metaphor, the user "talks about the world"; i.e., she delegates action on and perception of this world to the interface, which acts as an intermediary between them. On the contrary, with the model world metaphor, the user "interacts with the world"; that is, this world is explicitly represented and responds in accord with the user's actions.

In addition, Hutchins (1987) contends that there is engagement when the objects stand for themselves, when there is no longer an arbitrary relationship between meaning and form. This is one of the characteristics, but also one of the weakest aspects of direct manipulation, because it banishes any abstract reference that enables much of the power of language (Buxton, 1993). Hutchins (1987) pushes this idea further by arguing that the model world metaphor eliminates referential distance[1], and that a "collaborative manipulation metaphor" in which language and action coexist and supplement each other might be more beneficial. The complementarity between conversation and manipulation has been subsequently investigated in mixed-mode or multimodal interfaces (Cohen et al., 1989; Grasso et al., 1998).

In sum, the feeling of engagement involves many notions: immediacy, incrementality, referential distance (the match between the nature of a physical manifestation and its conceptual effects), "mode of interaction metaphor" (action or conversation), and the perceived relation between the user, the computer, and the world.

2.1.4. GRACEFULNESS

The study of conversational and mixed-mode interaction has exposed another aspect of directness that had been neglected in previous discussions. It is apparent during human communication in the "rapport" (Frohlich, 1993) that develops between interacting parties when they share enough *common ground* (Clark and Brennan, 1991) to communicate well and efficiently. Each party can infer the meaning of an utterance thanks to a continuously accumulated common ground, and relies on the others' ability to do the same when speaking. Efficiency in human conversation is not just a matter of individual economy of effort, as prescribed in the Gricean maxims (Grice, 1975) of quantity (*"be as informative as required, but not more"*) and manner (*"be as brief and as clear as possible"*). It is rather governed by a principle of *least collaborative effort* (Clark and Wilkes-Gibbs, 1986): what is minimized is the total work of *both* parties to reach mutual understanding, leading to shorter conversations.

By transposing the communication-related ideas of rapport and least collaborative effort to human-computer interaction, Frohlich (1997) defined *gracefulness*, a feeling of smoothness, elegance, and efficiency. Gracefulness amounts to fewer and shorter "turns" between user and

---

[1]We do not agree with Hutchins' argument that referential distance is eliminated in a model world metaphor. For example, *surrogate objects* (Kwon et al., 2011) allow manipulating properties of one or several objects through an archetypal representation. They introduce referential distance but may still agree with the model world metaphor.

computer, thanks to a continuously updated *shared context* (Brennan, 1998), allowing the same effect on the world to be achieved with a minimal amount of interaction.

Gracefulness exists independently of the other dimensions of directness, but is associated with engagement and cognitive directness. First, in a manipulation-based interface, the accumulation of shared context is well supported due to the same aspects of direct manipulation that had been previously linked with engagement: continuous representation of objects on the screen provides up-to-date shared context, incremental user operations and corresponding system feedback give each parties confirmation that previous expressions have been acknowledged, and selections indicate the shared focus of ulterior expressions (Brennan, 1998). And second, gracefulness and semantic directness have conflicting requirements when the shared context is not entirely "in the world" but also "in the head": in this case, reducing the amount of interaction implies more cognitive effort for the user (Frohlich, 1993).

2.2. Related frameworks

The previous works have shown that, behind the principles proposed by Shneiderman (1983), several sorts of distances separate the user from the world she interacts with: semantic and articulatory distances (linked to the cognitive aspects of interaction), the distance placed in the user's action on the objects of interest (engagement), or the number of "twists and turns" that separate a user from the desired effect (gracefulness). Other frameworks have been developed that do not stem from the concept of direct manipulation but give support to some of the previous findings and give complementary insights into the notion of directness.

2.2.1. DIRECTNESS IN PYGMALION

Before the definition of direct manipulation, Smith (1977) described the design of Pygmalion, a graphical programming system leveraging direct manipulation for reflective thought and learning. Several aspects of this design and principles developed by Smith pertain to directness.

First, in Pygmalion, the display was used as an extension of the programmer's working memory that could assist thinking. *Working memory* refers to a person's ability to remember and manipulate information over a short period of time. The number of items in working memory is limited, but can be somewhat extended by *chunking*—the grouping of related pieces of information into one unit (Miller, 1956). Visual organization, Smith argued, was an effective method of chunking, thus interactive displays had the potential to offload the user's memory.

Second, to successfully externalize thoughts, the chosen representation should be *articulate*; that is, the effort in translating an "internal representation" into an external representation should be minimal. The number of mental transformations to go from the internal to the external representation defines the *translation distance*. According to a minimum translation principle, the most articulate representation for a thought will be the one that implies the less translation distance.

And third, in Pygmalion, the actions on visual representations are incremental, and the user can immediately determine their consequences on the display. Incrementality favors exploration and creative uses of the representations.

2.2.2. SKILLS, RULES, KNOWLEDGE

Vicente and Rasmussen (1990) proposed a prescriptive design framework, *ecological interface design*, in response to the inadequacy of contemporary design methods for process control systems, such as those used in nuclear plants. They argued that the traditional "single-sensor-single-indicator" approach, where every sensed property of a complex system could be directly perceived and operated on, left the operator in charge of inferring the high-level state of the system, which ought to be provided by the interface because it is more relevant than low-level details when problems arise. The traditional approach was unable to support high-level problem solving and to leverage the action and perception capabilities of the operator, leaving much of the complexity of the task domain in the hands and minds of the operator.

The ecological interface design framework aims at more direct interaction between the operator and the process at both levels. To fulfill this goal, it provides, on the one hand, an *abstraction hierarchy*, a stratified representation of the contents and structure of the task domain that structures interface concepts so as to guide the operator in problem solving behavior (Vicente and Rasmussen, 1992). On the other hand, it uses the *SRK taxonomy*, a model of human behavior that can be used to represent interface concepts and design interaction in a way that matches human capabilities. The SRK taxonomy posits three levels of human information processing behavior (Rasmussen, 1983):

Skill-based behavior This is the lowest level, describing mostly unconscious and automated sensorimotor behavior, continuously guided by the variation of spatiotemporal variables perceived from the environment. At the skill-based level, control is parallel and effortless.

Rule-based behavior Throughout her life, the individual constitutes a collection of sequences of automated actions in rules that can be used based on the current goal or task. Rule-based behavior is concerned with the selection of units of practical knowledge relevant to certain features of the situation at hand.

Knowledge-based behavior Here, the individual forms an explicit goal from her aims and her analysis of the environment, and tests several plans to fulfill the goal, either by action or by processing symbolic information. At the knowledge-based level, control is sequential, effortful, and bounded by short-term memory.

Ecological interface design states that users must be able to use either level depending on the demands of the task. Therefore, the design does not enforce a specific level, especially one higher than needed, and it is structured so that skill-based, rule-based, and knowledge-based behavior are successfully supported (Vicente and Rasmussen, 1992).

To support skill-based behavior, there should be a high degree of *coupling* between perception and action, one of the characteristic features of direct manipulation. The structure of the display should also correspond to the hierarchical structure of movement in space and time, so as to

facilitate the formation of visual chunks and the corresponding phrasing of user input (Buxton, 1986).

To support rule-based behavior, the display must provide cues that help the selection of a relevant action. The mapping between these cues and the constraints or laws inherent in the task domain should be consistent, so that the user can depend on rules in a reliable fashion and use a less effortful rule-based behavior in most situations where knowledge-based behavior might have been required with traditional approaches.

Finally, to support knowledge-based behavior that will nevertheless be required in unexpected situations, the task domain must be represented as an abstraction hierarchy that can be used as an external mental model, helping problem solving and thought experiments by reducing cognitive load. The simultaneous availability of multiple levels of abstraction makes cognitive directness more likely.

In summary, ecological interface design advocates integration and flexibility in the levels of control and representation to encourage tightly coupled, skill-based behavior, when possible, while giving appropriate support for knowledge-based behavior.

2.2.3. FLOW-SUPPORTING INTERFACES

Feelings of mastery, confidence, reduced anxiety, and enjoyment are among the reported effects of direct manipulation (Shneiderman, 1983) that evoke a highly positive and rewarding experience known as *flow*. Csikszentmihalyi (1990) summarizes the following principal components of flow:

Balance between skills and challenges The activity one is engaged in provides opportunities for action towards a meaningful goal, but requires appropriate capabilities in order to have a chance of completing it. When the person's skills do not match the challenges of the situation, boredom or anxiety ensue.

Deep involvement and uninterrupted concentration Attention is entirely directed towards the object of the activity, such that the person becomes unaware of the details of the specific actions taken and of the irrelevant peculiarities of the situation. External needs disappear from awareness and the activity becomes *autotelic* (Csikszentmihalyi, 1975), performed for its own sake.

Sense of control The person does not worry about failing and is confident in her ability to exercise control over an uncertain situation in order to complete her goals. This feeling of control is present even in situations involving pure luck, such as gambling.

Clear goals and feedback The person has a clearly expressed goal for the activity, or a strong sense of the end result, and is sustained in her activity by receiving regular feedback relevant to her progress in attaining of the desired outcome.

Loss of self-consciousness and transformation of time As the whole focus is on the activity, awareness of the self as separate from the environment disappears. The usual perception of time is distorted to fit the pace of the activity.

Bederson (2004) investigated how interactive systems can support these components of flow in order to get out of the user's way and support her in focusing on the object of interest. To balance skills and challenges, the interface should not only be usable by novices but also provide experts with extremely efficient methods that require minimal cognitive effort and reliance on feedback for activation. In other words, the user should be able to act in an automatic fashion after a learning period. To support sustained focus and avoid interruptions, the interface should issue notifications at the right time and at the right place (for example, in the user's visual periphery). It should also be adaptable so that, prior to the activity, the user decides what are the most relevant display items. And to relieve the load on the user's short-term memory, it should avoid brutal changes of context by using animation and displaying information in context. To maintain a sense of control, the interface should favor adaptability to adaptivity, which is potentially disruptive if users cannot anticipate the changes. It should also support backtracking and undoing actions, which is easier when immediate feedback is provided. Finally, it should provide means for users to remind themselves of their goals and their progress towards them.

2.3. Directness in mediated action

Thus far, we have assumed, without really detailing it, that the computer plays two simultaneous roles in human-computer interaction. The first role is that of the model world: in many cases, part of the environment we want to interact with is a dynamic representation that exists in the machine itself. The second role is that of a mediator: the user interacts with the world *through* the user interface (Bødker, 1987), which can make them closer or farther apart in several respects. The following body of works leans towards a clear separation of both roles of the computer and provide new insights for understanding directness.

2.3.1. ACTIVITY THEORY

A useful framework to comprehend mediation in human-computer interaction is *activity theory*[2], a framework originating from Russian cultural-historical psychology. The basic component of activity theory is the *activity*, the interaction of a subject towards an object, not necessarily physical, to meet some of her needs (Kaptelinin and Nardi, 2006). Through the activity, both the subject and the object are transformed. However, the subject rarely interacts directly with the object: activity is usually mediated by *artifacts* that augment our natural capabilities. Artifacts can be external, existing as physical objects, or internal, like language, and they are used to affect things around us or to affect ourselves as well as other human beings. They carry the shared experience of a group and allow us to appropriate it through activity. Activities are

---

[2]Note that we only discuss the individual aspects of this framework, thus we leave aside the social considerations that are important for activity theory but not directly relevant to our topic.

conducted through a sequence of *actions* that can be hierarchically structured. Each action is directed towards a conscious goal, that can be different from the motivating object of the activity, especially in a socially distributed activity. An action, in turn, is conducted through a sequence of *operations* that are performed unconsciously to adjust the action to the situation at hand.

The three levels of the activity correspond roughly to the questions "why?" (*motivational aspects*), "what?" (*intentional aspects*), and "how?" (*operational aspects*), however these levels are constantly changing. An action can be turned into an operation through repetition and learning, a process called *operationalization*[3], allowing the subject to perform it unconsciously. By means of operationalization, the subject constructs a *repertoire* (Bødker, 1987) from which an operation is performed when the situation allows it. However, in some cases the situation may not allow to perform a specific operation, or the operation does not produce the desired result, and the operation becomes an action through *conceptualization*. In such cases of *breakdown* the artifact shifts from being *ready-to-hand* to *present-at-hand* (Winograd and Flores, 1986): it emerges to our consciousness, allowing us to reflect on the situation and to come up with a more appropriate procedure. Hence, while we usually act on the object of the activity through artifacts, sometimes our actions are directed towards the artifact itself.

Bødker (1987) separates three aspects of an interactive system towards which operations can be directed. *Subject-object-directed aspects* concern the objects of the model world, those that we manipulate through the interface. They support operations directed at an object as well as the shift of focus between different objects, which is influenced, for example, by modes and the possibility of undoing mistakes. *Handling aspects* concern the interface as an artifact, a means through which the subject interacts with her object. They support operations on the interface itself as well as the shift of focus between its parts. Finally, *physical aspects* concern and support operations towards the material features of the computer and the input and output devices involved, their physical shape, and how they are manipulated.

Breakdowns can occur that shift the attention of the subject towards any of these aspects. For example, imagine a user of a word processor starting to type a paper on a public computer. After a while she notices that she frequently misses some letters: supported by the subject-object-directed aspects of the interface, she shifts her focus from the current paragraph to individual letters. To diagnose the problem, she may first turn towards the word processor (handling aspects) to verify that the document window is focused. As the problem does not seem to originate from the interface, she then turns towards the keyboard (physical aspects), and notices that its unusual physical layout is the cause of her typing errors. Together, all three aspects support the change of focus from the object to the interface when a problem occurs, as well as shifts back to the object.

---

[3] *Operationalization* and *conceptualization* must not be confounded with the similar pair of terms in the context of experimental design (Hornbæk, 2013), as used in Section 5.2.

2.3.2. THE INSTRUMENTAL APPROACH

To understand the progressive integration of artifacts into human activity, Rabardel (1995) proposes an approach around the notion of *instrumental genesis*: through activity, the artifact becomes an *instrument* to its user. The instrument is not only the artifact, but also a set of general psychological patterns of behavior, called *utilization schemas*, into which previous experiences with the artifact are structured. These experiences include handling and managing the artifact itself (*secondary-task-related activities*) and activities for which the artifact is a means (*primary-task-related activities*), leading to two levels of utilization schemas, respectively *usage schemas* and *instrumented-action schemas*. However the level of a particular utilization schema solely depends on how it manifests itself in the activity. Utilization schemas help understand the situation (*epistemic functions*), they help transforming the world (*pragmatic functions*), and they orient the activity (*heuristic functions*). Two processes govern the development of utilization schemas (Béguin and Rabardel, 2000): they are applied to several kinds of artifacts (*assimilation*), and they change when the situation changes (*accommodation*), leading to more diverse uses. Utilization schemas and artifacts are relatively independent from each other, thus the skills developed in another context can be taken advantage of when designing a new artifact.

However, the development of utilization schemas only describes the *instrumentation* side of instrumental genesis. Artifacts are constantly used beyond their intended purpose, based on the adequacy of their properties to the situation at hand: for example, when trying to hit a nail in the absence of the proverbial hammer, we instead use a wrench whose weight can fulfill the hammer's function. Through this process of *instrumentalization*, we give new functions to some of the properties of the artifact. Depending on the transiency of the association and amount of physical transformation of the artifact, instrumentalization occurs at different levels (Rabardel, 1995): the association can be temporary, relative to specific circumstances and involving no transformation (*local instrumentalization*), or it can be lasting and more general (*permanent instrumentalization*), or it can be permanent and involve modifications in the physical structure of the artifact (*structural instrumentalization*).

2.3.3. TRANSPARENT INTERACTION

For Bardram and Bertelsen (1995), interaction is transparent when most of the work is handled through known operations, with few shifts of attention towards the interface. Consequently, the interface must aid proper operationalization of actions into operations. Proper operationalization leading to transparent interaction depends on the level of generality, abbreviation, and mastery of an action. To attain a sufficient level of generality, the user must have applied the action in enough situations in order to fully comprehend its scope and its invariants. To attain a sufficient level of abbreviation, she must be able to skip some operations, such as frequent explicit verifications of the outcome, that become unnecessary when the expected result of the action can be inferred. And to attain a sufficient level of mastery, she must have practiced the action enough so that she can apply it with ease and speed to a new context.

However, computer activities are seldom oriented towards learning the interface, but rather towards getting things done: *active users* normally do not focus on generalizing and abbreviating their actions and tend to stick with the inefficient methods they master even if they are aware of better ones (Carroll and Rosson, 1987). The user interface, then, should do the work, and support the formation of actions amenable to proper operationalization, by conveying the generality of possible actions and encouraging brief ones that can be mastered. Because learning occurs ultimately through daily use, this kind of support should not be provided solely in manuals but rather into the interface itself, so as not to deflect the user's focus, by showing, based on the user's initial abilities and knowledge, what can be achieved through mastery of such general and brief actions (Bardram and Bertelsen, 1995).

2.3.4. OPERATIVE TRANSPARENCY

But transparency also pertains to the way the user can cope with breakdowns when they occur (Winograd and Flores, 1986), so that her focus can quickly shift back to the object of the activity. In this case, the transparency of the artifact itself is important. Rabardel (1995) identifies two opposing approaches to artifact transparency in the literature: the black box and the glass box metaphors. In the *black box metaphor*, the artifact is transparent when it is invisible to the user; that is, when its structure does not perceivably impede or transform the subject-object relationship. Its internal workings are hidden, and only the *virtuality* (Nelson, 1980), what seems to be there, is important.

On the contrary, in the *glass box metaphor*, the artifact is transparent when it is visible, its mechanisms can be understood, and it is accountable to some extent for its behavior (notably, behavior can be observed and reported, Dourish, 2001), so that the user avoids making misleading inferences about it. Dourish (2001) proposes an example of the problems of a black box approach: in the desktop metaphor, local and remote directories can be used in the same way, and their technical differences are hidden behind the "folder" abstraction. Nevertheless, these differences still have practical effects, for example when deciding which folder to compile a software project, because access time is much higher on the remote folder, but the interface does not give enough clues for the user to be able to understand it easily. Glass-box transparency, however, can be useless, or even problematic, if the visible structure of the artifact does not promote a representation compatible with the object of the activity, which is likely, for example, in "single-sensor-single-indicator" systems (Vicente and Rasmussen, 1990, fig. 1).

These examples show that extreme positions in either kind of transparency should then be avoided, and that an useful conceptualization of artifact transparency must be related to the needs, goals, and capabilities of the user. Rabardel (1995) defines *operative transparency* as the way the properties of the artifact that are relevant for user action are made accessible. The criteria for operative transparency then depend on the goals of the activity: while ease of use and reliability will be relevant in a work context, in a learning context one will rather focus on ways to understand the inner workings of the artifact and favor, for example, constrained actions.

In all cases, the properties of the artifact involved in operative transparency belong to several categories. First, they can be related to the aspects of the structure and behavior of the artifact itself that are relevant for action (*material causality*): for example, the track of a scrollbar reveals a movement constraint on the handle, which cannot be moved freely in two dimensions. Second, it can be related to the way the artifact mediates object-directed action (*instrumented-action causality*): for example, the orientation of magnetic guidelines (Beaudouin-Lafon, 2004) reveals how the graphical objects will be aligned when attached. Third, it can be related to the nature of the values inherent in the artifact and how it estimates the situation (*axiological dimensions*): for example, a computer vision tracking system can display which image features are relevant to its computation. And fourth, it can be related to the autonomous behavior and proper goals of the artifact (*teleonomic dimensions*).

2.3.5. INSTRUMENTAL INTERACTION

Aside from transparency, considering the separate role of the computer as a mediator of human action can lead to other interesting properties. Some are developed in *instrumental interaction* (Beaudouin-Lafon, 1997, 2000), an interaction model that provides a clear terminology to separate the respective contributions of the computer-as-artifact and the computer-as-model-world (Figure 2.2).

In this model, an interface object is either a *domain object*, which likely represents the usual object of the activity, or an *interaction instrument*[4], an artifact mediating the interaction between the user and domain objects, and possibly other instruments. The interaction instrument has a physical part, which is an input device controlled by the user, and a logical part. The logical part of the interaction instrument responds to the user's *action* by issuing a *reaction*, but it also translates the action into a *command* for the domain object. Then, the object's *response* may be transformed by the interaction instrument as *feedback*.

On the basis of this terminology, instrumental interaction provides three properties of interaction instruments (Figure 2.2), allowing designers to compare and evaluate them with respect to their specific situation.

First, the *degree of indirection* concerns the spatial and temporal distances between the interaction instrument and the domain object. An interaction instrument establishes a *spatial offset* between its logical part and the domain object, and a *temporal offset* between the user's action and the domain object's response. A large spatial offset might deflect the user's attention away from the object of the activity towards the interaction instrument, but not always, and a large temporal offset might break the sense of causality between action and response.

Second, the *degree of compatibility* concerns the physical similarity between the user action on the interaction instrument and the response of the domain object, and integrates both levels of

---

[4]To disambiguate the different uses of the term *instrument* in Beaudouin-Lafon's instrumental interaction and in Rabardel's instrumental approach, we will only use *interaction instrument* to refer to the former.
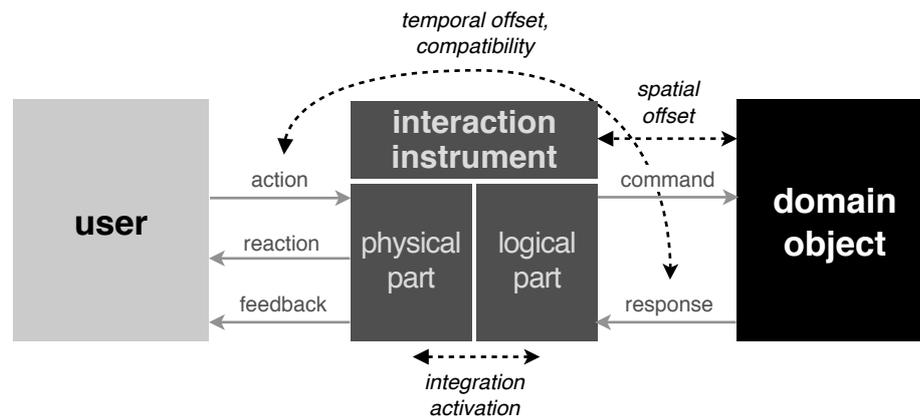
FIGURE 2.2: In the model of instrumental interaction (Beaudouin-Lafon, 2000), the user's actions towards domain objects are mediated by interaction instruments. Gray arrows depict the flow of events between the user, the interaction instrument, and the domain object. Black dashed arrows describe properties of interaction instruments.

referential distance (Hutchins, 1987) and movement compatibility (Sanders and McCormick, 1992).

Finally, the *degree of integration* concerns the relationship between the degrees of freedom (DOF) of the physical part and those of the logical part of the interaction instrument. It is determined as the ratio of the number of DOFs used by the logical part over the number of DOFs provided by the physical part. A ratio smaller than 1 indicates that not all the DOFs of the input device are used, but all the DOFs of the logical part are controlled simultaneously (integral control). A ratio larger than 1, on the contrary, indicates that several successive physical actions might be necessary (separable control). The degree of integration should be determined with regard to the perceptual structure of the task; that is, whether the task-relevant attributes of the manipulated object form a perceptual whole (Jacob et al., 1994). It is related to the gracefulness aspects of directness, in the sense that a proper degree of integration minimizes the amount of interaction.

Another important concern is *activation*, the dynamic association of an input device with the logical part of an interaction instrument: only when it is activated can the user control it. While the degree of integration deals with the mapping between physical and logical part *within* an interaction instrument, activation deals with the same kind of mapping *between* interaction instruments. It is *spatial* when it requires that the on-screen location indicated by the physical part coincides with that of a logical part, and it is *temporal* when it is effective in a modal fashion; that is, from the moment a specific action has been issued until another interaction instrument is activated for the same physical part.

The allocation of the available input devices to interaction instruments can follow different strategies. In an extreme *space-multiplexing* strategy (Buxton, 1987), like that used by graspable interfaces (Fitzmaurice, 1996), each device is permanently associated with an interaction instrument, and activation never occurs. On the contrary, an extreme *time-multiplexing* strategy is

Spatiotemporal directness	Cognitive directness	Mediational directness
Spatial proximity	Conceptual scaffolding	Coagency
Collocation *(user-artifact)*	Semantic distance	
Spatial offset *(artifact-object)*	Referential distance	Turn-taking
	Operative transparency	Turn granularity
Temporal proximity		Integration
Latency *(user-artifact)*	Operational scaffolding	Activation cost
Temporal offset *(artifact-object)*	Skill reuse	Sequentiality
	Predictability	
Compatibility	Skill/challenge balance	

FIGURE 2.3: Three dimensions of directness and properties of interactive systems that contribute to them.

followed when only a two-button mouse is available for graphical interaction: different interaction instruments are activated and associated to the same mouse along the way, but only one can be used at once. These issues are important because activation bears costs: spatial activation requires display space and takes time to point at the interaction instrument, temporal activation requires an explicit action, and both can divert the user from the object of her activity. However, these activation costs must be carefully balanced with other considerations, such as the number of available input devices or their acquisition cost (Douglas and Mithal, 1994).

2.4. Summary of directness in three dimensions

Throughout this chapter, we have discussed various aspects of human-computer interaction that contribute to the feeling of directness. We will now summarize this discussion by providing a definition of directness in three dimensions and how they can be supported in an interactive system.

We define *directness* as a subjective quality of computer-mediated interaction that pertains to the distance established by the system between the user's actions and her goals. From the previous discussion we consider three dimensions of directness (Figure 2.3). First, *spatiotemporal directness* is concerned with the closeness in space and time between the user's actions and their effects. Second, *cognitive directness* is concerned with the mental distance, or the amount of mental operations to be performed, between the user's expectations and the system's behavior. And third, *mediational directness* is concerned with the extent to which an agent, object, or process is perceived to intervene between the user and the environment. In the following, we will detail these three dimensions and, based on the work discussed earlier (Figure 2.4), describe properties of interactive systems that can support them.

2.4.1. SPATIOTEMPORAL DIRECTNESS

Interaction is spatiotemporally direct when action and effect occur next to each other and have similar dynamics in space and time. Spatiotemporal directness is mainly related to the incremental, visual, and feedback-driven aspects of direct manipulation. We consider three main properties that contribute to spatiotemporal directness: spatial proximity, temporal proximity,

	spatiotemporal directness			cognitive directness		mediational directness	
	spatial proximity	temporal proximity	compatibility	conceptual scaffolding	operational scaffolding	coagency	turn-taking
low syntactic variability (Shneiderman, 1979, 1983)			✓				
high-level semantic knowledge (Shneiderman, 1979, 1983)				✓			
semantic distance (Hutchins & al., 1985)				✓			
articulatory distance (Hutchins & al., 1985)			✓	✓			
engagement (Hutchins & al., 1985)		✓		✓		✓	✓
gracefulness (Frohlich, 1993, 1997)						✓	✓
minimum translation principle (Smith, 1977)				✓			
EID, SRK (Vicente & Rasmussen, 1990, 1992)		✓		✓	✓		
flow-supporting interfaces (Bederson, 2004)		✓		✓	✓		
transparent interaction (Bardram & Bertelsen, 1995)					✓		
operative transparency (Rabardel, 1995)				✓	✓		
degree of indirection (Beaudouin-Lafon, 2000)	✓	✓					
degree of compatibility (Beaudouin-Lafon, 2000)			✓	✓			
degree of integration (Beaudouin-Lafon, 2000)							✓
interaction instrument activation (Beaudouin-Lafon, 2000)							✓

FIGURE 2.4: Concepts related to directness and their position in our framework.

and compatibility. Each property can be considered from two distinct points of view: in Rabardel's terminology (Section 2.3.2), those would be that of secondary-task-related activities and that of primary-task-related activities. The first point of view concerns the relationship between the user's physical action and the interface considered as an artifact (user-artifact), while the second point of view concerns the relationship between the user's mediated action and the object of her activity (artifact-object).

SPATIAL PROXIMITY

Spatial proximity on the user-artifact side corresponds to the *collocation* of physical action and interface feedback, which is essentially the result of using direct input devices. On the desktop, on the contrary, visual feedback is not located in the same space as the kinesthetic reference frame (Balakrishnan and Hinckley, 1999) of the hands. Action-feedback collocation—i.e., "direct input"—is generally touted as more "natural", but more liable to issues such as inaccuracy (Holz and Baudisch, 2010) or occlusion (Vogel and Balakrishnan, 2010; Lee et al., 2012). In addition, performance differences are dependent on factors such as display orientation, task type, and the number of hands involved in input (Sears and Shneiderman, 1991; Forlines et al., 2007; Schmidt et al., 2009). On the artifact-object side, spatial proximity corresponds to spatial offset in the instrumental interaction model.

TEMPORAL PROXIMITY

Similarly, temporal proximity on the user-artifact side corresponds to the *latency* between physical action and interface feedback. As much as 200 ms latency substantially decreases performance (MacKenzie and Ware, 1993), and even a few milliseconds of latency can be perceived by users (Ng et al., 2012). On the artifact-object side, temporal proximity corresponds roughly to *temporal offset* in the instrumental interaction model[5].

COMPATIBILITY

Finally, compatibility concerns the degree to which physical spatio-temporal transformations correspond to object transformations: that the pointer moves to the right when the mouse is moved in the same direction is a case of compatibility. Similarly to spatial and temporal proximity, a more precise distinction can be made between user-artifact compatibility and artifact-object compatibility.

Figure 2.5 shows an example of the different properties contributing to spatiotemporal directness on the artifact-object side. Four different interfaces can be used to change line spacing in a text editor, with the one in Figure 2.5a being the most spatiotemporally direct. The interface in Figure 2.5b has less compatibility than the one in Figure 2.5a because the desired transformation (increasing the vertical space between lines) on the object of interest is achieved by dragging the

---

[5]More precisely, the temporal offset we consider here only resides between the logical part of the interaction instrument and the domain object, as opposed to instrumental interaction.
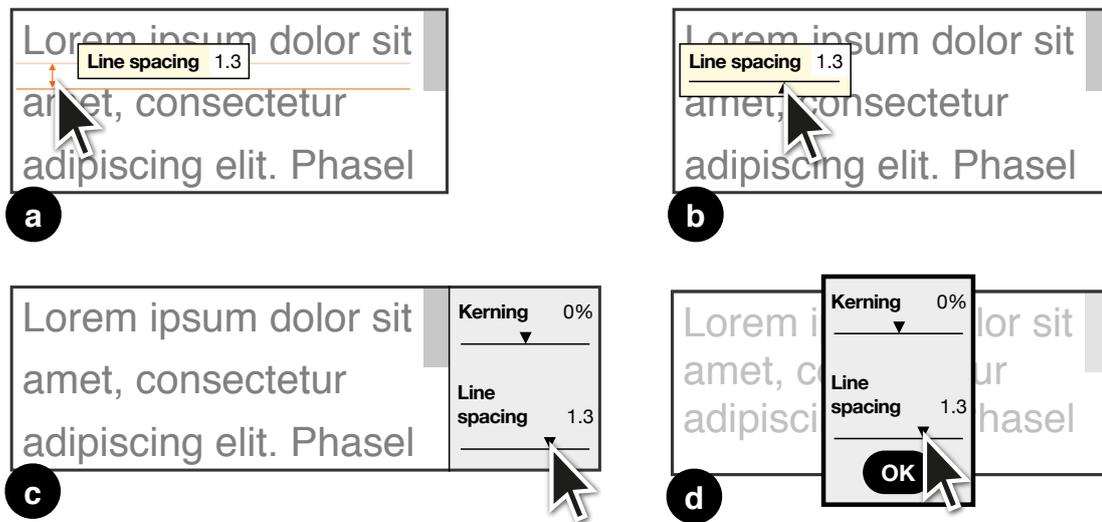
FIGURE 2.5: Decreasing degrees of spatiotemporal directness on the artifact-object side with different text editor interfaces to change line spacing. Line spacing is changed by: (a) vertically dragging the vertical space between lines of text; (b) horizontally dragging a slider (less conformance); (c) dragging a slider in a sidebar (more spatial offset); and (d) doing so with a modal dialog box (more temporal offset).

slider horizontally. The interface in Figure 2.5c has more spatial offset than both, because the interaction instrument is located in a sidebar remote from the text. And the interface in Figure 2.5d has more temporal offset than the others, because line spacing is changed through a modal dialog box that must be closed before the effect can be seen on the text.

2.4.2. COGNITIVE DIRECTNESS

Interaction is cognitively direct when few mental work is needed to perform an action that leads to the desired effects. Cognitive directness is related to the aspects of direct manipulation highlighted by Smith (1977) and Hutchins et al. (1985), as well as the support of flow (Section 2.2.3) and operationalization (Section 2.3.3). To describe the kind of properties involved in cognitive directness, we will use Norman's metaphor of the gulfs of execution and evaluation (Section 2.1.2). In the stages-of-action model, crossing these gulfs involves performing several mental operations. Both the system and the user can build bridges to facilitate these operations: the system does so by providing an interface with short semantic and referential distances, and the user does so by learning and operationalization. While the user's part of the bridge was neglected by Hutchins et al. (1985), it is a first-class concept in other models like activity theory, and Bardram and Bertelsen (1995) have shown how a system can support its construction. We will thus consider that both parts of the bridges enable cognitive directness, and that the system can support the construction of both by providing *conceptual* and *operational scaffolding*.

CONCEPTUAL SCAFFOLDING

Conceptual scaffolding is the set of means provided by the system to support high-level conscious action and breakdowns. In other words, conceptual scaffolding mostly supports knowledge-based and rule-based behavior (Section 2.2.2). We consider three properties that contribute

to conceptual scaffolding: semantic distance, referential distance (Section 2.1.2), and operative transparency (Section 2.3.4).

Semantic distance is large when the levels of description provided by the interface do not match those with which the user thinks about her task: for example, when a user reorganizes the structure of a document but headings and sections are not part of the language of the interface, which only provides low-level formatting commands, it is up to the user to maintain these structures.

Referential distance (Hutchins, 1987; Hutchins et al., 1985) is exploited when the nature of actions and their conceptual effects on the environment match in a non-arbitrary fashion. This is the case when input device movements are used to specify motion in 2D space, or when rhythmic patterns are produced (for example by clicking the mouse) to express the timing of events. Note, again, that referential distance is only concerned with whether the nature of the expression mimics the nature of what it means[6], and not with the spatio-temporal mapping between action and effect, which is rather the concern of the *compatibility* property.

Finally, *operative transparency* is related to how the system reveals its functioning and how it can be used. It includes the criteria of Rabardel discussed in Section 2.3.4, but also the considerations related to the perception of affordances, feedback, and feedforward, that have been much developed in the HCI literature (Gaver, 1991; McGrenere and Ho, 2000; Vermeulen et al., 2013).

OPERATIONAL SCAFFOLDING

On the other hand, operational scaffolding is the set of means provided by the system to support the operationalization of actions, and to sustain rule-based and skill-based behavior (Section 2.2.2). Mostly based on the recommendations of Bardram and Bertelsen (1995) for transparent interaction, we consider three properties that contribute to operational scaffolding: skill reuse, predictability, and skill adequacy.

First, *skill reuse* is the inclination of an interface to tap into the repertoire of already formed operations, resulting in lower syntactic variability (Section 2.1.1) and facilitating learning and operationalization. For example, many emerging interaction styles rely on everyday skills, such as informal knowledge of physics principles, proprioception, environmental and social skills (Jacob et al., 2008). When using general methods, the user can leverage existing utilization schemas (Section 2.3.2) and develop them through assimilation and accommodation.

Second, *predictability* is the extent to which the user needs to rely on the system's feedback, even after extensive learning, to complete an action. One aspect of expert behavior compared to novices is the elimination of verification actions that check the outcome of a previous action

---

[6]In the Theory of Sign Production (Eco, 1976; de Souza, 1993) one would rather speak of whether the *type of continuum to be shaped* is *homomaterial* or *heteromaterial*.

(Kieras and Polson, 1985; Bovair et al., 1990), but if the results of an action is needed to specify the next action and cannot be inferred, the user's ability to abbreviate her action is limited, which hampers operationalization (Bardram and Bertelsen, 1995).

Third, *skill/challenge balance* refers to the proper match between the physical actions needed to use the interface, and the extent to which the user is able to master them (Sections 2.2.3 and 2.3.3). Approaches like ability-based design (Wobbrock et al., 2011b) provide principles, such as enabling adaptation to users' skills and monitoring performance, that specifically promote skill adequacy.

2.4.3. MEDIATIONAL DIRECTNESS

Interaction is mediationally direct when no intervening party and few operations are needed to perform an action that leads to the desired effects. Therefore, this dimension is "mediational" in two senses: first, in the sense that the world can be inaccessible to the user except through the interpretations of the computer, which acts as an explicit intermediary (Ihde, 1979), and second, in the sense that the interface may simply "get in the way" of the user, thus increasing the amount of work to be done in order to perform an action. The first sense refers to a main property of *coagency*, while the second one refers to another main property we will call *turn-taking*.

COAGENCY

Coagency is the perceived relation between the user, the interface, and the world, which can lean between two extremes (Ihde, 1979): in one case, the interface is an instrument and modifies our experience of the world (*embodied relation*), while in the other case the interface is an intermediary to the world, which is experienced indirectly (*hermeneutical relation*).

To clarify, consider the experience of skimming a long written document during its preparation in a word processor on a laptop computer. We glance through its structure by swiftly scrolling, and we can even feel that there are complex parts when some of the most crowded figures engender discontinuities in the otherwise smooth scrolling movement. While doing all this, most of our experience is focused on the document itself, and we forget the touchpad, the display, and the lacking processor resources through which we experience it. In the language of Ihde (1979), we are in an *embodiment relation*, in which the interface modifies some of our senses and action capabilities but is not the object of attention.

Now, consider that the figure on page 79 suddenly makes scrolling grind to a halt. The word processor is entirely frozen and does not respond to any further input. Unwilling to throw away hours of unsaved work, we then feel compelled to focus on the computer itself and, as advanced computer users, to diagnose the problem using various tools (such as a task manager, a profiler, or a debugger) that can give us a reading into the state of the failing process and hopefully make it recover. Our relation with the profiler is a *hermeneutic* one (Ihde, 1979). We do not experience *through* the computer, but instead we are engaged in experiences *of* the computer, and we are left to the profiler's opaque interpretations on the state of the world.

TURN-TAKING

Turn-taking is mostly related to gracefulness (Section 2.1.4) and refers to the notion that interaction is sometimes artificially constrained in successive and discrete "turns" between the user and the computer, which is typical of command-line based interfaces, dialog boxes, and confirmation messages. Several properties can be considered for turn-taking: *turn granularity*, *integration*, *activation cost*, and *sequentiality*.

Turn granularity is related to the incremental and feedback-driven aspects of direct manipulation that leads to few and small "twists and turns" (Frohlich, 1993). Nielsen (1993) argued that the granularity of turn-taking would keep getting smaller in "next-generation interfaces", leading to single continuous user actions instead of slowly progressing, sequential iterations. In addition, turns are partly determined by response times, which according to Bødker (1987) delimit operations because it forces the user to consciously attend to the application's behavior. Thus, a large amount of turn-taking may hamper operationalization, as well as distract the user from her goal (Miller, 1968). At a higher level, the delimitation of actions is also reinforced by the muscular tension applied by the user throughout an action and released at its end, and the *conceptual connectivity* of the constituents of a command should be matched in the phrasing of the physical action by maintaining *kinesthetic connectivity* (Buxton, 1986).

Turn-taking is also influenced by *integration*, the match between perceptual and control structures (Jacob et al., 1994), and *activation cost*, which have been discussed in Section 2.3.5. There is a proper degree of integration when two parameters meant to be manipulated together, like the horizontal and vertical positions of a graphical object, can effectively be controlled simultaneously by the user. On the other hand, activation cost concerns the amount of effort and time needed to associate an input device with an interaction instrument before using it.

Finally, turn-taking also concerns *sequentiality* and parallelism in a more general way. The CIS (Complexity of Interaction Sequences) model can be used to describe an interface and characterize its propensity to turn-taking using properties such as *order* and *parallelism*, which describe how sequential and parallel action structures are imposed (Appert et al., 2005). Sequentiality in turn-taking can also be reduced by performing actions with both hands, with several degrees of parallelism (Bailly et al., 2005). Two-handed interaction has been much explored in the literature (Leganchuk et al., 1998) and has different properties depending on the respective roles of both hands. In his well-known kinematic chain model, Guiard (1987) proposed three principles for asymmetric bimanual action that have guided various designs (McLoone et al., 2003; Kessous and Arfib, 2003; Kulik et al., 2012): the non-dominant hand precedes the dominant hand in action, and the dominant hand acts in the spatial context set by the non-dominant hand, at a finer spatial and temporal scale. Latulipe (2006) showed that different tasks had various degrees of symmetry, and proposed guidelines to support the symmetric cases.

2.5. Conclusion

In this chapter, we examined the concept of directness by discussing existing explanations, analyses, and related theories. We tried to discriminate the various meanings of the term and integrated them in a framework postulating three fundamentally different types of directness (spatiotemporal, cognitive, mediational), with several properties of interactive systems supporting them.

As Figure 2.4 shows, the three dimensions encapsulate most of the concepts pertaining to directness reviewed in this chapter. While directness remains a vague notion, our framework clarifies some of the existing concepts by separating previously entangled aspects (e.g., articulatory distance split into *referential distance* and *compatibility*). This is particularly noticeable for engagement, which pertained to incrementality (*turn granularity*) and immediacy (*temporal proximity*)—"the essence of direct engagement" in direct manipulation, according to Hutchins et al. (1985)—but also to coagency and referential distance (Hutchins, 1987). Our approach does not provide much explanatory power, but the separation of concerns can be valuable to stimulate design and to differentiate various aspects that are often involved simultaneously.

In the following chapter, we will detail the notion of *expressiveness*, and use the three dimensions of directness to provide motivations for the contributions that will be detailed in the rest of the thesis.

Expressiveness on the desktop

The New Oxford American Dictionary defines *expressive* as "effectively conveying thought or feeling". According to this definition, expressiveness relates to the user's ability to successfully evoke, by means of an interactive system, an intended interpretation, whether "interpretation" originates from the system, the user herself, or another person. The system, like any tool or medium, is not expressive in itself, but it enables human expression in various ways.

As the above definition suggests, the notion of expressiveness is indissociable from human communication and meaning-making. Research on these phenomena provides a rich theoretical basis for our forthcoming discussion of expressiveness. However, note that our focus is not interaction with anthropomorphic computers, but rather interaction with computers as artefacts. Despite the different skills involved in tool use and human communication, both perspectives are not incompatible, and we will see that the *computer-as-tool* paradigm (Beaudouin-Lafon, 2004) can benefit from some of the notions originating from communication.

In this chapter, we try to determine how a system can support the user's expressiveness, and how it can do so both within the limits imposed by standard desktop and laptop equipment and in a way that does not impede directness. We will first examine the human aspects of expressiveness, as related to the phenomena of communication and interpretation. We will then focus on the system side of expressiveness; that is, the means an interactive system provides to its users to communicate their intentions both to and through the system. Finally, we will focus specifically on desktop and laptop systems, determine various strategies to influence the user's expressiveness with them, and set the stage for the contributions of this thesis.

3.1. Expressiveness

We will start by detailing the notion of gesture and will show that the issue of expression can be approached using two perspectives: reciprocal action and communication. Then, we will detail the multiple types of communication that occur in human-computer interaction: the user "communicates" with the system, but also with herself or with others through the system. The first type is highly constrained due to the system's limited "interpretation" possibilities. In contrast, the user's interpretations of the system and of the effects of her actions are constantly evolving, progressively aligning with the system designer's intended uses through practice, but also transcending them by attributing novel uses to the system. Finally, we will conclude this discussion of the human side of expressiveness by showing how a system can support users' expressive intents.

3.1.1. EXPRESSION AND GESTURE

Human expression, at least when interacting with the kind of systems we focus on in this research, occurs primarily through the gestural channel. Cadoz (1994) attributes three main functions to human gestures. First of all, gestures have a *semiotic function*: through gestures we convey information to our environment. This function appears in gestures that complement speech, such as deictics that point to an object that is referred to in discourse, or beats that emphasize a particular sentence (McNeill, 1992). It also appears in sign languages, musical conducting, and the use of hand gesture interfaces (Mulder, 1996). But gestures are often *instrumental*; that is, they are often performed towards a physical object.

In this situation, the two other functions play an important role. Through gestures, not only we convey information, but we also obtain information from our environment: this is the *epistemic function*. Information is conveyed by kinesthetic, proprioceptive, and tactile perception, and more generally through the integration of all senses. However, what is exchanged through object-directed gestures is not merely information, to which meaning can be attributed, but also energy: this is the *ergotic function* of gestures. Through gestures we transform our environment, and the environment reacts and resists.

The dimensions of the instrumental gesture can be further detailed with the notions of *selection*, *excitation*, and *modulation* (Cadoz, 1988), which transpose the respective ideas of noun, verb, and adjective from the linguistic to the gestural domain (Cadoz, 1994). A selection gesture establishes a link between the person and one or more independent parts of the object to prepare for a future action. Piano fingering is a classic example of selection gesture where the hand becomes mechanically coupled with the keys, but we will consider that selection gestures also extend to the acquisition of the key of a computer keyboard, or to the acquisition of a button in a graphical user interface. An excitation gesture is one through which energy is transmitted to the object, be it in a continuous fashion or not. Finally, a modulation gesture applies continuous or discrete variations in the properties of the object without directly contributing energy. All three aspects can be found in some gestures: for instance, in a single gesture, one could select one key of a multi-touch enabled piano keyboard, hit it in order to produce a string plucking sound, and modulate its brilliancy by oscillating the finger around the impact point (McPherson and Kim, 2011).

Under its three functions, the gestural channel unites the seemingly different perspectives of manipulation and communication: expressing oneself is a matter of exchanging energy as well as information. However, as we will see, communication is more than information transmission, because it involves interpretation.

3.1.2. INFORMATION TRANSMISSION, HUMAN COMMUNICATION, AND SIGNS

In the well-known model of Shannon (1948), communication occurs through five parts. A *source* produces a message, which is converted into a signal by a *transmitter* before passing through a medium, the *channel*. At the end of the channel the *receiver* converts the signal back into a message, which is read by a *destination*. Despite its wide use, the suitability of this model outside engineering is debated (Chandler, 1994; Rastier, 2007). Indeed, although the message probably has a particular meaning, the information theoretic perspective on communication is not concerned with meaning-making, but only with the accurate and effective transmission of the message from the source to the destination. Moreover, this meaning may well be different from the points of view of the source and of the destination, and depends on the context in which communication occurs.

The source need not be human, transmission need not be intentional, and in fact, the above process is merely information transmission unless the destination is a human being, in which case the message is interpreted. Interpretation is enabled by the process of signification, where things, or *signs*, are identified with other things they stand for (Eco, 1976). In the context of communication, the signal becomes a sign at the moment one associates a structured unit taken from its physical manifestation, an expression, to a content the expression represents. Communication therefore implies the existence of an underlying *code*—a culturally constructed system of signification that provides *sign-functions*, conventional mappings between expressions and contents.

Eco (1976) takes this definition of sign-function from Hjelmslev (1961), who further subdivided expression and content in six parts using the notions of purport, substance, and form. From the perspective of content, *content-purport* is the undiscriminated and unformed continuum of thought, areas from which *content-substances* are isolated and formed. The differences between them is what defines the respective *content-forms*.

Hjelmslev (1961) gives the following example with the arbitrary process of defining colors (Figure 3.1). Different languages have set different boundaries on the color spectrum: what in English corresponds to the colors green, blue, and gray are all covered by the word "glas" in Welsh; however part of the area covered by the gray color corresponds to "glas" but another part corresponds to "llwyd", together with the brown color. These different content-forms are organized in opposition to each other in a system, each corresponding to a particular unit in the system, which is the content-substance. Thus, the color blue is a content-form opposed to all other colors as part of the subdivisions of colors in the English language. This subdivision is what forms zones of the color spectrum, the content-purport, into content-substances. A similar distinction exists from the perspective of expression: "mouse-down" and "mouse-up" are two opposed *expression-forms* in the system that organizes the *expression-purport* of the continuum of possible elevations of a particular mouse button into two *expression-substances*.
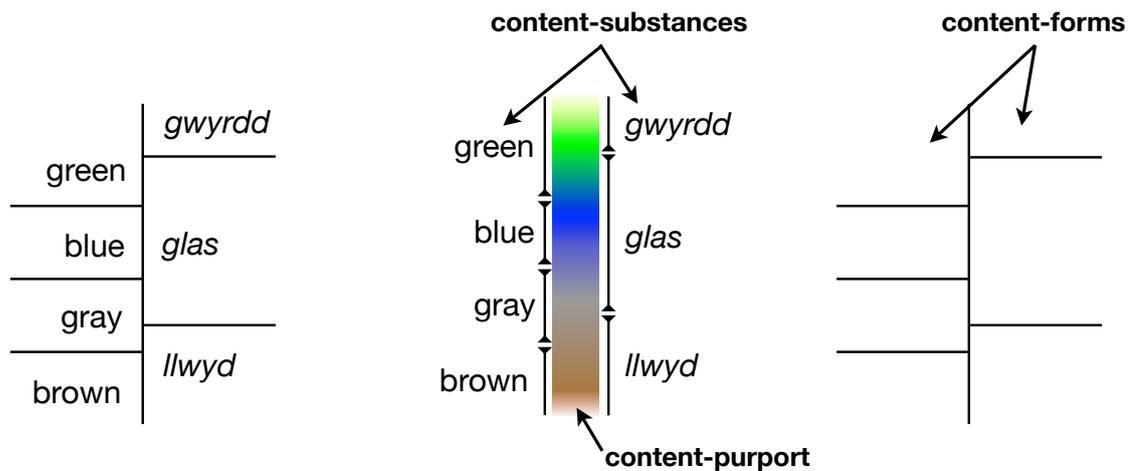
FIGURE 3.1: Illustration of purport, substance, and form with the example from Hjelmslev (1961) of the boundaries between concepts of color in Welsh (italics) and English (left). Several *content-substances* are isolated from the thought continuum of colors (*content-purport*; middle), and formed in a system of oppositions as *content-forms* (right).

Taverniers (2008) summarizes the relationship between purport, substance, and form as follows: "purport provides the substance for a form". But forms have a special place in this distinction because expression-form and content-form are related to each other through a sign-function.

Through a sign, an expression "means" something else, but the expression-substance and content-substance of the sign are also signs (Hjelmslev, 1961). And because signification happens independently of communication, these other signs may trigger yet another sign, and so on, like a dictionary always defines a word by referring to other words, which are defined using other words. Thus, the process of interpretation, or *unlimited semiosis*, is never-ending, and the chain of associations followed by someone at a particular moment is unpredictable (Eco, 1976): meaning evolves over time.

3.1.3. THE SHARED CODE IN HUMAN-COMPUTER INTERACTION

In addition to *user-system* and *user-user* communication, the *semiotic engineering* theory posits that another type of communication also occurs between the system designer and the user, and that it plays an important role in defining the former communication processes (de Souza, 1993). In this *designer-user metacommunication*, the designer communicates (unidirectionally) through the interface her understanding of the needs, goals, preferences, and habits of the user, as well as the user's possibilities for interacting effectively with the system in order to complete what the designer assumes are her goals. Thus, talking about a system's expressiveness can be misleading because it can refer either to whether the designer was successful in conveying the meaning of the system to the user (designer-user metacommunication), or to whether the user is able to convey meaning through the system (user-system and user-user communication). The

second case describes what we are interested in—the user's expressiveness with the system—and all further mentions to expressiveness should be understood in this sense.

Designer-to-user and user-system communication processes share the same code (de Souza, 2005), which a user must learn and use in order to be able not only to effectively use the system, but also to express herself through the system successfully; that is, without risking too many breakdowns due to "misusing the code".

Semiotic engineering discerns three dimensions in the human side of the shared code: *intent*, *expression*, and *content*. The user's intent is realized when there is consistency between her *illocutions*, the effects she wants to have through interaction, and *perlocutions*, the actual effects; in other words, intent is respected when what the user wants to be done is perceivably done. Expression and content are the elements put into correspondence by the shared code, but more generally any forms and corresponding meanings that are used to realize the intent.

Humans and machines have widely different interpretation mechanisms: on the human side, meaning evolves by semiosis, but on the machine side, "meaning" is the result of computation. This creates a *semiotic gulf* (de Souza, 2005) between the user and the system. On the one hand, the code is arbitrary, artificial, and more or less fixed: it reflects the designer's understanding and anticipations on the user's expected activity at design time. But on the other hand, the user's interpretations evolve and are not bounded by the shared code.

3.1.4. EXPRESSION, CONTENT, AND THE USER'S EXPRESSIVENESS

In an effort to understand how the semiotic gulf can be bridged, de Souza (2005) investigated system support for user-initiated customization and extension by examining each possible combination of modifications in intent, expression, and content, on the basis that untargeted and impossible perlocutions may occur if the evolution of expressions, contents, and intents goes too far away from the shared code. Any modification in expression or content automatically expands the scope of intents; that is, the modifications introduced will always enable new illocutions to be formed. This leaves four combinations of modifications in expression and content that may or may not require changes in the shared code. To extend the code, either the system should be adaptable, or new versions of the system should include these evolutions. But our interest in this study is not the implications for adaptable software. This study is useful to us because it shows not only how the human side of expressiveness depends on the system side, but also how designers can improve expressiveness in human-computer interaction.

First, even with the same expressions, the system's perlocutions can still match illocutions and meanings that are different from those intended by the designer and encoded in the system (de Souza, 2005). There are two ways in which the meaning and intent associated to an expression can evolve even when the system does not: either the expression keeps the same meaning but is used for new goals (*repurposing*), or its meaning is changed in the mind of the user (*figurative speech*). In the first case, an existing expression and its associated content can be repurposed for a

different intent, one that was not foreseen by the designer of the system: for instance, when we cheerfully produce error beeps using the keyboard to produce familiar rhythmic patterns. In the second case, an expression can also be attributed another content and intent, for example when we try to transpose the effect of an existing action in a new context by metaphor or metonymy. Thus, a system can support unintended uses as long as no new expressions or contents are required, albeit in a limited fashion. In the long run, the introduction of new contents in the code is required for perennial uses of figurative speech.

Second, the introduction of new expressions can have two different effects regarding user-system communication (de Souza, 2005): either it allows to do something that can already be done in a new way (*rephrasing*), or it allows to do new things that could not be done before (*linguistic expansion*). In the first case, rephrasing, the new expression can be associated with an existing meaning in the shared code. The properties of the new expression can have different effects on interaction: for example, perhaps the new expression allows the same meaning to be conveyed more quickly in a given context. In the second case, the new expression entails new meanings and intents that are supported by the system: its linguistic domain is fully extended up to the semantic level.

Beyond user-system communication, expressions can be used independently of their meaning in the shared code. For example, file managers can be configured to sort the contents of a folder by some property (e.g., name or file size) or to leave the sorting task to the user, who can freely position files in the 2D space of the window. Free arrangement of file icons was found to be widely used in studies of file retrieval and organization, not only due to its sorting function, as expected in the shared code, but rather because users gave it a crucial reminding function (Barreau and Nardi, 1995; Zacchi and Shipman, 2007). Apart from changing the sort order, leaving a file near the trash can icon of the Windows desktop may not carry any particular meaning for the system, but it reminds us that we will have to delete it at some point (Hollan et al., 2000). When support for free arrangement is inadequate[1], the user is also deprived of its reminding function.

Thus, the user's expressiveness relies partly on her ability to find other uses and meanings for the expressions allowed by the interface, and partly of the system's support for the formulation of intentions and meaning. However, as Figure 3.2 shows[2], the system can also hinder the user's expressiveness when introducing changes. Generally, improving the user's expressiveness with the system consists in introducing new expressions, either by providing ways of rephrasing existing meaningful interface concepts, or by introducing new contents, together with ways of expressing them. Nevertheless, one should keep in mind that alternative intentions and meanings can always be formulated by the user independently of the shared code.

---

[1]System-initiated rearrangements happen even with modern file managers: for instance, try unplugging a computer monitor.

[2]Note that the positions of the points in Figure 3.2 are only suggestive. For instance, a rephrasing (the introduction of a new expression for an existing meaning) can be successful even if the former expression is impossible to perform.
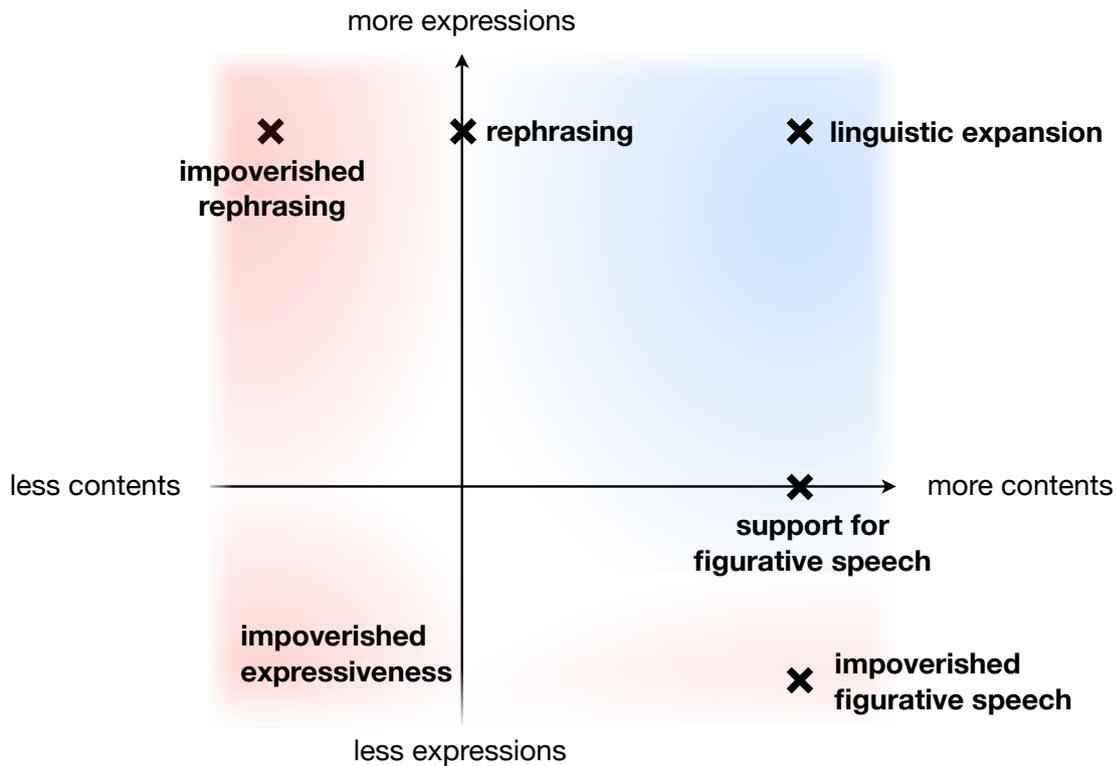
FIGURE 3.2: Effects on the user's expressiveness of the introduction of new expressions and contents in a system. Around the upper-left quadrant (in blue), possible introductions have positive effects: rephrasing, support for figurative speech, and linguistic expansion. All other quadrants (in red) may have negative effects on the user's previous ability to convey meaning.

3.2. Expressive power

Expressions, meanings, and intents can always be richer and more diverse in the minds of users and bystanders than in the designer's original interpretation of what could be expressed through the system. Therefore, at this point, it seems important to separate what a user can do through interaction with the system from what she can do that has meaning in the system. The former refers to the user's expressiveness and depends on her ability to express herself with the system, but we will call the latter *expressive power*, the potential offered by the designer of the system. If the user's knowledge and mastery of a system is sufficient, her expressiveness is enabled by the system's expressive power; that is, the amount of contents and expressions available in the shared code. Our goal in this section is to clarify what constitutes the system side of the shared code, and how it can be extended.

3.2.1. EXPRESSION AND CONTENT ON THE SYSTEM SIDE

The first step in our discussion of expressive power is to define counterparts to the notions of expression and content. To this end, we will use the pipelines model proposed by Nigay (1994), which is complementary to Norman's stages-of-action model (Section 2.1.2), in that the latter
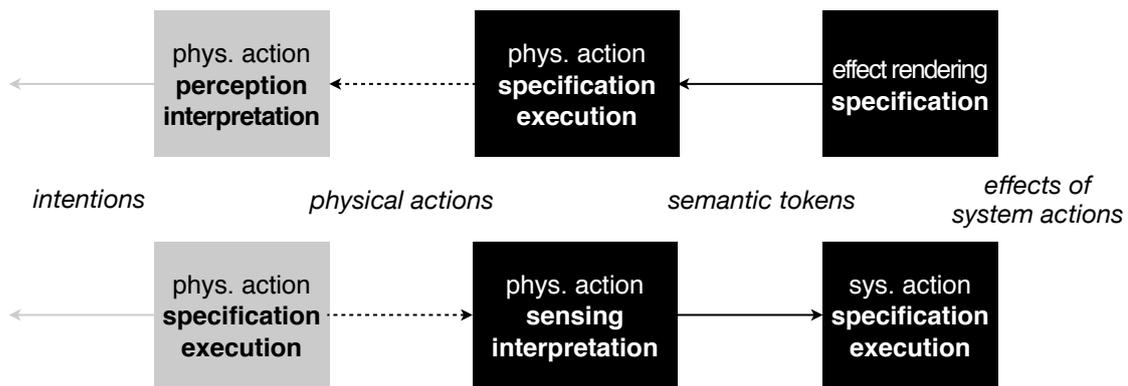
FIGURE 3.3: The six functions of the pipelines model (Nigay, 1994), distributed between user (gray boxes) and system (black boxes). Solid arrows indicate mental or computational activity, while dashed arrows indicate physical actions.

detailed the user side of human-computer interaction, while the former focuses on the system side. The pipelines model divides interaction in six functions representing the cycle of successive transformations of information between intention, physical action, semantic token[3], system action, and effect (Figure 3.3):

Physical action specification and execution The user translates an intention into a proper sequence of actions that will match the intention, and then executes it through the system. This function regroups a mental operation and a physical operation, both performed by the user, that are equivalent to the *Action specification* and *Execution* stages in Norman's model (Norman, 1987).

Physical action sensing and interpretation The system senses the user's physical actions and translates them into *semantic tokens*. Interpretation also takes into account the interaction context; that is, information relevant to interaction encoded in the system state. This context includes, for example, the current window, or the selected object.

System action specification and execution From semantic tokens, the system generates and executes a sequence of system actions; that is, uninterruptible units of computation. Each system action produces an effect, in that it changes the state of the system.

Effect rendering specification An effect might give rise to other semantic tokens representing its consequences and the associated feedback. Note that an effect may or may not be related to a user-issued system action.

User-directed physical action specification and execution The system then translates rendering-related semantic tokens into a sequence of physical actions that can be perceived by the user, and executes the sequence.

Physical action perception and interpretation The user perceives the physical actions issued by the system and interprets them with respect to her intentions and goals. This function is equivalent to the *Perception* and *Interpretation* stages in Norman's model.

---

[3]What we call *semantic token* is called *informational unit* in Nigay (1994) and Coutaz and Nigay (1994).

In the pipelines model, the user and the system communicate using *interaction languages*, which define structures of syntactic, temporal, and causal relationships that structure expressions mapped to semantic tokens, and *input devices*[4], which transduce physical properties changed by the user's physical actions. To successfully produce the expected perlocutions, the user must structure her physical actions on input devices so as to form valid expressions with respect to the interaction language that must be used. Using these notions, we can now more precisely define expression and content, whose correspondence make up the shared code in user-system communication.

An *expression* is the product of one or several physical actions on one or more input devices and structured by an interaction language. The devices define the expression-substances from the expression-purports they sense, while the interaction language defines the expression-forms (Nigay, 1994). Input devices and interaction languages will be discussed in more detail in the following.

A *content* is a semantic token that can be produced by the system in a given context after interpretation of an input expression, and that can be processed by the system in order to generate system actions which have effects. In the same way as Eco (1976) distinguishes *tokens* as instances of more general *types* of expression or content, we will assume that semantic tokens are specific instances of more general semantic types: for instance, "open the Preferences window" is a semantic token of the type "open window X". In a sense, some semantic types correspond to tasks the user may want to perform. Because the interpretation of semantic tokens by the system leads to effects, they also encompass the system side of intent (de Souza, 2005): they generate perlocutions that may or may not correspond to the user's illocutions.

Totally new expressions and contents can be introduced in the shared code by various means, some of which will be detailed in the next section. But what can be done to extend an existing base of expressions ? We will first introduce general notions about input devices, and then discuss a framework centered around the extension of interaction languages.

3.2.2. INPUT DEVICES

Mackinlay et al. (1990) developed a semantic model of input based on a general notion of *device*, an abstract object that converts an input value to an output value. The notion of device in this model brings together physical input devices, such as a mouse, virtual devices, such as a pointer, as well as application parameters, such as the zoom level of a view. A device has the following six components:

Operator The operator describes how the input value is read. When the device senses a physical property, it is an *input device*, and the operator is a *manipulation operator* that describes the nature and components of the sensed physical property (for example, a translation along the X axis).

---

[4]Nigay (1994) also defines output devices, but we are primarily concerned with the input side of the pipelines model.

Input domain (In**)** The input domain is the set of input values that can be read. The number of dimensions of the input domain is n_{In}.

Device state The state of the device includes the input and output values, as well as other internal parameters at a given instant.

Resolution function (f**)** The resolution function describes how an input value from the input domain is transformed into an output value from the output domain. In the following, we will use *resolution function* and *transfer function* indiscriminately.

Output domain (Out**)** The output domain is the set of output values that can be produced. The number of dimensions of the input domain is n_{Out}. For a physical input device, another important measure related to the output domain is the resolution, or *sensitivity*[5], in counts per inch (CPI).

Works The works of the device are additional rules that describe its physical properties or internal mechanisms (for example, the self-centering effect of isometric input devices[6]).

The composition of two or more devices is also represented as a device. Mackinlay et al. (1990) describe three ways in which devices can be composed: *connection* (the mapping between the output domain of a device and the input domain of another), *layout composition* (the physical disposition of adjacent input devices in space), and *merge composition* (the combination of several devices to form higher-dimensional input and output domains). This way, a three-button mouse can be described as the following composition: two devices, each sensing relative position on the X and Y axis, are merged into a two-dimensional one, and laid out together with three "mouse button" devices. In graphical user interfaces, the mouse is also connected to another device, the pointer, which maps relative two-dimensional displacements into an absolute position on the screen.

3.2.3. EXTENDING THE INTERACTION LANGUAGE

Several representations exist that can be used to describe an interaction language at various levels, such as GOMS (Card et al., 1983), UAN (Hartson et al., 1990), the three-state model of input (Buxton, 1990), or CIS (Appert et al., 2005). Our concern being the expansion of expressive power, we will rather focus on the framework proposed by Cechanowicz and Gutwin (2009), which can be used to analyze and generate a wide range of "augmentations" of existing interaction languages in graphical user interfaces.

The augmented interactions framework is concerned with actions on the visible objects of the interface that visually represent data. An action consists in manipulating the object using input devices, and can be described according to the number of dimensions the action operates upon in the represented data and the continuity (discrete/continuous) of the dimensions. The most common are *1D-D*(iscrete) actions, which change the state of an object, and *1D-C*(ontinuous)

---

[5]Although the notion of resolution applies to any physical device used for input or output, we will differentiate the *sensitivity* of an input device, in counts per inch (CPI), from the *density* of a display, in pixels per inch (PPI).
[6]See Section 3.3.1 for details.

and *2D-C* actions, which describe changes of continuous values. They represent basic lexical types that can be combined to form higher-level actions: for instance, dragging an object across the display combines a 1D-D action and a 2D-C action. Combination creates augmentations of the base interaction language that allow new expressions to be formed.

In fact, the augmented interactions framework focuses primarily on augmenting the interaction language. Actions are discussed independently of the input devices used to perform them, as well as the meanings they can convey. The framework proposes seven possible augmentations to the interaction language that represent plausible combinations of the two following strategies, applied to 1D and 2D actions: increasing the number of controllable states in a given dimension, or increasing the number of controllable dimensions (*dimensionality*). The number of states of the interaction language can be increased in two ways, according to the augmented interactions framework: by adding new states to a discrete action, or by upgrading a discrete action to a continuous one.

Instead of separating discrete from continuous actions, if the controllable states of a given dimension can be represented numerically, we will rather speak of granularity and extent. *Granularity* is the difference between two consecutive controllable values. Increasing granularity makes a dimension "more continuous", and consists in allowing more fine-grained values than was initially possible. This can be done either by providing more controllable values or by dynamically changing granularity in the course of the action. On the other hand, *extent* is simply the range of controllable values in the considered dimension. Increasing extent thus consists in allowing more extreme values than was initially possible, either by extending the range or by dynamically shifting it during the action.

The framework (Cechanowicz and Gutwin, 2009) also discusses how augmentations that increase the *dimensionality* of an action can be mapped to new contents. Adding a 1D-D action introduces modes or quasimodes (Raskin, 2000) and can allow to express variations of a content (e.g., holding the ctrl key during drag-and-drop transforms the meaning of the action from "move" to "copy"). Adding a 1D-C action to a 1D-D one permits the specification of a continuous value to parameterize a command, as can be done with FlowMenus (Guimbretière and Winograd, 2000), adding it to a 1D-C one permits integral control of two continuous values, like OrthoZoom (Appert and Fekete, 2006) combines vertical scrolling and zooming, adding it to a 2D-C one gives control over three dimensions, and so on. Finally, adding a 2D-C action to a 2D-C one can allow, for example, the simultaneous rotation, scaling, and translation of a two-dimensional object.

In addition, Cechanowicz and Gutwin (2009) discuss four ways in which input can be used in order to make augmentations possible: considering time as another input dimension (*timing*), linking two dimensions of the action by using constraints (*constraining*), leveraging otherwise unused degrees of freedom in input (*filling*), or using new input capabilities or input devices

(*adding*). Among these methods, timing and constraining are parsimonious with respect to the input devices used in the formation of expressions, while filling and adding are not.

In summary, from our reading of the augmented interactions framework, there are three general ways in which an existing base of expressions can be extended: by increasing their *granularity*, by increasing their *extent* for a particular dimension, and by increasing their *dimensionality*.

3.3. Increasing expressiveness on the desktop

In this section, we will outline general strategies to increase the user's expressiveness and detail many examples in the literature. We limit ourselves to the context of desktops and laptops, as they are the systems we are interested in, thus we first need to detail their standard input apparatus. We will then focus on two general strategies to increase expressiveness by forming new expression-content mappings. As we will see, while they can provide interesting expression opportunities for the user, they do not necessarily increase the expressive power of the system. The next three strategies are based on the previously uncovered ways to expand a base of existing expressions: increasing *granularity*, *extent*, and *dimensionality*.

3.3.1. COMMON DESKTOP AND LAPTOP INPUT DEVICES

We consider that standard desktops and laptops provide at least four different input devices: a *pointing device* integrated with *pointing buttons* and a *scrolling device*, as well as a *keyboard*. In addition, laptops frequently integrate a *microphone*, a *camera*, and various sensors. We will briefly describe these devices and the inputs they provide.

Pointing device The main pointing device in a desktop or laptop system is indirect; that is, the output of the system (typically the movement of a pointer on the screen) is not collocated with the user's input. The most common pointing devices are the mouse and the touchpad. A mouse senses its relative position in two dimensions (X, Y) on the surface it lies on. A touchpad is a touch-sensitive surface that senses the relative or absolute position of one or more fingers lying on it in two dimensions, and while it can be used for several purposes, the movement of an on-screen pointer is controlled by sensing the relative position of only one finger. Both dimensions are usually aligned with the anteroposterior and mediolateral axes of the user.

There exist other pointing devices that are less commonly used, but they will not be considered in this research. The mouse and the touchpad, like the trackball or the graphics tablet and stylus, are isotonic: they do not oppose resistance and measure relative or absolute position. Other pointing devices, like the TrackPoint (Rutledge and Selker, 1990) that is integrated in some desktop and laptop keyboards (Douglas and Mithal, 1994), are isometric or elastic: they oppose resistance but limit or suppress movement, and sense force instead of motion. Isometric devices are *self-centering*: they return to a neutral position once the user no longer applies force.

Pointing buttons A set of one, two, three, or more buttons are usually integrated within the pointing device. Mice commonly feature three buttons, the central one being also a scrolling device (see below). Touchpads feature buttons either above or below the touch-sensitive surface, but sometimes the surface itself acts as a button and can be pressed. Buttons sense only two states (i.e., their output domain is {pressed, released}), usually along the Z axis normal to the surface the device rests on, and provide haptic feedback when state changes. Pointing buttons are thus named because they are controlled using the same hand that controls the pointing device.

Scrolling device The pointing device also integrates a scrolling device that typically senses relative position in one or two dimensions. Although isometric scrolling devices exist (Zhai et al., 1997b), they are uncommon. The main dimension sensed by a scrolling device is usually aligned with the anteroposterior axis of the user. Scrollwheels, which are present in mice between the left and right button and also frequently act as a button, are often limited to this dimension. Touchpads provide a virtual scrolling device that more frequently senses both dimensions, either by dedicating a small vertical and horizontal strip in their surface, or by interpreting two-finger gestures as scrolling movements. While various researchers and designers have described the advantages of using the scrolling device with the non-pointing hand (Buxton and Myers, 1986; Zhai et al., 1997b; McLoone et al., 2003), scrolling is normally performed with the pointing hand in desktop and laptop systems.

Keyboard A keyboard features a layout of about a hundred two-state buttons, or *keys*. Most keys are alphanumeric, such as $\boxed{\text{A}}$ or $\boxed{\text{=}}$: they are associated with symbols that are produced as an input when pressed. Function keys, such as arrow keys or $\boxed{\text{F1}}$, serve diverse and more or less fixed functions. Modifier keys, such as $\boxed{\text{ctrl}}$ or $\boxed{\text{⇧}}$, are not meant to issue actions but rather to change the effect of other key presses; pressing several keys at once is called *chording*.

Microphone, camera, and other sensors Laptops commonly integrate a microphone and a camera to provide audio and video input capabilities. The camera is usually located above the frame of the display, approximately level with the user's face, and directed towards her. The location of the microphone varies greatly between laptop manufacturers: some are positioned next to the camera, while some others are found around the keyboard or on one side of the laptop. Some laptops also feature various other sensors, such as accelerometers used to prevent hard disk damage during shocks, or light sensors used to adapt display brightness, that can be deliberately used as input channels.

3.3.2. REPHRASING AND ENRICHMENT

As seen previously, the user's expressiveness can be augmented by making new expression-content correspondences in the shared code; that is, by changing the system's expressive power. New

contents result from additions at the semantic level, that of semantic tokens and types, while new expressions result from additions at the interaction language or input device levels. Based on the systematic study by de Souza (2005) described on Section 3.1.4, we identify two general strategies that intervene at the level of mappings between expressions and contents. *Rephrasing* concerns the expression of an existing content using previously unrelated expressions, and *enrichment* allows expressions to express previously unrelated contents.

REPHRASING

Rephrasing consists in providing new ways to express existing semantic types by using another interaction language. Rephrasing can be inspired from existing interaction languages that are transposed and adapted to fit another context, developed from basic input paradigms or laws of human behavior, and introduced by the availability of non-standard input devices.

For example, principles from computer games have been reused in other contexts to provide a different perspective on existing tasks. Motivated by the referential match between the vocabulary of system administration (e.g., killing daemons) and the violence of first-person shooters, Chao (2001) rephrased UNIX process management, which is usually performed by using a set of shell programs in command line, into a modified version of Doom. Monsters represented running processes the user could shoot at, and once a monster was killed, so was the corresponding process. Similarly, Berthaut et al. (2011) transposed musical performance into the language of multiplayer first-person shooters to leverage the existing control and collaborative skills of players to serve musical purposes.

Activating commands by drawing virtual marks on the display is another example of rephrasing. Marking menus (Kurtenbach and Buxton, 1994) are hierarchical radial menus that support the smooth transition between novice and expert performance, unlike other means of command activation such as linear menus, toolbars, and keyboard shortcuts (Lane et al., 2005; Scarr et al., 2011). After invocation by pressing a button, a radial menu displays all items at the first level around the invocation point. Then, still holding the button, the user draws a mark in the direction of the desired item and a submenu possibly appears. Once the desired terminal item is selected, releasing the button activates the command. Expert users can just quickly draw the mark corresponding to the path taken to select the menu item, which has been rehearsed through practice. GestKeyboard (Zhang and Li, 2014) is another example of rephrasing of command invocation based on gestures over an unmodified computer keyboard.

Other rephrasings are inspired from changes in basic input paradigms. *Crossing* (Accot and Zhai, 2002) provides an alternative to the pointing-and-clicking actions that pervade graphical user interfaces: instead of acquiring a target and then pressing it to activate a command, one can just intersect the target. Accot and Zhai (2002) described how several widgets such as buttons and checkboxes could be rephrased in the crossing paradigm, and Moscovich (2009) showed how crossing-based widgets that must be slid each in a particular direction can resolve ambiguities in

the interaction with small targets on touchscreens. Apitz and Guimbretière (2004) generalized crossing to the entire interface of a drawing application featuring crossing-based hierarchical menus and scrollbars. They also demonstrated how crossing enabled composing several actions in a single stroke, a point also explored by Baudisch (1998) in his notion of "toggle maps" that can be "painted". Perin et al. (2015) explored crossing-based sliders that leverage interaction with the orthogonal dimension of several aligned sliders to manipulate them simultaneously. They also reified the mark left by crossing gestures and showed that it could be amenable to various manipulations. Motion along a predetermined trajectory, or *steering* (Accot and Zhai, 1997), also inspired the design of new techniques: Yamanaka and Miyashita (2013) proposed *nudging* graphical objects by pushing their boundaries from outside as a rephrasing of dragging actions.

The addition of non-standard input devices can also prove useful for rephrasing existing functions. For example, toolglasses are transparent palettes controlled using a secondary pointing device that can be used to specify both a tool and an object to which it applies in a single two-handed gesture (Bier et al., 1993). Kurtenbach et al. (1997) demonstrate how two rotation sensing pucks on tablets provide users with the opportunity to simultaneously interact with domain objects and to translate or rotate the workspace. Bailly et al. (2013) extend the computer keyboard with actuated force-sensitive keys that, when raised, can be manipulated in ten different ways to select discrete commands and specify continuous parameters at the same time. Head and gaze tracking techniques and devices have also been proposed to orient a camera (Gaver et al., 1995) or to scroll a window (Bérard, 1999; Kumar et al., 2007), leaving the user's hands free for other tasks.

ENRICHMENT

Another strategy to change the mappings of the shared code consists in extending the semantic types associated with an existing interaction language. The enriched content can be expressed depending on the current interactive state only, or on particular actions performed by the user.

Many enrichments have been proposed that apply to simple dragging actions in particular situations. Some do not change the way actions are performed but change their meaning depending on the current state of the system. Chapuis and Roussel (2007) describe the burden of selecting objects from a background window: the window is brought to the foreground at the beginning of the selection, hiding potential targets in upcoming drag-and-drop or copy-paste actions; but if the window was not brought to the foreground at the beginning of a selection, critical parts of the upcoming selection would have been hidden behind other windows. They propose two refinements to the semantics of selection when it occurs on a background window: one momentarily pops the window to the foreground until the end of the dragging action (*restack*), and the other curls all the overlapping windows so that the entire content of the selected window can be seen (*roll*). Beaudouin-Lafon (2001) experimented with windows that rotate when dragged to facilitate their selection in piles of windows, and proposed peeling back a window by dragging one of its corners in order to reveal the underlying ones until the dragging action stops and the window folds back.

Some enrichments to dragging instead discriminate particular properties of user actions and attribute them alternative meanings. The Boomerang technique (Kobayashi and Igarashi, 2007) allows the user to suspend a drag-and-drop action by throwing the dragged file above a threshold velocity. A translucent circle then appears, allowing to resume the action by dragging it. Dwell-and-spring (Appert et al., 2012) enriches any dragging action with cancel and undo commands by presenting a spring attaching the former position of the dragged object and the current one when the user dwells: finishing the dragging action on the spring handle puts the dragged object back to its original position, but moving elsewhere makes the spring disappear. Faure et al. (2009) propose a "trailing" widget that follows the pointer during drag-and-drop and reveals a crossing-activated radial menu allowing to perform various window management operations such as *desk pop*, a variant of the "Show Desktop" command where desktop icons are superimposed over existing windows. Fold-and-drop (Dragicevic, 2004) enables the user to leaf through overlapping windows during dragging: a small fold appears on each window the dragged object leaves, and can be pushed by moving back towards them to reveal the underlying windows. Bubble clusters (Watanabe et al., 2007) enrich free spatial layout, which is commonly used to implicitly express temporary relationships and emerging structures (Marshall and Shipman III, 1995), by reifying aggregations of objects, allowing the user to reorganize them by dragging objects around, and to act on groups without having to express selections.

As with rephrasing, introducing non-standard input devices can also prompt the design of enrichments. Zeleznik et al. (2001) show how three-state "pop-through" push buttons can be used to show more advanced options in a menu, to bring up contextual menus and dialog boxes, or to complete specific actions associated with the selection that has been performed by normal clicking. Forlines et al. (2005) leverage multi-level input devices, such as pressure-sensitive mouse buttons (Cechanowicz et al., 2007) or keyboards (Dietz et al., 2009), to enrich clicking and dragging with undo commands in order to facilitate exploration: if the device is pressed lightly, the command is temporarily performed, but to confirm it the user must press harder, or else the system comes back to its previous state.

In summary, rephrasing and enrichment strategies play with the mappings between expression and content, and possibly involve non-standard input devices. While they can contribute to the user's expressiveness with the system because they offer new opportunities for creative uses, the expressive power of the system is not systematically increased, as opposed to the following three strategies: increasing *granularity*, *extent*, and *dimensionality*.

3.3.3. INCREASING GRANULARITY

Increasing the granularity of discrete actions performed with buttons or keys requires non-standard input devices, such as the pressure-sensitive mouse (Cechanowicz et al., 2007) or keyboard (Dietz et al., 2009) cited earlier. For pointing and dragging actions, granularity is limited due to the fact that screen real-estate is likely not enough to display each desirable position for a graphical object with one pixel. Thus, pixels limit the granularity of pointing actions.

Users can circumvent this issue if they can change the zoom level of the displayed objects, but this requires much work to go set an appropriate zoom level (e.g., using a slider), perform the action, and reset the zoom level afterwards (*turn-taking*, *spatial proximity*). Several attempts to increase granularity have been discussed in the literature that are based either on changes in the mapping between the position of the pointer and that of a dragged object, or on temporary zooming modes.

CHANGING POINTER MAPPINGS

Ahlberg and Shneiderman (1994a) proposed several designs of Alphasliders, variations on the standard slider widget that allow the user to vary the granularity at which the slider thumb is positioned while dragging. In the "acceleration" variant, the mapping between pointer displacement and the number of items skipped varies depending on the velocity of the pointer, with only three levels of granularity (fine, medium, coarse). In the "micrometer" variant, granularity level is adjusted using pointer motion orthogonal to the slider. However, Alphasliders reduce *compatibility*, they only apply to one-dimensional parameter control, and they do not generalize to all actions performed by pointing.

Masui et al. (1995) described an alternative approach, FineSlider, based on the metaphor of a rubber-band between the pointer and the manipulated object, which is pulled more or less strongly depending on the length of the rubber-band, resulting in a dynamic rate of value change. This approach also extends to general 2D manipulation. The FineSlider approach decreases *spatial proximity* because it requires that the user drags a point remote from the object she wants to manipulate. In addition, in its 2D generalization, the object to be manipulated has to be selected beforehand (*turn-taking*).

Circular motion has also been used to provide variable granularity in the control of a one-dimensional parameter. The Virtual Scroll Ring (Moscovich and Hughes, 2004) and the Radial Scroll Tool (Smith and schraefel, 2004), originally applied in a scrolling context but also applicable here, are both based on mapping the distance from a central point to the granularity of value change: small circles cause large changes of the parameter value, while larger circles allow finer changes. As with Alphasliders, circular motion decreases *compatibility* and only applies to one-dimensional control.

TEMPORARY ZOOMING MODES

Other works use a temporary zooming mode that can be activated in order to explicitly control the desired granularity. For example, OrthoZoom (Appert and Fekete, 2006) map movement along the orthogonal dimension of a slider to zoom level, but has similar limitations to the "acceleration" Alphaslider.

Ayatsuka et al. (1998) proposed to map a key or mouse button to the activation of a "Popup Vernier" mode. Once activated, a zoomed grid with verniers is displayed behind the dragged

object: in this mode, pointer movements displace the grid commensurately in the opposite direction (*compatibility*), but also displace the object by a tenth of the distance. Subsequent presses of the Popup Vernier key decrease the scale of manipulation by one tenth.

Ramos and Balakrishnan (2005) proposed Zliding, an interaction technique for one-dimensional sliders also based on the control of the zoom level. The variant preferred by participants in their controlled experiment is based on pressing two modifier keys to increase or decrease the scale factor. Another variant uses the level of pressure on the tip of a stylus to control the zoom level, but despite the opportunity of concurrent control of position and granularity it did not yield superior performance.

Oscillatory movement can also be used to control the zoom level, as proposed by Olwal et al. (2008) with the Rub-Pointing technique. Repeated diagonal motion, or "rubbing", is interpreted as zooming in when performed in a direction, while it is interpreted as resetting the zoom level in the other direction. While this technique was designed for touchscreen interaction, it could be used with an indirect pointing device during dragging.

These methods decrease *semantic distance*, because a high-level fine-grained selection intention must be decomposed into a selection intention and a zoom level control intention. Moreover, apart from the pressure-controlled Zliding variant, which requires non-standard input, the mappings used to control zoom level are arbitrary, thus exhibiting a high referential distance.

In focus+context interfaces, a zoomed view is already provided by a lens whose center is usually locked to the position of the pointer, but two issues remain: how to switch between display-scale and lens-scale control, and how to summon the lens when needed? Appert et al. (2010) compared three interaction techniques for switching between scales of control that rely on different mechanisms. The *Key* technique simply relies on holding a modifier key to temporarily switch the granularity of control from display-scale to lens-scale. The *Speed* technique smoothly switches between both levels of granularity depending on the velocity of the pointing device. The better-performing *Ring* technique decouples the center of the focus view from the pointer, which moves freely inside the view at lens-scale; but crossing the lens pushes it at display-scale.

But most applications do not require the lens to be always visible. Ramos et al. (2007) described several ways of activating pointing lenses, which were intended to facilitate pointing at small targets in collocated pen interaction, but could also be used on the desktop to enable dragging and pointing at targets smaller than a pixel. Delay-activated lenses appear after dwelling for 400 ms, and can be pushed by gently reaching a border in order to reveal the surroundings of the magnified area. The lens disappears if the pointer crosses its edge above a predetermined velocity threshold. In another variant, the lens is always trailing the pointer, and quickly entering it allows to interact with the magnified content. Finally, a pressure-activated variant was found faster than the other variants and was also preferred, but it requires non-standard input in order to be used on the desktop.

3.3.4. INCREASING EXTENT

With pointing actions, the issue of extent is mainly characterized by the fact that the display area where pointing is permitted does not cover the entire range of underlying values that could be selected or controlled. This leaves the burden of increasing the dedicated display area or setting its extent to the user, who must either perform this setup action beforehand, or do it in parallel with the primary action.

The standard scrolling device can be used for this task, but there are reasons to look for alternatives, some of which relate to directness. The use of a scrolling device is deliberate (*semantic distance*), and may be difficult for some users due to the fact that it is controlled using the same hand that performs spatial manipulation. In addition, scrollwheels are often limited to one scrolling direction at a time[7], thus two-dimensional scrolling must be controlled separably despite being an integral task (*turn-taking*). Moreover, the scrolling device is not always mapped to scrolling functions: for example, map navigation (e.g., Google Maps) and 3D modeling applications (e.g., Blender) use it to control the zoom level.

Instead, Malacria (2011) proposes that the CycloPan technique that maps linear oscillatory gestures along a line to 2D scrolling (Malacria et al., 2010) could be used while dragging, but we have already seen that oscillatory movement rates poorly in *compatibility*.

Another approach is to leverage the zone beyond the dedicated display area for scrolling. The Position + Velocity slider described by Ramos and Balakrishnan (2003) applies this method: when the pointer crosses the left or right boundary of the horizontal slider, the interval of values mapped by it is pushed further to the left or to the right. The distance between the pointer and the considered boundary defines the rate of displacement of the interval.

This is similar to a more general interaction, edge-scrolling, that has been observed in all kinds of desktop applications in the past twenty years (e.g., Berry et al., 1991; Furnas and Zhang, 1998): during the manipulation of a graphical object, crossing the boundary of the viewport it is enclosed in automatically scrolls the document, as if it was a direct extension of the dragging movement. Curiously, however, few works have investigated its design and its effectiveness.

3.3.5. INCREASING DIMENSIONALITY

In many cases, the device imposes a hard limit on the number of dimensions that can be manipulated simultaneously, and this may not be enough for some applications, such as 2D and 3D spatial manipulations. In this case, when no superior device is available, manipulation has to be carried out sequentially (*turn-taking*). However, another application domain, musical expression, shows that creatively harnessing existing devices can provide ways to increase dimensionality.

---

[7]For instance, in OS X, the scrollwheel is assigned to vertical scrolling by default, but horizontal scrolling can be performed by holding the ⇧ modifier.

INCREASING THE DIMENSIONALITY OF SPATIAL MANIPULATIONS

Several extensions to the mouse have been proposed to increase the number of degrees of freedom sensed and enable users to perform a wider range of integral manipulations of 2D and 3D objects. Ohno et al. (1985) first introduced the mouse wheel in a five-button mouse prototype in order to control the rotation of the pointer. Venolia (1993) included two wheels on top of a "3D mouse" that controlled the depth of a pointer on a 3D environment. The wheel was later integrated in commercial mice and is now commonplace, but it usually rephrases scrolling instead of increasing the dimensionality of spatial manipulation. MacKenzie et al. (1997) describe a modified mechanical mouse including a second ball, which allows it to sense relative orientation along the Z axis in addition to displacements on the XY plane. With such a mouse, the position and the rotation of a 2D object can be changed integrally. Almeida and Cubaud (2006) built a similar "yawing mouse" prototype by sticking two optical mice together side by side. Simultaneous control of position and orientation has also been investigated using a mouse equipped with two pressure sensors by Shi et al. (2009), who compared four mappings between pressure level and rotation angle.

The Rockin'mouse (Balakrishnan et al., 1997) adds rotation along the X and Y axes to the sensed properties of the standard mouse, and has been used to control the position of a 3D object along the three dimensions simultaneously. Hinckley et al. (1999) describe the VideoMouse, which uses a camera as a mouse sensor along with LEDs that illuminate a gridded mousepad, and employs computer vision techniques to sense motion along six degrees of freedom. While translation on the Z axis may be set aside to allow lifting and repositioning the mouse, the remaining 5 DOFs can be used to manipulate 3D objects. Since then, commercial 6-DOF mice, such as the SpaceMouse, have appeared in the consumer market, but other prototypes have been devised: for example, Perelman et al. (2015) proposed a round "roly-poly" mouse that outperforms the SpaceMouse in 3D translation tasks.

INCREASING THE DIMENSIONALITY OF MUSICAL EXPRESSIONS

Another application domain where concurrent multidimensional input is of first importance is the control of sound synthesis (Hunt and Kirk, 2000). Musical performance, because of its strong temporal constraints (Orio et al., 2001), is likely to require the control of a large number of parameters in real-time. Consequently, most uses of standard desktop and laptop input in music concern knowledge-based behavior (Malloch et al., 2006)—e.g., live coding (Collins et al., 2003)—and rule-based behavior—e.g., using sequencer software, or dragging-and-dropping samples (Wessel and Wright, 2002)—while skill-based behavior is mostly performed using dedicated external *controllers*, either the many ones commercially available or prototypes, for example those designed by members of research communities such as NIME (New Interfaces for Musical Expression). Examples of skill-based performance using standard desktop or laptop input are less common but nevertheless exist. Most leverage graphical interaction, such as the musical first-person shooter mentioned earlier (Berthaut et al., 2011), the graphically manipulable

scanned synthesis interface of Couturier (2006), or the Metasurface (Bencina, 2005) that allows the interpolation of several predetermined sets of parameters by moving a point on a two-dimensional area.

Few other examples bypass the display and directly use the input from standard devices. Jordà (2005) describes one of his former instruments, the QWERTYCaster, which is an assemblage of a computer keyboard, a trackball, and a joystick roughly resembling a guitar. Fiebrink et al. (2007) detail how the keyboard, the touchpad, and the microphone present in standard laptops have been used individually for performing various compositions. However, these examples present limited dimensionality, even compared to the possible number of simultaneous inputs that a standard laptop can provide.

3.4. Increasing granularity, extent, and dimensionality, without indirectness, on standard desktops and laptops

The rest of this dissertation focuses on interaction techniques that increase the granularity, extent, and dimensionality of user actions, with two requirements: they should not introduce too much indirectness, and they should be limited to the standard input devices found in desktops and laptops. As we have seen, existing techniques suggest that there is a tradeoff between expressive power, directness, and the limitation to standard input devices: increasing expressive power requires better input devices or richer interaction languages, leading to possible decreases in directness unless adequate non-standard devices are added. Our approach to this tradeoff consists in starting from standard devices and to reconsider how their composition leads to the control of parameters with a satisfying degree of expressive power, while maintaining low referential distance and reusing existing skills. We will now detail the contributions of this thesis in three parts.

3.4.1. INCREASING THE GRANULARITY OF BASIC POINTING AND DRAGGING ACTIONS WITH SUBPIXEL INTERACTION

Concerning the granularity of basic pointing and dragging actions, existing approaches do not question the fact that it is fundamentally limited to the integral pixel positions offered by the display. However, studies (e.g., Guiard et al., 1999; Bérard et al., 2011) have shown that input devices and the people using them can perform movements at a much smaller scale than the pixel limit of a standard display. But due to the resolution function (Section 3.2.2) of the "pointer" device that outputs integral pixels, much of this resolution is likely lost. The arm being "a multiscale structure *par excellence*" (Guiard et al., 1999), it is capable of controlling a device with various levels of detail. If, by "finely pointing" with the input device, the user can perform fine-grained pointing and dragging actions on the screen, this would represent a highly direct method to increase granularity with low *semantic*, *referential distance*, and *turn-taking*, as well as high *skill reuse*. This is the approach we follow during Part II.

In Chapter 4, we explore this possibility, which we call *subpixel interaction*: the normal behavior of the pointer is unchanged, but actions with very fine granularity are nevertheless possible. We show that it can be achieved if the pointing transfer function takes into account the characteristics of the pointing device, the capabilities of the user, and the granularity requirements for the parameters to be controlled by pointing. We also demonstrate that subpixel interaction enables several interactive situations that previously required explicit changes of zoom level or other less direct procedures.

In Chapter 5, we propose a measure of the human ability to control fine input device movements called *useful resolution*. We show that is is better suited to the requirements of subpixel interaction that other existing concepts. We propose an experimental protocol, and measure the useful resolution of users of a highly sensitive gaming mouse by performing a controlled experiment showing that is depends on movement direction. We also discuss how interactive systems can provide calibration interfaces to preserve *skill/challenge balance*, but also how they can harness fine movements beyond human capabilities in specific situations.

3.4.2. INCREASING THE EXTENT OF DRAGGING ACTIONS WITH EDGE-SCROLLING

Concerning the extent of basic dragging actions, they are often confined to the area where the dragging action originated, and several techniques have taken advantage of the screen regions outside this area to provide scrolling with low *turn-taking*, as activation is often a matter of crossing the edge of the area. These techniques are generally known as autoscroll, or *edge-scrolling* as we will call them[8], and are used daily by millions of desktop computer users. In desktop applications, they are present in various forms that may not rate equally on directness and expressive power, but there is few existing research on the design, use, and evaluation of edge-scrolling, which we study during Part III.

In Chapter 6, we propose a design space of edge-scrolling techniques based on the existing literature and try to determine how existing designs fit into it, and how they are used and perceived by users. We use the design space to analyze 33 existing implementations with a novel reverse-engineering method, and confirm that there exist substantial variations in their transfer functions. We also conduct an online survey, with results showing that edge-scrolling is widely-used, and that common usability problems can be traced back to specific dimensions of the design space.

In Chapter 7, we focus on the generative and evaluative aspects of the design space of edge-scrolling. We propose two novel techniques, *push-edge* and *slide-edge scrolling*, that guarantee a constant expressive power—unlike existing techniques that also have higher *referential distance* and lower *skill reuse*—and that outperform a standard design in a text selection task. We also evaluate several existing edge-scrolling designs and identify specific design choices that impact performance. In addition, based on the results of our studies, we propose other evaluation criteria and general design recommendations.

---

[8]Autoscroll is a more common name, but it implies rate-based scrolling, and also refers to a scrolling functionality activated by middle-clicking on some mice (Zhai et al., 1997b).

3.4.3. INCREASING THE DIMENSIONALITY OF MUSICAL CONTROL

Concerning dimensionality, we have seen that the most obvious method to increase it is the addition of adequate non-standard input devices that provide new input degrees of freedom. This method is used in the context of 2D and 3D spatial control, as well as in the context of musical control. However, we have also seen that the creative combination of standard input devices might be a better solution for our requirements. We explore high-dimensional musical control using standard input in more detail in Part IV.

In Chapter 8, we discuss several considerations, including *skill reuse*, for the design of "laptop musical instruments" that enable direct musical expression using standard laptop input. We describe the design and implementation of three such instruments that allow the control of polyphonic piano, wind instrument, and modern drum kit simulations, respectively. We also describe how we successfully used these instruments in various musical situations, such as studio recording and live performance. In addition, we show how the methods developed in this project can inspire the design of direct interaction techniques in other contexts than musical performance.

Increasing granularity with subpixel interaction

Leveraging input and human resolution with subpixel interaction

In the first part, we outlined ways for an interactive system to support direct interaction, in the three senses of the term (Chapter 2), and to support the user's expressiveness on standard desktops and laptops (Chapter 3). In particular, we identified three areas of interest for extending a system's expressive power: increasing the granularity of basic pointing and dragging actions, increasing the extent of dragging actions, and increasing the dimensionality of musical control. The present part will focus on the first area.

Over the last thirty years, computer users have largely benefited from tremendous increases in processing power, memory, and storage capacity. Graphical interfaces have played a crucial part in the useful appropriation of these resources, making it possible to manipulate ever more complex data. However, while processing, storage, and memory capabilities have been subject to more than a thousand or million-fold increase, the situation is quite different for pointing and display devices (Figure 4.1, top), notably because they are subject to human limitations. For instance, most modern commercial displays still have a pixel density lower than 150 PPI and cannot display more than 3.5 megapixels. Even with the 15-megapixel high-density displays now available at an affordable price, this only represents less than hundred-fold increase compared to the original Apple Macintosh.

As data density keeps increasing, it often exceeds the density of the display, and direct manipulation becomes a problem. There is a myriad of examples where there are not enough pixels to provide the required granularity: resizing calendar events to minute resolution (e.g., when booking flight departures or tracking billing time), navigating a video using a timeline slider, performing fine vector drawing tasks (e.g., drawing a structural wall to centimeter resolution or aligning objects in vertex-dense areas), cropping a large image, or picking an exact location on a map. As we have seen in the previous chapter, existing solutions involve a change of pointer mapping or a scale adjustment, but these approaches take for granted the curious way in which pointing device movements are mapped to data.

Current operating systems receive motion increments (in counts) from the pointing device and transform them, using a transfer function, into pointer movements that are subsequently routed to applications. The problem is that this mapping serves as the basis for all graphical interactions, and since movements are ultimately measured in pixels, they are only as fine-grained as the display density allows. So, a pointing device is not really how we interact with data: it is a device
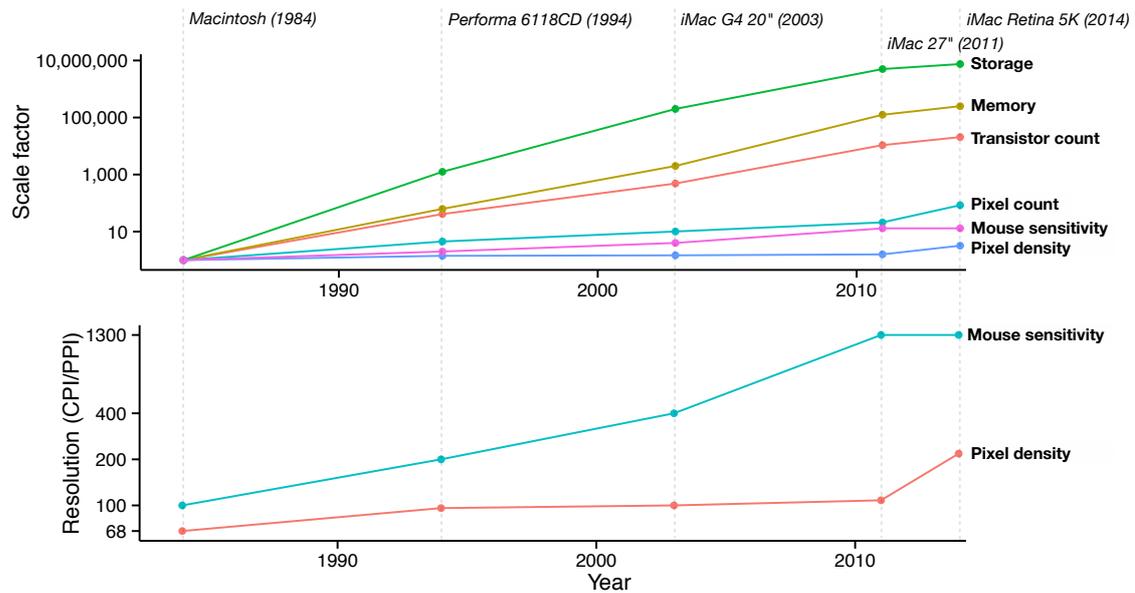
Figure 4.1: Thirty years of evolution of consumer PC hardware, from 1984 to 2014: hardware specifications compared to the original Macintosh (top); comparison of mice sensitivity (CPI) and display density (PPI) between four computers (bottom).

through which we interact with the on-screen pointer, through which we interact with data. This mapping imposes whole-pixel input as a hard constraint, and ignores the high sensitivity of modern pointing devices, as well as our fine motor skills.

This kind of mapping seemed reasonable thirty years ago, when the pointing devices of early personal computers had a sensitivity commensurate with the density of displays. But today's pointing devices largely exceed the density of the best displays available (Figure 4.1, bottom), reaching sensitivities between 400 to 10000 CPI. These are more suitable for capturing our fine motor capabilities, which surpass the normal range of human vision (Langolf et al., 1976; Starkes et al., 1993). What we propose are subtle, but critical modifications to pointing transfer functions that enable *subpixel interaction*, the opportunity to interact "in-between pixels" to attain a level of granularity matching data density, without the indirectness of existing solutions.

In this chapter[1], we will first detail notions related to pointing transfer functions and quantization. We will then describe three factors that determine the opportunity for subpixel interaction—device characteristics, human capabilities, and data granularity—and show how pointing transfer functions can support them. We will also detail how subpixel interaction enables the various examples discussed above. Finally, we will outline the limitations of this approach, its integration in existing systems, and general guidelines for the design of subpixel-enabled applications.

4.1. Related work

Limb movements on the pointing device in *motor space* are converted into pointer movements in *display space* by the transfer function, which applies a control-display gain (CD gain) multiplier,

---

[1]The work presented in this chapter has been carried out in collaboration with Nicolas Roussel, Géry Casiez, and Daniel Vogel.

converting the scale of movements from motor to visual scale. CD gain is the ratio between the velocity of the pointer and the velocity of the pointing device. A low gain facilitates fine-grained movements, while a high gain facilitates ample movements. In most modern operating systems, CD gain varies dynamically depending on device velocity (Casiez and Roussel, 2011) thanks to a *pointer acceleration* mechanism: rapid limb movements are magnified in display space, while the precision of slow limb movements is enhanced. According to Casiez et al. (2008), pointer acceleration designs are consistent with the stochastic optimized-submovement model (Meyer et al., 1988, 1990), which decomposes a pointing movement into a primary ballistic submovement at high velocity, covering most of the distance to the target, followed by secondary corrective movements at lower velocities adjusting for likely undershooting or overshooting. The transfer functions used in modern systems have positive effects on pointing performance compared to constant gain functions (Casiez et al., 2008; Casiez and Roussel, 2011).

There are limits to the CD gains that can be used in a transfer function. Casiez et al. (2008) proposed that there is a *usable range* of CD gains, defined by a minimum gain related to the maximum expected target distance and the maximum operating range (the motor space area dedicated to pointing device movement), and a maximum gain related respectively to the sensitivity of the pointing device and the display density, and to the minimum expected target width and a measure of the resolution of the hand and fingers. Nancel (2012) offers an alternative interpretation of the usable range by defining Exp_{output} as the ratio between the maximum expected target distance and the minimum expected target width, giving a measure of the minimum number of positions that a user should be able to move the pointer at in a given interface. He also defines Exp_{input} as the ratio between the maximum operating range and the distance corresponding to the resolution of the hand and fingers, or the device sensitivity if it exceeds that. With this formulation, if $Exp_{\text{input}} \geq Exp_{\text{output}}$, the range of gains between the minimum and maximum value allows both the maximum expected target distance to be spanned in a single movement, and a target with the minimum expected width to be acquired within the combined resolution limits of the user and the device. If $Exp_{\text{input}} < Exp_{\text{output}}$, then the minimum and maximum gain values are outside the usable range. In an earlier effort, Ballagas and Borchers (2006) proposed a method to determine gain values based on the *submovement range*; that is, the maximum number of discrete counts that a user can produce using the input device in a primary submovement, whose duration is defined by reference to the Model Human Processor (Card et al., 1983). The rationale behind submovement range is that the user should be able to position a pointer in the vicinity of any on-screen target at the end of the first submovement, even if further corrective submovements are required later, in accordance to the stochastic optimized-submovement model. Combined together, these works show that CD gain is amenable to three major problems: clutching, limb resolution, and quantization.

When CD gain is below the usable range, the need for repeatedly disengaging, repositioning, and reengaging the device (*clutching*) is increased if the operating range is not large enough. However, clutching might not be as severe an issue as limb resolution and quantization problems.

Simulated inertia and friction can be used to take advantage of clutching by letting the pointer continue its course in a controllable fashion (Beaudouin-Lafon et al., 2014), and a study by Nancel et al. (2015) suggested that clutching is actually favored by trackpad users, is easier to perform, and causes less errors compared to "clutch-less" movements.

Limb resolution problems can also occur due to the fact that modern pointing devices may exceed a user's fine motor skills. In 2011, Nicolas Roussel inventoried the sensitivity of 375 mice sold on Newegg, a popular american online retailer of computer hardware where sensitivity data was provided. The results[2] show that most of them have a sensitivity over 800 CPI, and that top-selling mice at the time had a mean sensitivity of 4000 CPI. Overall, sensitivities ranged from 400 CPI to 10000 CPI, enabling minimal measurable displacements of 0.0635 mm and 0.00254 mm respectively. Whether a user can leverage this level of detail depends on her ability to control fine movements of the input device in motor space. Guiard et al. (1999) found that users can comfortably acquire 0.06 mm targets in motor space (423 CPI), but this is not an upper bound since it was the smallest width they evaluated. More recently, Bérard et al. (2011) defined a *device's human resolution* (DHR) as "the smallest target size that users can acquire with an ordinary amount of effort using a particular device". They found that the DHR, which depends on the input device and the user's sensorimotor capabilities, was between 700 and 1400 CPI. If the CD gain is too high, limb resolution problems might indeed occur because the user cannot control the pointer finely enough.

Finally, a CD gain might so high that the smallest movement makes the pointer jump several targets at a time, making the acquisition of the in-between targets impossible. This is due to the *quantization* inherent in the input device, the display, and the transfer function. A pointing device senses continuous movements and reports them in discrete increments at a given rate, with physical increment length and rate being determined by the sensitivity and update frequency of the device. A transfer function might further quantize pointer movement, making fewer increments, or increase the length of increments using a high gain value. The display also plays a part in quantization, as it is discretizes pointer movements in pixels, whose physical size is determined by the density of the display. All these sources contribute to what Chapuis and Dragicevic (2011) call *pointer quantization*, "the subdivision of cursor movements into discrete movements". They restate the problem of "jumping targets" as issues of *target misalignments* between motor and visual space: targets whose size in display space are not a multiple of the CD gain might actually be larger or smaller in motor space due to the fact that they do not align with possible pointer positions. For example, if the pointer only advances by 8 pixel increments, a 3 pixel target between two pointer positions will have a zero size in motor space and cannot be acquired. But quantization problems cannot be disregarded by considering that realigning targets solves them, because it does not when the space of possible targets is denser than the display: for example, selecting a particular second of a 1.5 hour video on a timeline slider would require it to be 5400 pixels wide, the width of three 1080p displays, so that all possible targets can be realigned

---

[2]`http://libpointing.org/resolutiondata`. Last accessed 06/20/2015.

to pixels. In order to better comprehend the problem of quantization that may occur in subpixel interaction, one needs to consider that the space of possible targets is quantized differently and separate from the motor and visual spaces.

Ballagas and Borchers (2006) proposed that such a *selection space*, composed of *selexels* (atomic selectable units, also having a fixed physical size), indeed exists independently of the motor space in which limb movements are performed and to the visual space in which pointer movements occur. Thus, a transfer function should match the three spaces. They did so by reformulating CD gain as a combination of a control-selection (CS) and a selection-display (SD) gain, making apparent another facet of quantization. On the one hand, the relationship between motor space and selection space exposes the problem described earlier: when the CS gain is lower than 1, moving the device by 1 count spans several selexels, leading to some inaccessible targets. On the other hand, the relationship between selection space and display space exposes an output quantization problem. In most desktop interfaces, the SD gain is equal to 1, meaning that every selexel is assigned to an independent pixel of the display. In some interfaces, not all the pixels are meaningful with respect to selection (for example, a simple menu system arranged in a fixed grid), leading to less selexels than pixels, and a device with limited operating range and sensitivity might be used without problem if SD gain is higher than 1, in which case the pointer is an area cursor in display space (Ballagas and Borchers, 2006). In these situations, every selexel is visible, but this is not the case anymore when the SD gain is lower than 1, and the user requires a secondary source of visual feedback to facilitate selection.

The notion of a selection space is conceptually useful: what is selected or manipulated and what is displayed inhabit two different worlds, even though the principles of direct manipulation make the user believe the contrary. But one cannot assume that the resolution of the selection space is fixed. In a video player, the timeline slider and the volume slider have different underlying models (e.g., the timeline slider is linked to the current position on the video), which have different ranges and accuracy constraints. The position on the video can be set in seconds or frames, while the video volume corresponds to a percentage of the original audio level of the video. The space of selectable values has a different density for each of them.

Most modern pointing devices have a sensitivity larger than the density of a common display, allowing finer movements than the pixel limits imposed by the display. When interacting in the sub-pixel domain, the issues of limb resolution and quantization discussed above in the case of high CD gains might also appear. Therefore, to allow subpixel interaction at the desirable level of detail, output quantization must be avoided by providing a secondary source of feedback, and the pointing transfer function must consider the characteristics of the device, of the user, and of the model underlying the graphical object being manipulated.

4.2. Breaking the pixel barrier

Current systems do not allow subpixel interaction because they fail to incorporate three key parameters in the transfer function. First, the level of detail offered by the device is often ignored:
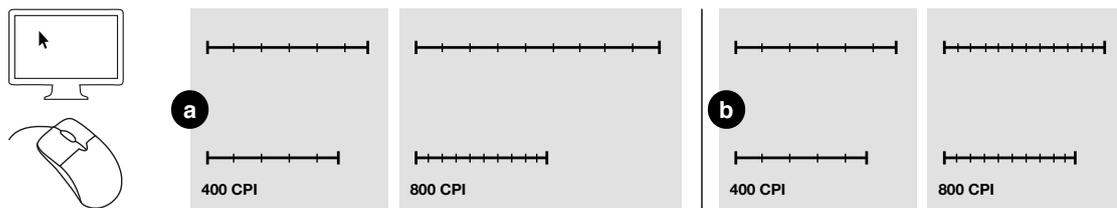
FIGURE 4.2: Effect of changing the sensitivity of the pointing device on the magnitude and resolution of pointer displacements resulting from the same input device displacement in motor space: (a) in current systems, changing from a 400 CPI to a 800 CPI input device increases the distance traveled by the pointer; (b) ideally, a change in device sensitivity should not change pointer displacement but the resolution at which the pointer is positioned.

when a device with higher sensitivity is plugged in, the result is larger pointer movements, not more fine-grained ones for the same physical distance in motor space. In addition, resolution is lost during the conversion between pointing device displacements to pointer displacements. Second, the user's fine motor skills are also ignored: further increases in device sensitivity might be useless beyond a given threshold due to the limited capabilities of a particular user. And third, the granularity of the data model underlying what is graphically manipulated—that is, the density of the selection space—is ignored as well. In this section, we will describe how to take into account these three parameters in order to make subpixel interaction possible.

4.2.1. TAKING THE INPUT DEVICE INTO ACCOUNT

To harness the sensitivity of the input device, the system first needs to know some of its properties related to measurement. Most devices comply with the USB HID specification (DWG, 2001), which allows them to describe the characteristics of the measured values (in device-dependent counts) for each axis, so that the system to which the device is connected can theoretically convert the values in metric units, but few devices provide the necessary information in practice (Casiez and Roussel, 2011). In addition, the pointing transfer functions of most modern operating systems ignore device resolution information anyway (Casiez and Roussel, 2011): Windows relies on hard-wired constants for input device sensitivity and update frequency as well as display density and refresh rate, OS X only takes account of input device sensitivity, and Xorg (used on Linux) only uses input device update frequency while ignoring the rest. Taking the characteristics of the input and display device into account is important to ensure that a physical movement of distance D_{motor} on the pointing device always leads to the a pointer movement of distance D_{visual} on the screen for any pointing device or display. However, in the current state of hardware and device drivers, a higher pointing device sensitivity results in a change of magnitude, not in a more fine-grained movement (Figure 4.2).

Even though device characteristics were not ignored and proper device-independent distances were measured, most of the resolution would be thrown away because the resulting pointer position is determined in integral pixels. The pointing transfer functions of modern operating systems store the fractional part of the resulting pixel displacements as remainders, but only the

integral part is further transmitted to applications (Casiez and Roussel, 2011). Because the display would not be able to reflect the added resolution anyway, this makes sense for pixel-scale motions, but in many cases there are more meaningful positions to be selected than there are pixels. A simple solution would be to expose the fractional part of pointer displacements and reflect it in the pointer position used in applications. Using floating-point pointer coordinates would also greatly simplify the handling of remainders in the implementation of pointing transfer functions.

In summary, we propose that transfer functions take into account device resolution and update frequency characteristics, so that movements in motor space are no longer magnified by input device sensitivity or constrained by display density, and that floating point values resulting from the computation be forwarded *as is* to higher-level code. In theory, this could increase granularity of control up to 100 times, assuming a high-end 10000 CPI mouse and a display density of 100 PPI.

4.2.2. TAKING THE USER INTO ACCOUNT

But having a highly sensitive device does not mean that the user can fully take advantage of it, thus the transfer function must also take into account her capabilities and limitations. They will influence the number of subpixels (S) accessible to the user through the transfer function; that is, the number of practicable subdivisions of a pixel, which establishes a first limit to the increased resolution offered by subpixel interaction. In order to give a working formula for S, a measure of the user's capabilities for finely controlling an input device must be found.

As discussed earlier, the DHR of a mouse is in the range of 700 to 1400 CPI (Bérard et al., 2011). For the touchpad, Bérard and Rochet-Capellan (2012) found a DHR around 150 CPI. However, the DHR only concerns the smallest target size that can be comfortably attained, and says nothing about the distance ϵ_U (in millimeters) of the smallest displacement one can produce with an input device. Chapter 5 will present a methodology to measure the resolution U corresponding to this value (with $\epsilon_U = 25.4/U$), which we call the *useful resolution*. Our results with a 6400 CPI gaming mouse suggest that most users have a useful resolution between 200 and 400 CPI, but our best performing participant had a useful resolution of 1280 CPI: the useful resolution highly varies between individuals. Supposing a pointing device with sensitivity R_{input} (in CPI), the relevant resolution to take into account to include the user's capabilities is R_{human} (in CPI) such that:

$$R_{\text{human}} = \min(U, R_{\text{input}})$$

Considering a display with pixel density R_{display} (in PPI) and the CD gain G currently applied by the pointing transfer function, the number of practicable subpixels S will then be:

$$S = \frac{R_{\text{human}}}{G \times R_{\text{display}}} \tag{4.1}$$
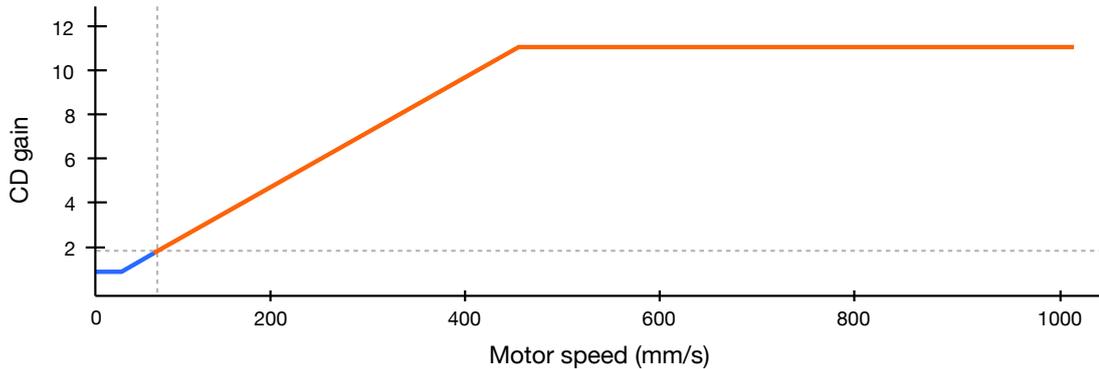
FIGURE 4.3: Subpixel pointing transfer function with parameters similar to typical system functions of modern operating systems, considering a 4000 CPI mouse sampled at 500 Hz and a 90 PPI display refreshed at 60 Hz. The blue part corresponds to the gain levels that generate subpixel motion.

S is actually the number of ϵ_U required in motor space to move from one pixel to another. Also, note that the highest S value S_{max} corresponds to the lowest gain level G_{min} of the transfer function.

What resolution improvement would this represent in a modern system for a user with very good motor skills? Typical pointing transfer functions from modern operating systems produce CD gains between 0.8 and 4.1 for motor speeds under 2.5 cm/s, and then smoothly transition to higher gains at higher speeds (Casiez and Roussel, 2011). Assuming a transfer function producing similar gain values, starting with $G_{min} = 1$ (Figure 4.3), a 4000 CPI mouse with a 500 Hz update frequency, a 90 PPI display with a 60 Hz refresh rate, and a useful resolution of 1000 CPI, then the user would have access to $S_{max} \approx 11$ subpixels. Clearly, this might not be enough in many cases.

4.2.3. TAKING THE SPACE OF SELECTABLE VALUES INTO ACCOUNT

As discussed earlier, pointing or manipulating a graphical object actually selects a value in an area of a selection space, which exposes the model underlying the graphical object. Models can represent discrete (e.g., integers, video frames) and continuous elements (e.g., floating point values). The range of elements can be bounded (e.g., choosing a frame from a video) or unbounded (e.g., specifying the width of a polygon in a vector drawing application). In practice, unbounded models can be considered bounded given reasonable minimum and maximum values for the task, and continuous models may be considered discrete given a reasonable level of detail for the task. By treating all models as discrete, we can define the cardinality N of a model as the number of elements which users expect to select from.

Supposing that P pixels of display space are allocated to give access to N elements in the model (with $P < N$ in the subpixel case), allowing the selection of any element would require $s = N/P$ subpixels. An optimal CD gain level G_{opt} supporting this can be determined from Equation 4.1:

			G_{opt}	
	Duration	Time for 1 pixel	1 sec. precision	1 frame precision
TV show	52:00	00:09.905	1.121	0.037
Movie	1:30:00	00:17.143	0.648	0.022
Gone with the Wind	3:58:00	00:45.333	0.245	0.008
Bergensbanen	7:14:13	01:22.708	0.134	0.004

315 pixels

FIGURE 4.4: Optimal CD gain values (G_{opt}) for one-second and one-frame resolution navigation in videos of various durations using the slider interface of QuickTime Player. Red G_{opt} values are outside the CD gain range of the typical transfer functions of modern operating systems, and require pointing transfer function adaptation.

$$G_{opt} = \frac{R_{human}}{s \times R_{display}} = \frac{P}{R_{display}} \times \frac{R_{human}}{N} \tag{4.2}$$

Let us consider example G_{opt} values when navigating a video at different levels of detail using the dedicated slider of the QuickTime Player application (Figure 4.4, left) on OS X. The slider has a maximum width of 315 pixels, and the number of elements it gives access to depends on the duration of the video and the desired level of detail. One pixel of the slider spans about 10 seconds of a TV show, 17 seconds of a typical movie, 45 seconds of the movie *Gone with the Wind*, and almost one minute and a half for a "slow television" program (*Bergensbanen*). Using the modern system described above, the transfer function shown in Figure 4.3 can only provide the G_{opt} value needed for one-second resolution navigation with the TV show, but not with other videos (Figure 4.4, right). Achieving frame resolution is even more challenging: assuming a frame rate of 30 fps, G_{opt} values would be much lower. In order to approach these optimal gain levels, we need a strategy for pointing transfer function adaptation.

The proposed adaptation should not interfere with normal pointer behavior in the pixel domain, thus it should only produce the desired G_{opt} at low speeds. Specifically, given F_{input} the update frequency of the pointing device, we calculate V_{min}, the minimum speed (in mm/s) on which the transfer function operates, and V_{human} (in mm/s), the speed associated with R_{human}:

$$V_{min} = \frac{25.4}{R_{input}} \times F_{input}$$

$$V_{human} = \frac{25.4}{R_{human}} \times F_{input}$$

As the user will not be able to move the device at speed V_{min}, the G_{opt} gain value is maintained for all speeds under V_{human}. In order to maintain the normal behavior of the pointer around the point where device motions generate 1 pixel displacements (V_{pix}, with corresponding gain G_{pix}), the adapted transfer function interpolates in the gain range $[G_{opt}; G_{pix}]$ between speeds V_{human}
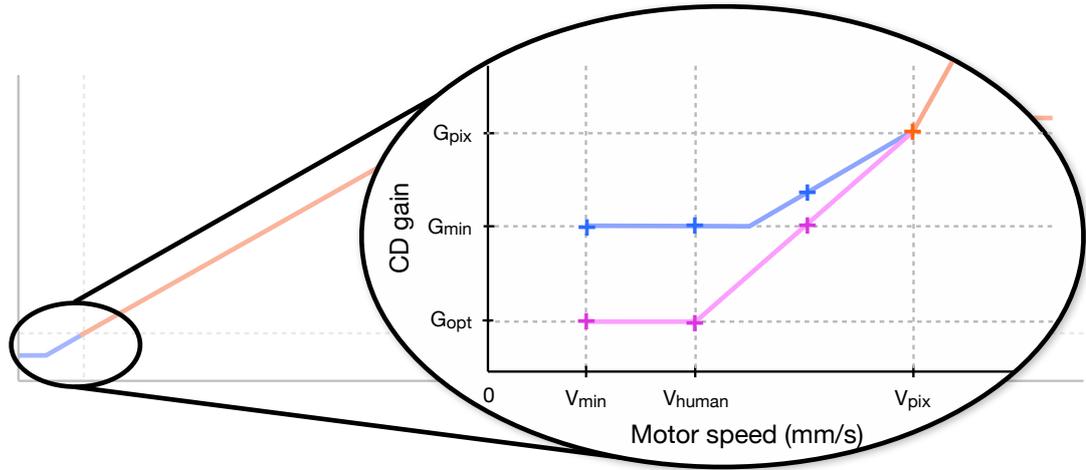
FIGURE 4.5: Closeup on the low-speed domain of the pointing transfer function of Figure 4.3 adapted to allow the selection of any element of a given model. The blue and red parts represent the original gains, and the purple part represents adapted gains. Crosses represent the discrete points of the transfer function.

and V_{pix}. Note that, unlike V_{min} and V_{human}, the values of G_{min}, V_{pix}, and G_{pix} depend on the original pointing transfer function.

Figure 4.5 shows the low-speed domain of a pointing transfer function to be adapted. The red line represents the points of the original function that produce pixel motions, and the blue one represents the part producing subpixel motions. The purple line illustrates a possible interpolation between points (V_{human}, G_{opt}) and (V_{pix}, G_{pix}). The crosses represent the discrete points of the transfer function, constrained by F_{input}. The interpolation consists in the following number of discrete steps:

$$n_{steps} = \frac{V_{pix} - V_{human}}{V_{min}} \tag{4.3}$$

There must be enough steps to keep a reasonable distance between interpolated gain levels to reduce the risk of overshooting, but at the same time, the gain values corresponding to pixel motions should not be altered since this would result in a possibly perceivable modification of pointer behavior. Note that in the worst case, there may be no subpixel speed/gain combination in the original function.

4.3. Application examples

Because current operating systems incorrectly handle input device and display resolutions and refresh rates, they cannot support subpixel interaction. Therefore, we created an environment with a dedicated pointing system (Figure 4.6) in order to illustrate the benefits of subpixel
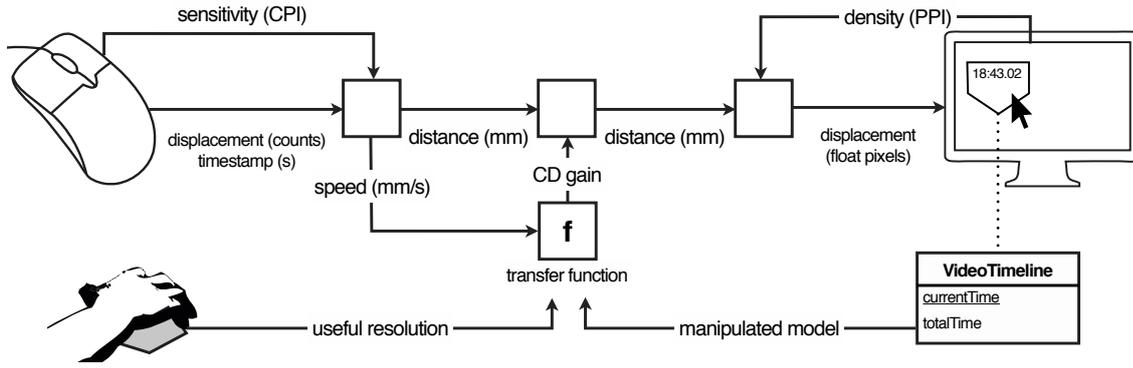
FIGURE 4.6: Processing of pointing input events in a subpixel-enabled pointing system.

interaction. This environment was written in C++ using the cross-platform Qt library, and running custom HTML5 applications using the WebKit web engine. In addition, the libpointing (Casiez and Roussel, 2011) library provided information on the hardware characteristics of input and display devices, as well as raw input events.

The environment ignored and blocked system pointer events, and its dedicated pointing system instead processed raw input events to change the floating point coordinates of a custom on-screen pointer through a transfer function similar to the one shown on Figure 4.3 and formally described in the following.

Supposing that $F(dX, dt) = dX \times F_G(dX, dt)$ is the transfer function used to control the pointer, Equation 4.4 describes F_G, a unitless gain function producing values between $g_{min} = 1$ and $g_{max} = 10$, where dX represents a relative distance measured in motor space (in millimeters), dt is the elapsed time since the last input event (in seconds), $v = dX/dt$, and v_1 and v_2 are equal to 150 mm/s and 500 mm/s respectively.

$$
F_G(dX, dt) = \begin{cases} g_{min} & : v \leq v_1 \\ g_{min} + (g_{max} - g_{min}) \times \frac{v - v_1}{v_2 - v_1} & : v \in]v_1; v_2[\\ g_{max} & : v \geq v_2 \end{cases} \tag{4.4}
$$

When the user manipulates an object, the system first determines whether more resolution was required by computing G_{opt} as defined by Equation 4.2. If G_{opt} was below G_{min}, then the transfer function is adapted to provide the desired resolution. In this case, instead of F, the system uses the function $H(dX, dt) = dX \times H_G(dX, dT)$, where H_G is described by Equation 4.6.

$$H_G(dX, dt) = \begin{cases} G_{\text{opt}} & : v \leq V_{\text{human}} \\ F_G(dX, dt) & : v > V_{\text{pix}} \\ (1-q) \times G_{\text{opt}} + q \times F_G(dX, dt) & : v \geq v_2 \end{cases} \quad (4.5)$$

$$q = \frac{v - V_{\text{human}}}{V_{\text{pix}} - V_{\text{human}}} \quad (4.6)$$

We used a 22" Dell 2208WFP display set to its native resolution of 1680×1050 pixels (90 PPI) at 60 Hz. Our input device was a Microsoft Sidewinder X8 mouse with a 500 Hz polling rate (F_{input}) and a 4000 CPI resolution, its maximum configurable value, leading to a V_{min} of 3.2 mm/s. In our examples, we fixed the useful resolution U to a middle-ground 300 CPI, which, using our particular transfer functions, led to a maximum number of 3 subpixels (S_{max}) without transfer function adaptation, and a V_{human} of 42.3 mm/s. The other parameters were as follows:

$$G_{\text{pix}} = 1.0$$

$$V_{\text{pix}} = \frac{25.4}{R_{\text{display}}} \times \frac{1.0}{G_{\text{pix}}} \times F_{\text{input}} = 141.1 mm/s$$

Therefore, in this configuration, approximately 44 discrete speed values ($V_{\text{pix}}/V_{\text{min}}$) were available below V_{pix} for subpixel interaction, and $n_{\text{steps}} = 30$ steps for transfer function blending.

We will now describe how subpixel interaction benefits several example tasks: editing calendar events, selecting a frame in a video, positioning a wall in a room plan using a computer-aided design software, cropping a high-resolution image to given dimensions, and placing a landmark on a map.

4.3.1. EDITING CALENDAR EVENTS

Standard calendar interfaces display daily and weekly views where events can be entered by direct manipulation. The user first drags a vertical line between the approximate start and end times of the event in the relevant column (day of the week) to create a rectangle representing the event. Then, she fills in text boxes arranged inside a dedicated property box to enter the exact start and end times, as well as the name of the event. Modifying start and end times afterwards requires a similar procedure. Many events, such as flight or train departures, require minute resolution and could not be specified exactly by direct manipulation anyway, because the number of pixels required for minute resolution would exceed the height of the view: 720 pixels are required for a 12-hour view, and 1080 pixels are required for 18 hours.

Usually, the user or the designer might have to trade off event visibility (the number of hours that can be displayed at once) with manipulation resolution. But with subpixel interaction,
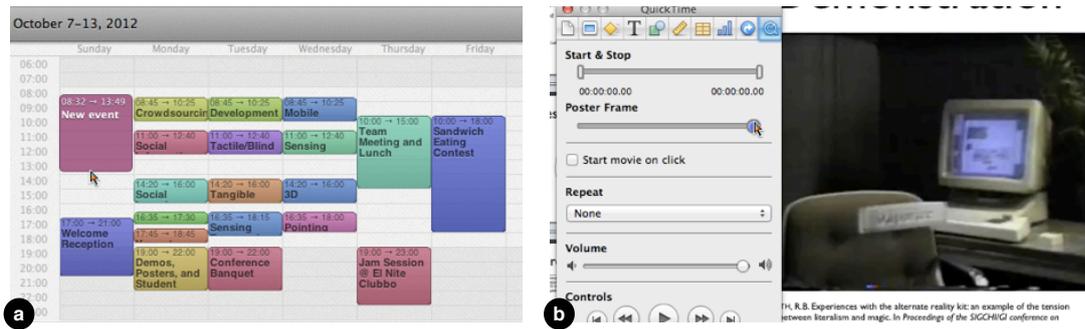
FIGURE 4.7: Examples of situations where subpixel interaction is beneficial: (a) editing calendar events with minute resolution in a weekly view, and (b) selecting a poster frame from a video while preparing a slide deck.

a 18-hour view can easily fit in a 400 pixel view and allow minute resolution, allowing event creation and edition with less turn-taking. A minute occupying 2.7 subpixels in this view, it is within reach of human resolution without even requiring transfer function adaptation. In our implementation of a calendar application, 17 hours of a day are displayed vertically in a 272 pixel view (Figure 4.7a), yielding 16 pixels per hour (3.75 subpixels per minute). Secondary feedback is provided by showing the start and end times of each event. Subpixel interaction could even allow interaction with individual events with minute resolution in monthly views.

4.3.2. SELECTING A FRAME IN A VIDEO

The video inspector interface in Apple's Keynote '09 presentation software provides a 198 pixel wide slider to select a single "poster frame" from a video that is presented before the video starts, and that will replace the video in printed versions of the slide deck. The slider cannot be used to select an exact frame if the video is longer than 8 seconds, at 24 frames per second. With a video length of 15 minutes and 58 seconds, at 30 frames per second, one pixel of the slider corresponds to 28740/198 = 145.1 frames, or 4.8 seconds in the video.

Contrary to the original interface, our replication of the same video inspector (Figure 4.7b) allows frame-level resolution. The pointing transfer function is adapted when the user drags the slider to ensure that any of the 146 subpixels can be reached (G_{opt}=0.023). We informally tested this scenario with four participants. All managed to comfortably reach the intended frame, and no one spontaneously noted any change in pointer behavior, in spite of the low value for G_{opt}. Note that selecting one frame in this scenario corresponds to a 14.8 bit pointing task.

4.3.3. COMPUTER-AIDED DESIGN

In computer-aided design (CAD) and vector drawing software, two-dimensional objects are created and manipulated on a virtual sheet that can be explored at different scales. Many tasks require exact specification of object dimensions and placement. Applications already provide detailed feedback on object positions and dimensions in the form of numerical measurements placed near the selected object. For example, consider a room plan where three walls are already
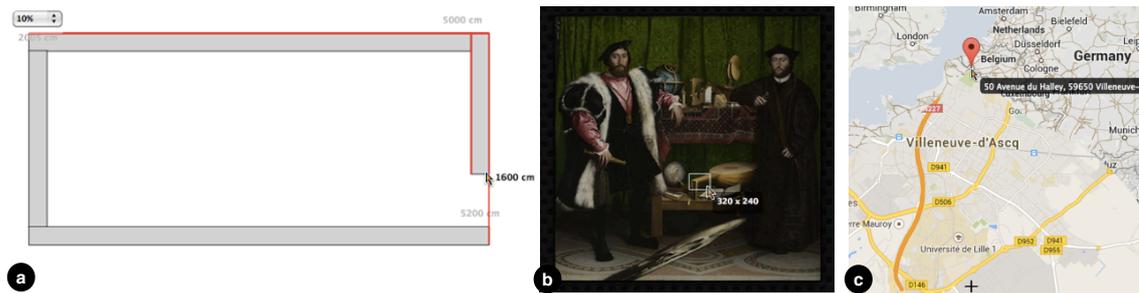
FIGURE 4.8: Examples of situations where subpixel interaction is beneficial (continued): (a) fitting a wall in an architectural CAD application, (b) cropping a high-resolution image to specific dimensions, and (c) accurately placing a landmark on a map of Europe. In examples (b) and (c), the user can benefit from lens-based focus-and-context feedback in addition to subpixel input.

drawn, and a fourth vertical wall with length 2205 cm must be created and aligned exactly with adjacent walls. The whole drawing is viewed at a 1:354 scale, mapping 10 cm to 1 pixel on the screen. Direct manipulation is preferable due to the graphical nature of the task, but to achieve the required level of detail the user must currently resort to text entry, or repeat sequences of zooming in, making adjustments, and zooming out, as many times as necessary for the wall to fit correctly.

We explored this scenario in a small subset of an architectural CAD application (Figure 4.8a). In our case, it is possible to dimension and position the wall with 1 cm resolution at the 1:354 scale, making the interface suitable for multi-point interaction (Shoemaker and Gutwin, 2007): the starting and end points of the wall are both visible (*control point visibility*) at a scale appropriate for the task (*scale adequacy*), and they can be manipulated without the need to adjust the view (*guaranteed manipulability*), while maintaining a view of the entire structure (*visibility of intermediate regions*). At this scale, 10 subpixels are required to perform the task with the required level of detail, involving only a slight adaptation of the pointing transfer function during direct manipulation.

4.3.4. CROPPING A HIGH RESOLUTION IMAGE

Digital cameras and smartphones can capture images exceeding 20 megapixels, yet most computer displays cannot display more than 5 megapixels. For example, consider a 17.8 megapixel image measuring 4125×4125 pixels displayed in a vertically maximized 1600×1600 pixel window on the high-resolution display of a modern 13 inch laptop. To fit the window, it must be scaled to 39%, making one display pixel correspond to 3 image pixels. Consequently, cropping such an image to specific dimensions, say 320 by 240 pixels, usually requires zooming and adjustment sequences similar to the previous example. The problem is worse if the user is a "careful coordinator" (Hutchings and Stasko, 2004) who keeps many windows visible at once. For example, if the image fits in a 300×300 window (7% scale), one display pixel now corresponds to 14 image pixels in each dimension.

We replicated the second case in our environment with an image editor similar to OS X's Preview (Figure 4.8b). To facilitate selection, the software displays the coordinates of the pointer and the size of the selection rectangle in image space in a tooltip following the pointer. With subpixel interaction, selecting a rectangular part at image pixel resolution requires 3 subpixels in the first case and 14 subpixels in the second. Even with transfer function adaptation in the second case (with $G_{opt} = 0.794$ with our configuration), the task can easily be performed. When cropping according to visual details, lens-like feedback would be more appropriate to supplement subpixel input.

4.3.5. PLACING LANDMARKS ON A MAP

Suppose a user plans a road trip across Europe and wants to quickly calculate the length of her itinerary. Knowing the approximate locations of all the stops in her journey, she uses a web map application to place landmarks on each of them. To get a full view, she zooms out the map so that both Lisbon in Portugal and Moscow in Russia are displayed. At this level, 1 pixel represents approximately 3.6 kilometers. To obtain an accurate enough estimate of the length of the itinerary, each stop should be located with a resolution of about 50 meters, which represents 4 pixels at the zoomed-in level that is used to place the landmarks. In a standard map application, the user would have to switch back and forth between both zoom levels.

We tested this scenario by implementing a map application using the data from Google Maps (Figure 4.8c). The map is displayed at the zoomed-out level described earlier (zoom level 5, scale 1:36,978,596[3]). When slow and fine-grained pointer movement is occurring, a tooltip with the exact address appears, and a focus lens is displayed below the pointer, showing the map at the zoomed-in view (zoom level 13, scale 1:144,448) and a cursor at its center. With subpixel interaction, the user controls both the pointer and the focus cursor at the desired resolution of 50 meters. Attaining this level of detail requires 72 practicable subpixels in the zoomed-out context view, leading to a G_{opt} of 0.046. The smallest possible displacement makes the focus cursor jump by 4 pixels in the focus view. While this G_{opt} level is small, it is twice the one used in the video example, which posed no problem to our participants.

4.4. Discussion

By increasing the resolution of pointing actions, subpixel interaction allows to perform several tasks with high granularity without establishing explicit modes, changing the normal behavior, or requiring non-standard input devices: fine-grained positioning of the pointer is performed directly by finely controlling the pointing device. In this section, we will discuss the limitations of subpixel interaction, its implementation in existing operating systems and GUI frameworks, and guidelines for subpixel application designers.

---

[3]Scales provided by Esri, a well-known developer of geographic information system (GIS) software: `http://webhelp.esri.com/arcgisserver/9.3/java/index.htm#designing_overlay_gm_mve.htm`. Last accessed 06/20/2015.
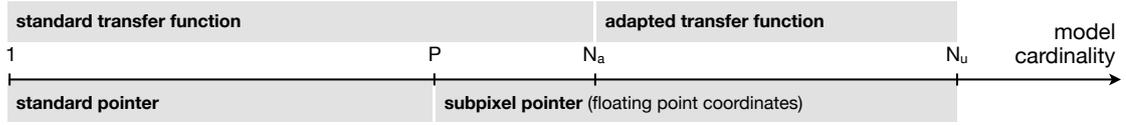
FIGURE 4.9: Zones of applicability for subpixel pointer and adapted transfer function depending on the cardinality of the manipulated model.

4.4.1. DOMAIN OF APPLICABILITY OF SUBPIXEL INTERACTION

While many common situations can benefit from subpixel interaction, its domain of applicability is bounded. There exists a level of detail beyond which the user will inevitably have to use less direct methods that establish an explicit high-resolution mode.

The domain of applicability of subpixel interaction with a given graphical object is determined by the cardinality of the model (N) represented by the position of the object, and by the number of available integer pixel positions (P) the object can take. The domain of applicability also depends on the user's input and output configuration as well as on the pointing transfer function: they all determine the maximum number of practicable subpixels (S_{max}, see Equation 4.1). Once N is above $N_a = S_{max} \times P$, the transfer function must be adapted to guarantee that the user is able to reach all values of the model. We also postulate the existence of another critical value $N_u > N_a$ above which usability issues prevent the user to comfortably reach any value of the model. The possible values of N then define four zones in the domain of applicability of subpixel interaction (Figure 4.9):

$N \leq P$ Every value of the model can be addressed with a standard integer pointer. While it is compatible, subpixel interaction is unnecessary in this case.

$P < N \leq N_a$ Only subpixel interaction can allow the user to address all values of the model by pointing. However, the pointing transfer function need not be modified: there is no change in pointer behavior.

$N_a < N \leq N_u$ The pointing transfer function must be adapted to ensure that the user can address any value of the model by pointing.

$N > N_u$ The user can still address all model values, but controlling the pointer is more difficult as N further increases beyond N_u because the adapted transfer function is not gradual anymore.

Hence, subpixel interaction is applicable when the model cardinality N is in the range $[0; N_u]$. Beyond this range, the adapted transfer function makes subpixel interaction more difficult. While G_{opt} keeps decreasing to accommodate model cardinality, V_{human} and V_{pix} remain the same, as well as the number of discrete speed steps (n_{steps}, see Equation 4.3) available to interpolate between (V_{human}, G_{opt}) and (V_{pix}, G_{pix}). Consequently, the adapted pointing transfer function exhibits a steep curve in the $[V_{human}, V_{pix}]$ range (Figure 4.5), leading to a discontinuity that might be harmful for performance (Casiez et al., 2008). N_u captures this inevitable fact: for

a given user and hardware configuration, there will always be an upper limit on the model cardinality that can be smoothly supported while preserving the standard pointer behavior. Several changes can be made to push N_u further, but they all have limitations. First, increasing the sensitivity and the update frequency of the pointing device reduces V_{min}, therefore increasing n_{steps} and providing more points for interpolation. Second, extending the range of interpolation over V_{pix} leads to a more gradual interpolation, but leads to perceivable modification of normal pointer behavior after some point. And third, increasing P will increase G_{opt}, which also leads to a more gradual interpolation, but controlling the size of on-screen widgets is not always possible.

4.4.2. IMPACT ON OPERATING SYSTEMS AND GUI FRAMEWORKS

The application examples presented in Section 4.3 were implemented in a restricted environment using libpointing (Casiez and Roussel, 2011) to create a transfer function taking into account device characteristics, human limits, and the cardinality of the model underlying the manipulated object. Extending the support for subpixel interaction to the entire operating system would require relatively small, but fundamental changes.

First, system pointing transfer functions must be updated to take into account device resolution and update frequency, so that higher device sensitivities result in higher pointer resolution. This would likely disturb the settings some people are used to, but this would also ensure that a preferred transfer function setting has the same effect independently of the hardware used. Operating systems would also need to issue floating pointer coordinates to the windowing system, which would ultimately dispatch them to applications. Some APIs, like Win32 or that of Java Swing, would then need to be updated so that event loops, methods, callback methods, and so on, support floating point coordinates. Other GUI frameworks that support display resolution independence, like Qt, WPF, or GTK+, already handle floating point coordinates for pointer events, thus making transition to subpixel interaction easier.

Since limb resolution varies between individuals, the operating system should also provide a method to specify the user's current useful resolution. This would require modifications in the device configuration interfaces provided by the operating system. The useful resolution could be part of the input device settings, with a conservative default value. Ideally, a simple calibration step like that discussed in Section 5.4.2 would tune this value to a specific individual.

Finally, GUI frameworks need to provide developers with a way to communicate the cardinality of the model currently manipulated back to the operating system, so that the transfer function can be adapted when a subpixel widget is manipulated. This requires that operating system transfer functions can be changed dynamically, which is already achievable to some extent on some systems. Frameworks with dynamic layout capabilities can also adjust the size of the widgets, making a subpixel-enabled slider smaller without sacrificing detailed control of the underlying data. For example, the preferred size of a slider would depend on N_a, and the minimal size of a widget would depend on N_u.
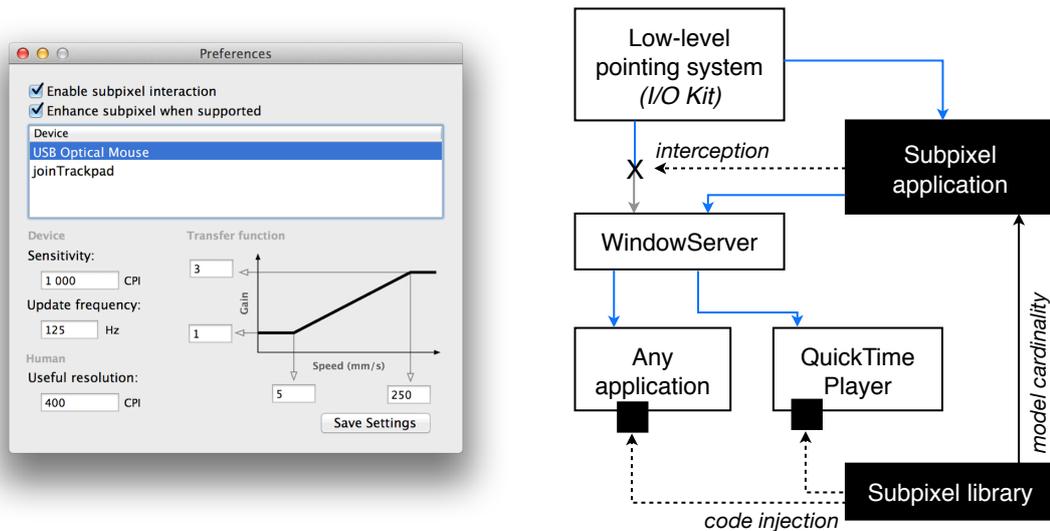
FIGURE 4.10: System-wide implementation of subpixel interaction on OS X. On the left, the device setting interface. On the right, a schematic description of the implementation (see text for details). Blue arrows represent the flow of pointing events.

We have experimented with system-wide implementation on OS X (Figure 4.10). An application developed in Objective-C using the Cocoa framework changed the pointing system so that OS X applications could receive subpixel pointer events. The subpixel application provided a device configuration interface allowing to enter parameters such as device characteristics, useful resolution, and transfer function parameters for each connected device. The application blocked pointing device events, using the Quartz Event Services, before they were taken into account by the window management system (`WindowServer`), which prevented them to be forwarded to applications. Then, the application used the I/O Kit to receive low-level pointing events, applied a custom transfer function, and injected subpixel pointing events into the windowing system, which would be forwarded to applications. All Cocoa applications treating display coordinates as floating point values, they were already compatible with subpixel interaction. However, in order to adapt the transfer function, applications needed to communicate the cardinality of the model underlying the currently manipulated object back to the subpixel application. Therefore, we implemented a subpixel library that modified Cocoa classes corresponding to widgets, such as `NSSlider` (for a slider widget), and other classes corresponding to custom application widgets in the QuickTime Player application, and injected the library into all running applications using Apple's Scripting Additions architecture. With this apparatus, one could for example load a video in QuickTime Player and use its slider to navigate at frame-level resolution.

4.4.3. DESIGNING FOR SUBPIXEL INTERACTION

Based our experience with the application examples presented earlier and the previous discussion, we will summarize three important points that application designers seeking to support subpixel interaction should consider:

Select the appropriate granularity The level of detail made accessible by subpixel interaction
depends on the characteristics of the underlying model, while it solely depended on the
available screen space with pixel-level interaction. It is up to the application designer to
select an appropriate granularity, keeping in mind that there is a limit to what subpixel
interaction and transfer function adaptation can offer. If the application is a simple video
player, one-second resolution probably suffices for most needs.

Make secondary feedback visible Many applications already provide a secondary source of
visual feedback separate from the object currently manipulated by the user. For example,
video players display the current frame and the timestamp of the video, and CAD design
applications display the exact coordinates of the graphical object being manipulated. The
spatial offset between the manipulated object and the secondary feedback source should
be small in order to avoid splitting the user's attention and facilitate the switch between
subpixel and pixel-level interaction.

Sometimes, it is better to use zooming and multi-scale interfaces In all the examples de-
scribed in this chapter, the user interacted with graphical objects at two well-defined scales:
that reflected by the display, and that reflected by the secondary feedback source, where the
maximum accessible resolution could be exploited. However, when fine-grained control
is needed at more than two scales, the user should be able to explicitly control the zoom
level.

4.5. Conclusion

In this chapter, we presented subpixel interaction, a fundamental way to increase the granularity
of basic pointing and dragging actions. We detailed how pointing transfer functions that take
into account device characteristics, human capabilities, and data density can enable subpixel
interaction to attain a level of detail bounded only by the user's fine motor skills. Finally, we
described a successful system-wide implementation of subpixel interaction, and provided three
guidelines for the support of subpixel interaction in applications.

Subpixel interaction contrasts with existing granularity-increasing methods in two points. First,
it does not impose indirectness, because it does not change the way people interact, and it even
remains compatible with techniques that rely on pointer input, such as focus + context lenses.
Second, it questions the artificial input resolution limitations imposed by the density of the
display: pixels may be the limit of what the user can see, but they should not limit what she can
do.

Human limits in small unidirectional mouse movement production

The previous chapter introduced subpixel interaction, a method to increase the granularity of basic pointing and dragging actions beyond the limits of the pixel. We have shown that its domain of applicability is limited in part by the user's ability to finely control a device. In this chapter[1], we contend that existing knowledge on fine motor skills with input devices and methods for measuring them are not adapted to the requirements of transfer functions supporting subpixel interaction, because they do not provide information analog to a device's resolution.

Many users of desktop computers are sensitive to input device considerations. Gamers are certainly among this group, as pointing devices shape a critical part of their gaming experience[2]. High sensitivity is even used as a selling argument by mouse manufacturers, especially for gaming products. But can users actually take advantage of a 10000 CPI mouse? Is the quest for higher mouse sensitivities worth pursuing[3]?

Probably due to the lack of high-resolution sensors until recently, there is a dearth of empirical data on the human ability to finely control these devices and exert small displacements. Related information can be found in Fitts' law studies involving small target widths in motor space (e.g., Langolf et al., 1976; Chapuis and Dragicevic, 2011). Successful acquisitions of targets as small as 0.018 mm using a high-end mouse have been reported (Bérard et al., 2011), but acquiring a small target at a certain distance is not necessarily the same as generating an equivalently small displacement. Thus, it is unclear whether these limits remain valid for tasks other than target acquisition, such as finely adjusting the position of a graphical object.

Our interest here is not in the smallest target a user can acquire using a given device, but in her *useful resolution*; that is, the smallest measurable displacement she can reliably produce with the device. Knowledge on the useful resolution is not only useful to determine the relevance of highly sensitive devices: it is a critical first step to clearly determine the granularity limits of subpixel interaction, termed N_a and N_u in the previous chapter. We will first review the existing

---

[1] The work presented in this chapter has been carried out in collaboration with Géry Casiez and Nicolas Roussel.

[2] For instance, some online communities engage in independent mouse benchmarking (e.g., `http://www.esreality.com/post/1265679/esreality-mousescore-2007/`. Last accessed 06/25/2015.) and many calibration guides have been proposed by their members (e.g., `http://www.overclock.net/t/173255/cs-s-mouse-optimization-guide`. Last accessed 06/25/2015.).

[3] To this question, a high-level executive of a gaming device manufacturing company offers a negative answer in an interview for Ars Technica, but his arguments are misleading because he equates mouse sensitivity with pointer speed (Casiez and Roussel, 2011): `http://arstechnica.com/gaming/2010/02/does-dpi-matter-in-gaming-mice-one-mouse-maker-says-no`. Last accessed 06/25/2015.

literature on small-scale human motor control and clarify the differences with our approach. We will propose an operational definition of the useful resolution and an associated experimental protocol to measure it. We will then report on the results of an experiment based on this protocol, and conclude by discussing the implications of our findings for subpixel interaction.

5.1. Related work

Despite the vast literature on motor control, little is known about the limits of small-scale human movements. We know that highly skilled activities, such as watch-making and microsurgery, operate below the threshold of human detection of kinesthetic feedback, and are essentially controlled by magnified visual feedback only (Jones, 1998). But these activities have been mainly assessed by the time taken to complete a particular procedure (Jones and Lederman, 2006), and the limits of human capabilities in terms of position, movement, and force control have not been extensively studied, mainly due to technological limitations (Jones, 1997). In microsurgery, movements as small as 0.15 mm are quite common, and some procedures like repairing small retinal blood vessels involve movements around 0.01 mm (Charles and Williams, 1989), with microsurgical instruments usually held between the thumb and the first two fingers, in order to reduce tremor (Starkes et al., 1993).

In HCI, most of the work applied from the motor control literature concerns rapid pointing movements and the "speed-accuracy" tradeoff governing them. The most popular model of pointing behavior is Fitts' law (Fitts, 1954; MacKenzie, 1992), which predicts the mean time MT taken to acquire a target with size W at distance D. While many variants of the law exist, the following formula offered by MacKenzie (1992) is the most widely accepted in the field (Guiard and Olafsdottir, 2011): $MT = a + b \log(D/W + 1)$, with a and b being empirically determined constants depending on the task environment, and $\log(D/W + 1)$ being known as the *index of difficulty* of the task (ID, in bits). In the Fitts paradigm, target size and distance (or movement form and scale, Guiard, 2009) are the independent variables manipulated by the experimenter, and are assumed to determine the measured movement time, with a constant 4% error rate constraint. While most pointing studies generally focus on tasks at scales compatible with human vision and with IDs below 9 bits (Guiard et al., 1999), a few researchers have explored the lower limits of applicability of Fitts' law.

In a study of motor performance with different limb segments, Langolf et al. (1976) investigated a reciprocal transfer task in which participants would use a peg attached to a handle, and displace it between two target holes on a plate placed under a microscope. Several plates were tested, providing target distances of 0.25 and 1.27 cm and target widths between 0.076 and 1.07 mm. Acquisition of the smaller targets were successful, and examination of movement trajectories revealed that primary and corrective submovements were performed discontinuously in bursts lasting between 150 ms and 200 ms, with successive corrections guided by visual feedback. However, movements of larger amplitude did not present these discontinuities and displayed a smoother velocity profile, contrasting with intermittent correction models of motor control

hypothesizing discrete submovements like those of Crossman and Goodeve (1983) and Keele (1968). This kind of profile later motivated the development of sophisticated parsing procedures to determine the characteristics of submovements (Meyer et al., 1988), as well as the development of other models of motor control like the multiple-process model of Elliott et al. (2010) assuming that error correction is performed continuously even during the primary movement. Langolf et al. (1976) also noted a tendency to undershoot the target on the primary submovement in the microscopic task, which is also observed at larger scales. To explain this, Elliott et al. (2010) propose that overshooting is more costly than undershooting, due to the longer distance to cover and the effort induced by overcoming the inertia at the reversal of the movement.

Guiard et al. (1999) investigated a discrete pointing task with an ID of 12.2 bits, where participants used a tablet with either a stylus or a puck to simultaneously control two horizontal on-screen sliders representing the same location in motor space. The first "macro" slider represented the entire range of the tablet, with its cursor locating the absolute position of the pointing device on the tablet with poor resolution: due to the high sensitivity of the tablet and the comparatively poor resolution of the display used, one pixel of the slider could represent any of 18 consecutive positions on the tablet. The second "micro" slider represented a detailed enough but incomplete region fixed around the target, with its cursor appearing when the macro cursor is within reach of the target. This configuration provided the required visual feedback to successfully acquire a 0.06 mm target located 28.9 cm away from the starting position with both the stylus and the puck, although movement time was smaller with the stylus.

Chapuis and Dragicevic (2011) also reported successful acquisitions of 0.06 mm targets (with visual magnification) in a discrete pointing task with target distances ranging between 0.12 mm and 0.96 mm, using a mouse and a display with effective resolutions[4] 400 CPI and 100 PPI, respectively. They found that the model of Welford et al. (1969), which incorporates a constant accounting for noise due to hand tremor, accurately modeled small-scale motor performance with visual magnification. However, they recorded no hand tremor with participants holding a mouse, and contended that the tremor constant could account either for constant neuromotor noise, for involuntary movements during mouse presses, or for the stick-slip effect due to contact between the mouse and the surface it lies on, which can induce a high risk of overshooting (Richard and Cutkosky, 2000).

Bérard et al. (2011) specifically investigated the limits of human motor control, asserting that these would likely restrict a user's ability to control highly sensitive input devices. They defined a device's human resolution (DHR) as "the smallest target size that a user can acquire with the device, given an ordinary amount of effort", and measured it by determining a target width, given a fixed target distance, for which Fitts' law would not apply anymore. In other words, the device's human resolution is the target width below which the slope of the Fitts' law regression line is not

---

[4]In their experiment, Chapuis and Dragicevic (2011) used a 2000 CPI mouse and a 200 PPI but divided the sensitivity of the mouse by 5 and the density of the display by 2 in order to approximate a "standard" desktop configuration and avoid issues related to display quality and mouse sensor accuracy.

constant anymore. They performed an experiment to determine the DHR with several devices presenting different characteristics, including a 5700 CPI mouse, used in a discrete pointing task with adequate visual magnification. Regression lines were computed for each sequence of three successive target widths, and the DHR corresponded to the largest target width where a significant difference in regression slope was found. For the mouse, the DHR was between 700 and 1400 CPI, corresponding to target sizes 0.036 mm and 0.018 mm, respectively, but participants still managed to acquire targets as small as 0.0045 mm. Bérard and Rochet-Capellan (2012) proposed an alternative approach based on the paradigm of @schmidt-79motor-output: movement time is imposed (but randomized), and the distance between the target center and the movement endpoint is measured. Using this protocol and a high-resolution optical finger tracking system, they measured a DHR with a touch-sensitive input device around 150 CPI (0.17 mm), and found that targets up to 0.11 mm were successfully acquired given a longer time constraint.

The previous body of works focused on the smallest target sizes one can acquire. However, we contend that movement distance would provide another limit of small-scale controlled movements that has not been previously investigated, and that would be valuable for two reasons. First, a limit based on movement distance would better reflect a measure equivalent to a sensor's resolution, which would be more useful in the context of subpixel interaction. Ristić (1994) defines the resolution of a sensor as "the smallest detectable change of input signal that causes a change in output signal". For a mouse and other pointing devices, it is the smallest reliably measurable displacement, a definition also accepted by Bérard et al. (2011). The corresponding limit for humans would then not be the smallest target size that can be acquired with a device, but rather the smallest displacement one can reliably produce with it. Second, if small-scale submovements are essentially discrete leaps, as observed by Langolf et al. (1976), the lower movement distance limit we are looking for would also represent a target size limit in the worst-case scenario where a submovement falls just before of after the target to be acquired at the very next submovement.

5.2. Defining the useful resolution

The *useful resolution* of a user with a given input device is a resolution (in CPI) corresponding to the smallest displacement (in device counts) the user can reliably produce with the device. In the following, we operationalize this definition of the useful resolution and derive an experimental measurement protocol.

5.2.1. OPERATIONALIZING THE USEFUL RESOLUTION

The useful resolution corresponds to a *reported* distance, thus it depends on the resolution of the input device. Supposing a device with resolution R, the smallest reportable displacement (1 count) is $\epsilon = 25.4/R$ mm, and any reported displacement will be a multiple of this value. Any displacement in the $[n\epsilon; (n+1)\epsilon[$ range will be downsampled to a reported distance of n counts.
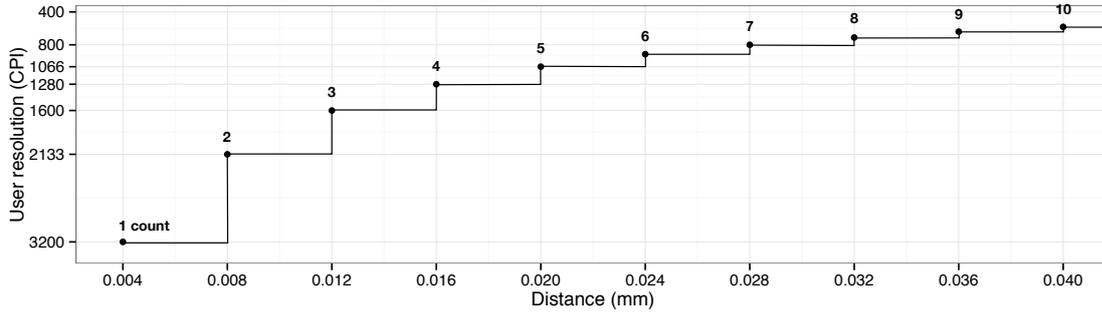
FIGURE 5.1: User resolutions (in CPI) corresponding to distances (in mm), for displacements between 1 and 10 counts reported by a 6400 CPI input device.

Therefore, a 1 count displacement will be reported for any movement whose amplitude is less than 2ϵ mm, which corresponds to half the device resolution (Figure 5.1). Each possible reported distance n (in counts) then corresponds to a *user resolution R_n* (in CPI) such that:

$$R_n = \frac{25.4}{(n+1)\epsilon} = \frac{R}{n+1}$$

The set $\{R_n : n \in \mathbb{N}_{>0}\}$ of discrete user resolutions then contains the useful resolution U, corresponding to distance n_u (in counts).

Following from the definition, U is such that the user cannot produce displacements smaller than or equal to n_u counts using the device in a reliable fashion. Thus, the useful resolution can be measured using the following approach: the user repeatedly performs a unit movement task (see below) to determine how reliably she can control the device such that the accumulated reported distance does not exceed a given maximum distance n_1. This maximum distance gradually decreases ($n_2 > n_3 > \ldots > n_i$), to the point that the user is not able to reliably produce less than n_i counts anymore. The useful resolution is then $R_{n_{i-1}}$.

5.2.2. UNIT MOVEMENT TASK

The *unit movement task* consists in moving the input device in a given direction along a given axis so that it produces at least one report. The task starts and ends with zero velocity and zero acceleration. The only task constraint is that the movement amplitude, as measured by the device, is less than or equal to the given maximum distance n. In other words, to complete the task the participant can always perform a purely ballistic impulse and need not rely on sensory feedback.

Within k task repetitions, the constraint will be respected in p trials, yielding $S_n = p/k$. This success rate therefore indicates how reliably movements of distance n or below can be produced. Introducing a frequency threshold of reliability α, the participant is considered able to reliably produce n counts if $S_n > \alpha$. In addition, she is considered unable to produce n counts when S_n falls below a threshold of unreliability β.

5.2.3. EXPERIMENTAL PROTOCOL

The experimental protocol is summarized as follows:

1. Select a starting maximum distance n (in device counts, $n \geq 1$) under which unit movements are expected to be produced with high reliability.

2. Test n: from k repetitions of a single displacement task with maximum distance n, determine success rate S_n.

3. If $n > 1$ and $S_n > \beta$, select a lower n value and go back to step 2.

4. The useful resolution U is the greatest R_n for which $S_n > \alpha$.

The successive maximum distances that will be tested is determined by the monotonically decreasing sequence $(n_1, \ldots, n_k)$. To select an appropriate sequence one must balance several considerations, such as the total time spent measuring, and the level of detail required.

For example, favoring precision over time and without prior expectation on the useful resolution of a given user with a device, the starting distance (n_1) should be such that $S_{n_1} = 1$, and distance should decrement by 1 count to ensure optimal precision. Supposing that a 1 pixel displacement can be performed reliably (0.254 mm with a 100 PPI display), and supposing a 6400 CPI input device, a possible starting distance may be 64 counts (0.254 mm), and a possible sequence is: $(64, 63, \ldots, 1)$.

The desired degree of reliability, and the corresponding useful resolution, are determined by α, while the β threshold specifies the stopping condition of the protocol: the entire sequence of maximum distances will be tested if $\beta = 0$, whereas the protocol will stop right after finding the first U candidate if $\beta = \alpha$.

5.3. Experiment: measuring the useful resolution

We performed a controlled experiment with a particular high-resolution mouse in order to analyze the effect of movement direction on S_n, to determine a suitable α threshold for subpixel interaction, and to subsequently determine individual useful resolutions. To sample a substantial part of the user resolution space from 100 CPI upwards, we applied the previous experimental protocol using $\beta = 0$ and deliberately not choosing α.

5.3.1. METHOD

The experimental software ran full-screen on a 22" Dell 2208WFP LCD monitor with a 1680×1050 pixel resolution (90 PPI) at 60 Hz. An Intel Core i7 MacBook Air ran the experimental software programmed in C++ and JavaScript using libpointing (Casiez and Roussel, 2011) to bypass the system pointing transfer function and access raw input reports. The input device was a right-handed Razer Imperator 2012 USB corded mouse lying on a varnished plywood desk (Figure 5.2, left), with coefficient of static friction $\mu = 0.17$. Using the Razer driver configuration software, we calibrated the mouse for the desk's surface and adjusted the mouse to a 500 Hz polling rate and a 6400 CPI resolution. We verified the advertised resolution by looking at the raw displacement reported for a reference distance measured with a ruler.
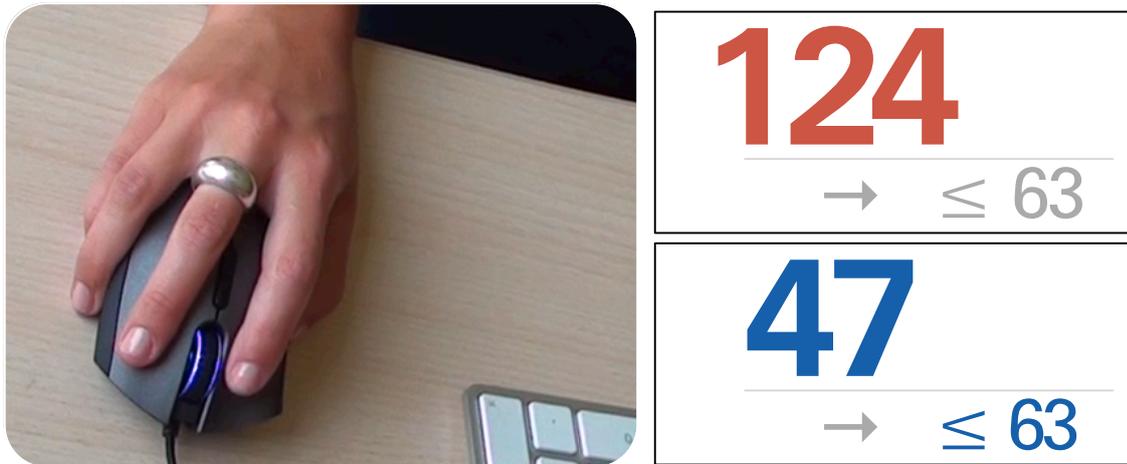
Figure 5.2: Physical layout (left) and experimental display after failed (top-right) and successful (bottom-right) trials.

TASK

Based on the previous protocol, participants had to perform unit movement tasks by moving the mouse along a requested direction for a distance greater than 1 count and lower than a specified maximum value. To present all conditions in a consistent way and avoid issues related to visual acuity, the maximum value and the current movement amplitude were shown as numbers, in counts (Figure 5.2, right). The requested direction was presented as an arrow. Participants were instructed to do their best to stay under the maximum allowed distance, without any time or posture constraint.

A trial started when the first displacement report was received from the mouse, and ended 750 ms after receiving the last displacement report[5]. A failed trial occurred when the cumulated movement amplitude exceeded the maximum value. Any motion reported in the direction opposite to the requested one canceled the ongoing trial, which automatically restarted after a 750 ms pause during which the situation was visually explained. Upon completion, a message indicating whether the trial was successful or not was displayed for a 750 ms pause, after which the instructions for the next trial were automatically presented.

DESIGN

We used a 3×4×8 within-subjects design. The independent variables were movement direction (DIRECTION) and the maximum allowed movement distance in that direction, expressed as a resolution (RESOLUTION) to facilitate further discussion. DIRECTION was evaluated with four levels (EAST, WEST, NORTH, and SOUTH) aligned with the mouse axes. There were eight levels of RESOLUTION (100, 200, 400, 800, 1280, 1600, 2133, and 3200 CPI, see Figure 5.3).

---

[5]The 750 ms timeout was determined after pilot testing. It was long enough to avoid false positives caused by uncontrolled pauses, and short enough to avoid false negatives caused by uncontrolled movements after task completion.

Counts	1	2	3	4	5	6	7	15	31	63
Distance (mm)	.008	.012	.016	.020	.024	.028	.032	.064	.127	.254
Resolution (CPI)	3200	2133.33	1600	1280	1066.6	914.2	800	400	200	100
Tested	●	●	●	●	○	○	●	●	●	●

FIGURE 5.3: Partial view of the user resolution space of the 6400 CPI mouse used in the experiment, with the levels of factor RESOLUTION tested in the experiment and the corresponding maximum distance for each of them.

Participants were given a few minutes to get used to the task before starting the experiment. They subsequently completed three successive BLOCKS. Each BLOCK consisted of 192 trials, broken down into 6 repetitions of the 32 DIRECTION×RESOLUTION combinations. The eight RESOLUTIONS were presented in ascending order. The presentation order for DIRECTION was counterbalanced across participants using a Latin Square design. Participants had to press a key after each series of 6 trials to move to the next one, and were encouraged to take a break before doing so. The experiment lasted approximately 40 minutes. At the end of it, participants were interviewed. They were also videotaped throughout the whole session.

In summary, the experimental design was: 12 participants × 3 BLOCKS × 4 DIRECTIONS × 8 RESOLUTIONS × 6 trials = 6912 total trials.

PARTICIPANTS

Twelve unpaid volunteers (mean age 24.6, SD 2.7, 8 males) served in the experiment. All were right-handed, and seven used a computer more than 6 hours a day. Three were using a mouse in computer games more than 2 hours daily. None suffered from any visuomotor impairment.

5.3.2. RESULTS

The experimental software recorded every trial, however before analyzing the data we removed canceled trials (10.5%), as well as the first trial for each block, as we observed that participants sometimes did not notice the condition changed. In the following, we first analyze success rate, then derive the useful resolution, and finally describe three subjective variables and participant's strategies extracted from post-experiment interviews.

SUCCESS RATE

The dependent variable was the success rate. We transformed the data using an Aligned Rank Transform (Wobbrock et al., 2011a) before running a repeated-measures ANOVA and testing for significant interactions between factors. When the assumption of sphericity was violated, we corrected the degrees of freedom using Greenhouse-Geisser estimates of sphericity. We used Bonferroni correction for pairwise comparisons.

We found a significant effect of BLOCK on success rate ($F_{2,22} = 18.6$, $p < .001$). Pairwise comparisons showed a significant increase of success rate between the first block and the two remaining ones
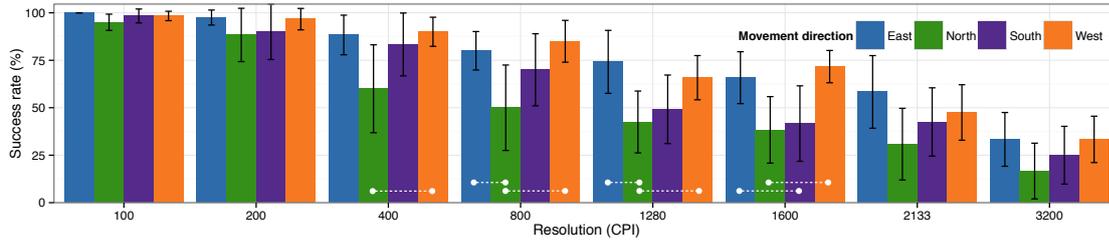
FIGURE 5.4: Mean success rate by DIRECTION for each RESOLUTION. Error bars represent 95% confidence intervals. Connected bars indicate a statistically significant difference.

(BLOCK 1: 54.3%; BLOCK 2: 64.6%, $p = .004$; BLOCK 3: 67.1%, $p < .001$), showing a learning effect. Therefore, we removed the first block from subsequent analyses.

We found a significant main effect of DIRECTION ($F_{3,33} = 11.5$, $p < .001$, $\eta_p^2 = .51$) and of RESOLUTION ($F_{3.5,38.3} = 102.2$, $p < .001$, $\eta_p^2 = .90$) on success rate, as well as a significant DIRECTION×RESOLUTION interaction ($F_{6.3,68.9} = 3.4$, $p = .005$, $\eta_p^2 = .23$, Figure 5.4).

Pairwise comparisons for the DIRECTION×RESOLUTION interaction did not show significant differences in success rate between directions for 100 and 200 CPI, for which participants were able to successfully complete the task 95.5% of the time[6]. However, for 400 CPI, success rate was significantly lower for NORTH than for WEST (60.0% vs. 90.0%, $p = .037$). For 800 CPI, success rate was significantly lower for NORTH (50.0%) than for EAST (80.0%, $p = .046$) and for WEST (85.0%, $p = .010$). For 1280 CPI, it was significantly lower for NORTH (42.5%) than for EAST (74.2%, $p = .008$) and for WEST (65.8%, $p = .031$). For 1600 CPI, success rate was significantly lower for SOUTH (41.7%) than for EAST (65.8%, $p = .030$), and it was significantly lower for NORTH (38.3%) than for WEST (71.7%, $p = .004$). For 2133 and 3200 CPI, we found no significant difference between directions on success rate. In summary, movements towards NORTH are more difficult to perform reliably than horizontal movements. Possible explanations for the effect of direction on the useful resolution will be discussed later on.

USEFUL RESOLUTION

The useful resolution U for a participant is the last resolution for which the success rate was higher than a desired degree of reliability α. The choice of α greatly influences the value and its utility. Figure 5.5 shows how the median useful resolution of all participants in the four DIRECTIONS evolves when α decreases from 100% to 0%: the difference between NORTH and horizontal directions increases as α decreases. A median participant can always ($\alpha = 100\%$) perform small movements with maximum distances corresponding to resolutions between 200 and 400 CPI (between 0.127 and 0.064mm), three quarters of the time ($\alpha = 75\%$) between 200

---

[6]Note that the mean displacement measured in successful trials (12 counts) is equivalent to a 3 pixel pointer displacement on a 96 PPI display with the default OS X pointing transfer function, which assumes a fixed input device sensitivity at 400 CPI for most mice (Casiez and Roussel, 2011). Such a small distance is common in tasks like finely adjusting the position of an object in a vector drawing program.
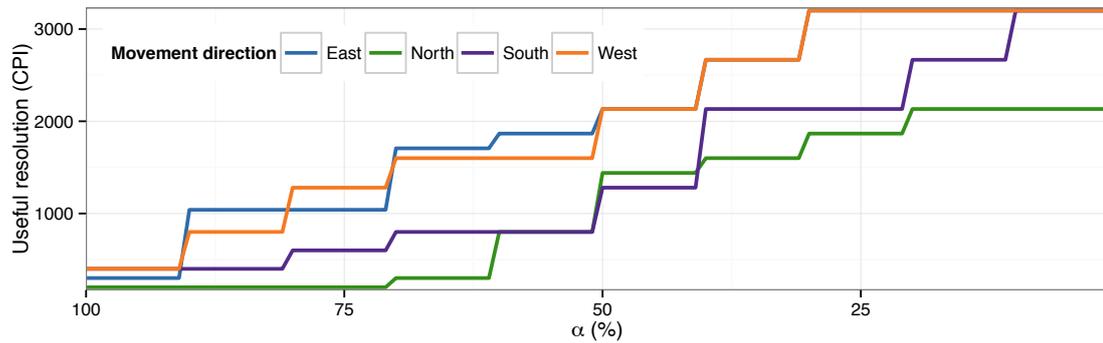
FIGURE 5.5: Median useful resolution (in CPI) by DIRECTION as a function of α.

Participant	1	2	3	4	5	6	7	8	9	10	11	12	median	IQR
EAST	1600	100	200	200	100	400	1280	800	400	2133	200	200	300	200-920
NORTH	1280	200	200	400	100	200	800	100	200	200	100	100	200	100-250
SOUTH	1280	800	100	400	100	400	400	200	100	200	100	400	300	100-400
WEST	1280	200	200	200	100	800	800	800	400	800	200	400	400	200-800
Overall	1280	200	200	400	100	400	400	100	200	200	100	200	200	175-400

FIGURE 5.6: Useful resolution values (in CPI) for each participant, by DIRECTION and overall, with α=95%.

and 800 CPI (between 0.064 and 0.032mm), and half the time ($\alpha=50\%$) between 1280 and 2133 CPI (between 0.020 and 0.012mm). This suggests that if movements as small as 0.012mm can be performed, they may be difficult to perform reliably. The possible uses of difficult small movements in user interfaces will be discussed later.

It is desirable that a default value for α provides good reliability and minimizes the discrepancy between movement directions, making it tenable to assume a single useful resolution value in calculations. The range [90%;100%] for α values exhibited minimal differences in useful resolution between DIRECTIONS (Figure 5.5). In addition, we found no evidence that the effect of DIRECTION on success rate was significant between 100 and 200 CPI (Figure 5.4), for which success rate was 95% on average. Therefore, we chose $\alpha=95\%$ as a default degree of reliability used to compute useful resolutions, on the basis that it should allow rough calculations and design decisions for subpixel interaction.

Individual U values for participants' useful resolution at the $\alpha=95\%$ degree of reliability lie between 100 and 1280 CPI. While one participant, an infrequent computer user, performed remarkably (1280 CPI), the useful resolution of others was noticeably lower (median: 200 CPI, IQR: 175-400, Figure 5.6). Note that only our best performing participant would benefit from a mouse with a sensitivity more than 2500 CPI and similar in every other characteristics to the tested one.

SUBJECTIVE ASSESSMENTS OF DIFFICULTY AND STRATEGIES

In the post-experiment interviews, participants all commented that task difficulty varied with DIRECTION. Consistently with our findings on success rate, most found horizontal movements
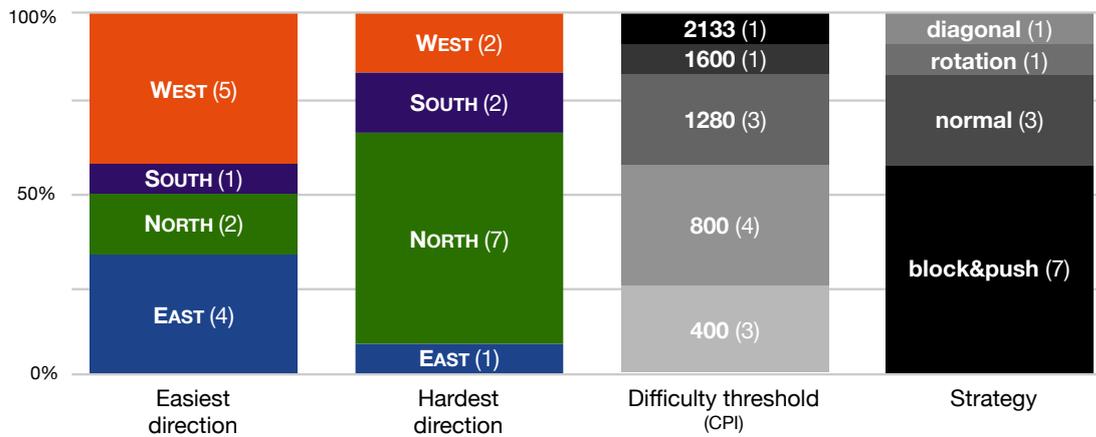
FIGURE 5.7: Summary of subjective assessments of easiest and hardest direction, difficulty threshold (in CPI), and strategy (from left to right).

easier to perform than vertical movements (Figure 5.7, left). In particular, the NORTH direction was found most difficult by more than half of the participants.

We also asked participants to determine a resolution beyond which they felt the task became difficult. Despite varying from 400 to 2133 CPI, the subjective difficulty threshold of each participant (Figure 5.7, middle-right) was consistently higher than or equal to her useful resolution in any direction. In their performance, this corresponded to success rates between 35% and 80% (mean: 56%, SD: 14%), which is considerably below our conservative $\alpha = 95\%$ default for determining the useful resolution.

Participants described four different strategies to complete the tasks (Figure 5.7, right): moving the mouse in the same way as usual tasks (*normal*), performing mouse rotations instead of translations (*rotation*), simultaneously blocking and pushing the mouse (*block&push*), and moving diagonally (*diagonal*). These strategies have different implications on the user's control over the device.

The *normal* strategy (Figure 5.8, left) was used by three participants ($U \in [100;200$ CPI]). They essentially described using the mouse as they would for normal scale pointing tasks. Such tasks are typically accomplished using mainly the wrist or more proximal joints. However, this strategy offers no control on stick-slip phenomena, which play a much greater role at small scales than at the scale of usual pointing. This lack of control likely motivated the other participants to try other strategies.

The *rotation* strategy (Figure 5.8, middle-left) was used by a participant ($U = 100$ CPI) who remarked that orienting the device was less amenable to stick-slip than when moving it as usual. However, the alignment between the center of rotation and the device sensor was difficult to control, sometimes generating reports in the wrong the direction and resulting in more canceled trials.

normal	*rotation*	*block&push*	*diagonal*
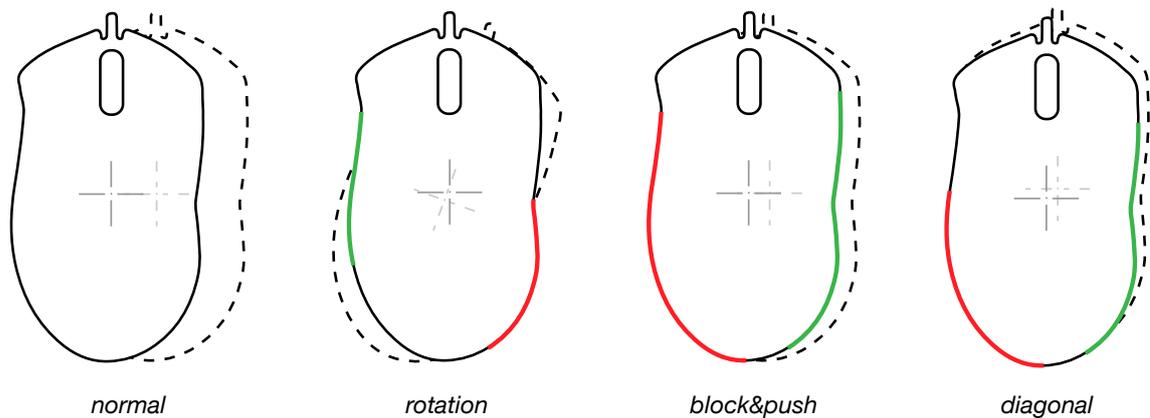
Figure 5.8: Four different strategies described by participants for the unit movement task. Dashed contours represent the mouse position after a movement to the East. Crosshairs show the position of the mouse sensor before and after movement. Possible implementations of these strategies are described with colored lines representing zones where fingers can exert pressure to either move (red) or stabilize (green) the device.

The *block&push* strategy (Figure 5.8, middle-right) was used by more than half of the participants ($U \in [200;400$ CPI]). It consisted in blocking the mouse using a part of the hand (e.g., the little finger, for a movement towards East) while gently pushing it along the requested direction (e.g., the thumb, for East) and slowly releasing the opposing force to achieve fine control. The blocking finger acts as a stabilizer to limit the extent of extraneous movement resulting from the higher force put into the pushing finger in order to overcome the static friction between the mouse and the desk.

Finally, the best performing participant ($U = 1280$ CPI) used the *diagonal* strategy (Figure 5.8, right), a variant of *block&push* where the pushing force was exerted diagonally instead of at a right angle. While this strategy seems effective when movements align with the mouse axes, it is unclear how well users can perform diagonal movements with it.

5.4. Discussion

In summary, when choosing 95% as the threshold for the reliable production of small displacements, most participants had a useful resolution between 200 and 400 CPI with our particular mouse. However, between 400 and 1600 CPI, performance towards North was significantly lower. We will first discuss how the different strategies employed by participants to accomplish the task can explain these results. We will then consider the implications of these results for subpixel interaction.

5.4.1. ANISOTROPY OF THE USEFUL RESOLUTION

Analysis of the Direction×Resolution interaction (Figure 5.4) suggested that movement direction has an influence on success rate, therefore impacting the useful resolution. In particular,

from 400 CPI, small movements on the vertical axis (especially North) were more difficult to control than on the horizontal axis, but the difference disappeared after 1600 CPI.

Participants' accounts of the different strategies and the observation of experiment videos support the hypothesis of Chapuis and Dragicevic (2011) that small-scale movements implied a different motor control strategy: in order to comply with the task demands, most indeed used a different, firmer grip than for usual pointing, and used joints more distal than the wrist to perform their movement. However, the significant difference between North and horizontal directions suggests that these strategies do not support small-scale movement towards North as well as other directions.

In the *block&push* strategy, the mouse is held firmly between the thumb and the small finger, which play a symmetrical role, but horizontal and vertical tasks are performed with different joint movements and are supported in a different way. In the horizontal axis, both fingers oppose through abduction and adduction, allowing a fine control on the pressure applied to both sides of the mouse and inhibiting the stick-slip effect. In the vertical axis, the mouse is instead pushed or pulled through flexion and extension of both fingers. Here, the palm can oppose the mouse in South movements, but not in North ones. Given that most participants used the *block&push* strategy, the absence of a blocking element opposing extraneous movement towards North could explain the significantly lower performance. However, beyond 1600 CPI, participants performed as poorly in North movements as in other directions, suggesting that participants had hit the limits of these fine control strategies.

5.4.2. IMPLICATIONS FOR SUBPIXEL INTERACTION

The useful resolution is an important information for subpixel pointing systems: it determines the user's ability to exploit the sensitivity of the input device and the extent to which the pointing transfer function must be adapted in order to guarantee adequate control over graphical objects.

The range of useful resolutions found in the previous experiment overlaps the density of modern displays, typically between 100 to 225 PPI. With the default transfer functions of modern systems, which produce CD gain values close to 1 at low velocities (Casiez and Roussel, 2011), pixel-level tasks should not be problematic from a useful resolution perspective. Provided that enough feedback is available, more fine-grained tasks are possible, but the pointing transfer function will likely need to be adapted when more than four times the pixel resolution is required.

The anisotropy of the useful resolution also has implications for subpixel interaction. On the one hand, clarifying the influence of the different strategies on unit movement task performance between Directions requires further investigation. But on the other hand, if strategies like *block&push* are as common in actual use as in our experiments, then designs relying on one-dimensional small-scale mouse movements should favor horizontal directions over vertical ones in order to better exploit users' motor capabilities. Fortunately, most graphical interface designs lean more toward, for example, horizontal sliders than vertical ones, due to the widescreen aspect
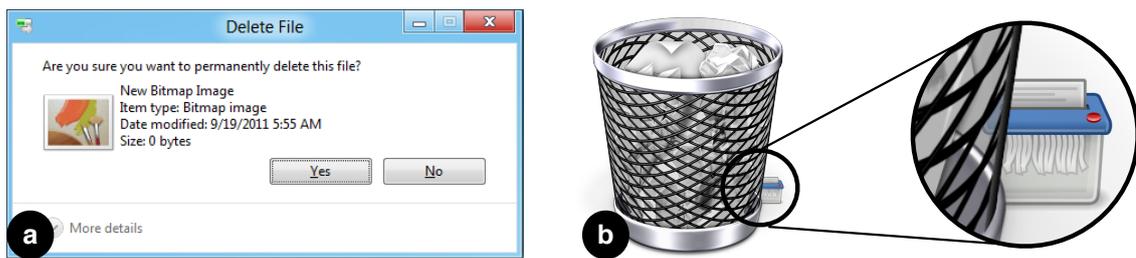
FIGURE 5.9: (a) A dialog box commonly used to ask the user for confirmation after a *permanently delete* command. (b) An alternative design informed by our results where permanent deletion can only be performed using difficult small movements (pressing the red button on the paper shredder).

ratios used in computer displays that govern the available screen real-estate: horizontal sliders are likely to make more values available to user control than vertical ones. The results of the previous experiment highlights that subpixel interaction does not make such considerations totally irrelevant, but emphasizes the human aspects of control.

While our results provide a range of useful resolutions that can be readily used as base values in systems, users must be provided with means to estimate and monitor their useful resolution with a particular device, in order to reach an optimal match between the software, the display, and the input device. The proposed experimental protocol can be used in a calibration tool to quickly estimate the useful resolution. Starting with a R_{n_1} around 200 CPI and using an α value of 95%, two or three iterations through the protocol with a randomized direction and $k = 20$ repetitions per maximum distance should be enough to determine an overall useful resolution.

The evolution of useful resolution over time with practice remains to be studied, but chances are that users might progressively become experts if regularly exposed to subpixel-resolution tasks. In microsurgery, procedures are taught in intensive one-week courses where students' performance improves quickly, although several years of extensive practice are needed to attain expert performance (Starkes et al., 1993). If subpixel interaction is seen as an efficient way to perform very fine-grained tasks, currently inaccessible levels of detail offer an incentive to improve manual dexterity. In addition to the calibration procedure described above, the system should also regularly monitor performance and encourage deliberate practice in order to accompany the user in the path to expertise.

5.4.3. BEYOND THE USEFUL RESOLUTION

The useful resolution sets a lower bound for small device movement production that can be reliably controlled, but users are able to produce even smaller movements (up to 2133 CPI, or 0.012mm, half the time for our median participant) and designers could take advantage of this.

For example, small movements below the useful resolution can be used in purposefully difficult tasks for commands with a high cost of accidental activation. The common characteristics of these commands is that the consequences of a wrong decision are inconvenient, and that they

are difficult or impossible to undo: for example, launching an untrustworthy program with administrator privileges, or permanently deleting an important file. A common technique is to warn the user of the risks and to ask for confirmation using dialog boxes (Figure 5.9a), but it is ineffective due to habituation (Raskin, 2000): users end up ignoring warnings and blindly confirm dialog boxes. Attracting the user's attention in novel ways can help, but habituation is inevitable and confirmation requests must be used only when absolutely necessary (Böhme and Köpsell, 2010).

To bring the user's attention to the potential negative effects of activating a command without her being able to quickly dismiss warnings, the system could make command activation a physically difficult task. If the user's useful resolution with a particular device is known, the system can determine a movement distance where user performance is poor. For example, to confirm a dangerous command, a user might have to perform a unit movement task with such a distance. Subpixel interaction and lens feedback could also be used to augment drag-and-drop interactions. Figure 5.9b shows that to permanently delete a file from the computer, the user must drop it on the small red button of the paper shredder icon hidden behind the usual trash bin icon. Appropriate lens feedback appears when the pointer is close to the shredder icon at low speeds, but the size of the red button in motor space is four times smaller than the useful resolution. Alternatively, a semantic pointing technique (Blanch et al., 2004) could make a *confirm* button smaller than the useful resolution in motor space. With all these techniques, designers can adjust the level of difficulty with respect to expected frequency of use or degree of severity of accidental activation. While the factors governing their discoverability and effectiveness have yet to be investigated, these examples demonstrate how the useful resolution can be used in other contexts than subpixel interaction.

5.5. Conclusion

In this chapter, we introduced the *useful resolution* and defined it as the smallest displacement a user can reliably produce using a particular pointing device. We explained how this notion departs from the existing literature on small-scale target acquisition, and proposed an experimental protocol based on series of increasingly small controlled movements, which is relevant to any pointing device. We reported on an experiment based on this protocol, where participants used a high-end 6400 CPI mouse to perform small controlled movements in four directions. Our results suggest that movement direction influences useful resolution, and show that most participants had a useful resolution between 200 and 400 CPI with the tested device, when choosing 95% as the success rate threshold for the reliable production of small displacements. We also analyzed the different strategies reported by participants to achieve best performance, and conjectured their effect on overcoming stick-slip phenomena. Finally, we discussed the consequences of these findings for subpixel interaction, the role of calibration procedures, and possible other uses of the useful resolution.

This concludes the second part of this dissertation. Our focus so far has been on establishing necessary conditions for subpixel interaction (Chapter 4) and providing an empirical basis for the fine motor aspects of its limitations (this chapter), but many important questions are left unanswered. Are there other factors that influence a user's useful resolution? How does it improve with regular practice? How different are the results and motor strategies with other devices, such as touchpads where users can roll and not only move their fingers? More generally, what are the performance benefits of subpixel interaction in actual settings with realistic scenarios? How does it benefit other granularity-increasing techniques, such as focus + context lenses, and what are the tradeoffs involved in their coordinated use? The need for secondary feedback during subpixel interaction has also been stated, but how should it be designed so as not to introduce indirectness? And finally, what determines N_u, the limit beyond which normal pointer usage is noticeably deteriorated and loses *predictability* ? Further work would help answering these questions and better frame the contribution of subpixel interaction to directness and expressiveness.

Increasing extent with edge-scrolling

Analyzing existing edge-scrolling designs

In this part of the dissertation, we focus on a specific method to increase the extent of dragging actions, *edge-scrolling*, which mainly results in scrolling like other less direct techniques (Section 3.3.4).

When a *document* has more information that can be displayed inside a *viewport* placed on the *display*, the interface must allow scrolling so that the relevant parts of the document can be brought into view (Figure 6.1a). In a typical scenario, the user starts a selection, moves her pointer near the edge of the viewport, and needs to scroll the view to complete her task. Edge-scrolling assists by defining a *scrolling area* relative to the considered *viewport edge* (Figure 6.1a), which once crossed triggers automatic scrolling of the view at a certain speed in the desired direction (Figure 6.1b).

Edge-scrolling is used daily by millions of desktop computer users, and is available in most desktop applications. But even a quick examination of the preinstalled software of an operating system reveals substantial differences between implementations. For example, a Windows 8 user will experience different behaviors when selecting text in Internet Explorer and in Notepad. In the former, the viewport automatically scrolls while the pointer remains below its bottom edge, but the latter additionally requires continual pointer movement for scrolling to occur. Many other behaviors differ both within and between operating systems. These differences are likely to have important implications for user performance, errors, and preferences, but also for directness and expressive power. Yet, despite the ubiquity of edge-scrolling use and the breadth of available implementations, there has been little research to understand how users interact with edge-scrolling and how it should be designed.
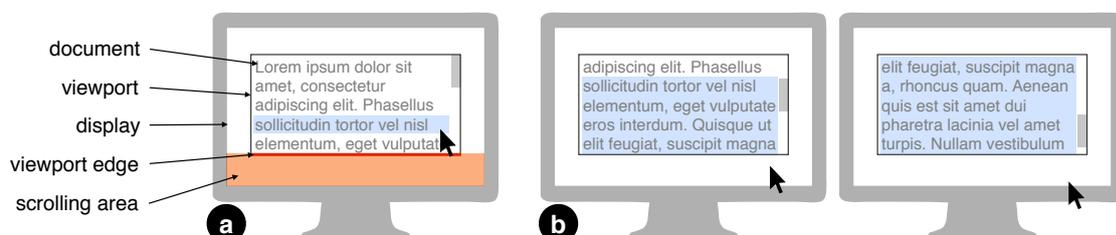
FIGURE 6.1: (a) Edge-scrolling is used in the course of a task (here, selecting text) in order to scroll a document that is displayed inside a viewport placed on the display. The user activates edge-scrolling by entering a scrolling area near the viewport edge. (b) Inside the scrolling area, pointing actions control the pace of scrolling.

In this chapter[1], we attempt to address the following questions: what are the factors influencing the design of edge-scrolling, how are they represented in the wealth of existing implementations, and how are they used and perceived by their users? We will first review the related literature to understand the differences between edge-scrolling and other scrolling techniques. We will then provide a framework for understanding edge-scrolling techniques based on three design dimensions and two task-related factors to consider. Using this framework, we will examine 33 existing edge-scrolling implementations using a novel reverse-engineering method and demonstrate substantial variance in design approaches. We will then report on the results of an interactive online survey, completed by 214 participants, confirming that edge-scrolling is widely used. Results also show that common usability problems can be traced back to the three design dimensions of our framework, but also to a fourth dimension, visual guidance, which we will finally include in a revised version of our framework.

6.1. Related work

Scrolling has been a basic concern in human-computer interaction since the earliest graphical interfaces (Sutherland, 1963), and a wide array of devices and techniques have been used to support it. It has been studied through a variety of empirical models and methods that we will review in the first parts of this section. We will then focus on the implications of using the pointer to drive scrolling, and finally narrow our focus on the existing literature specific to edge-scrolling.

6.1.1. SCROLLING MODELS

The previous work on scrolling performance models has exposed several influential parameters that a scrolling technique should take into consideration. When the user's goal is to reach a distant target, scrolling bears some resemblance to pointing, although the target is not visible until the movement ends. Hinckley et al. (2002) found that despite this difference, Fitts' law predicted scrolling performance well in a reciprocal scrolling task where target position was known ahead of time. To be representative of the variety of scrolling tasks, the distances involved in their study ranged from six to 384 lines of text. The significant interaction between scrolling distance and technique on scrolling time showed that performance differences for two techniques could vary greatly between short and large distances.

Scrolling is a basic operation frequently used together with pointing in higher-level tasks, such as selecting several paragraphs of text, which require both scrolling and pointing. Based on the global precedence principle found in human motor and perceptual systems, Guiard et al. (2004) argued that scrolling and pointing can be thought of as a high-level pointing task, with a *view pointing* phase (scrolling) where the viewport is the cursor and is scrolled until it intersects the target within the coordinate system of the document, followed by a *cursor pointing* phase

---

[1]The work presented in this chapter has been carried out in collaboration with Sylvain Malacria, Philip Quinn, Nicolas Roussel, Andy Cockburn and Géry Casiez.

(pointing) where the actual pointer acquires the target in view within the coordinate system of the display. They proposed that scrolling time has a Fitts-like logarithmic relationship to the target distance and the width of both the target and the viewport.

Andersen (2005) questioned the applicability of Fitts' law to scrolling tasks where target location was unknown beforehand, such as when searching an item in an unsorted list. Andersen argued that in this case performance was more likely limited by visual scanning rather than motor capacity, and proposed that scrolling time was a linear function of the distance to the target. This model was verified with a task similar to that of Hinckley et al. (2002) and predicted performance better than Fitts' Law.

Cockburn and Gutwin (2009) confirmed and integrated both the logarithmic and the linear model for selection tasks in scrolling lists. Their hypothesis was that prior spatial knowledge of the target, if available, enabled the user to prepare and perform a ballistic impulse towards the target, making scrolling time a logarithmic function of the number of list elements, while the need for visual feedback when searching for an a-priori unknown element limited the pace of scrolling and resulted in a linear relationship between scrolling time and the number of elements among which to perform visual search.

However, some scrolling tasks are more complex because there are sometimes several targets to acquire serially, or because knowledge of target location could change during the task. Based on the insight that the information necessary to conclude a scrolling task is spread differently in documents, Guiard et al. (2007) proposed to quantify the distribution of information in a document as a new independent variable, the *degree of goal directedness* (DGD), that can influence the performance difference between two techniques. However, because edge-scrolling is tightly coupled with pointing and dragging, we expect that real-world tasks where edge-scrolling is used will exhibit a smaller range of DGDs than with more general scrolling techniques.

Scrolling performance may also depend on viewport geometry. The time needed to scroll towards a known target on a touchscreen has been shown to depend not only on target size and distance, but also on the size of the viewport (Zhao et al., 2014), due to more frequent clutching when the viewport is small compared to the size of the document. Whether this also applies to non-collocated, indirect input is debatable. Clutching in indirect input depends strictly on the available motor space area around the device, not on the display. Cockburn and Gutwin (2009) hypothesized that viewport size may not influence performance because user focus for visual feedback was essentially on one point. In addition, Guiard et al. (2004) found that view size did not affect performance when above 40 pixels, but they used a multi-scale navigation technique combining scrolling and zooming, whereas edge-scrolling is primarily fixed-scale.

Thus, scrolling performance is expected to vary with target distance and size, and other task-related factors, such as DGD; as well as with external factors, such as viewport geometry. In addition, the particular scrolling technique that is provided by the system and the device with which is is used certainly have a great influence.

6.1.2. SCROLLING TECHNIQUES AND TRANSFER FUNCTIONS

Scrolling techniques have been proposed that leverage all sorts of input modalities. Scrolling input has been integrated into existing devices that are frequently used, such as the keyboard (McLoone et al., 2003), the touchpad (Arthur et al., 2008; Bial et al., 2010), or the mouse (Ohno et al., 1985; Zhai et al., 1997a; Hinckley and Sinclair, 1999). Techniques based on indirect pointing, like the scrollbar or middle-click rate-based scrolling (Zhai et al., 1997b), are commonly used in desktop systems. But scrolling techniques based on direct touch input have also been proposed and studied in the context of tabletop and mobile devices (Aliakseyeu et al., 2008), and some of them based on flicking gestures and simulated inertia are in wide use on today's mobile systems (Quinn et al., 2013). Yet other forms of input for scrolling include gaze (Kumar et al., 2007), the position of a mobile device in three-dimensional space (Mehra et al., 2006), or its orientation (Rekimoto, 1996).

A common element to all scrolling techniques is the *transfer function* that governs scrolling control by mapping dedicated or pointing input either to the absolute position of the viewport in document coordinates (*absolute position control*), to a displacement of the viewport (*relative position control*), or to the scrolling velocity (*rate control*). Each type of mapping has found its use in scrolling techniques, and each has its relative merits[2], but the study and comparison of scrolling transfer functions ultimately requires methods for extracting and replicating their behaviors.

Quinn et al. (2012) observed that the reproducibility of empirical studies of scrolling performance was compromised due to a lack of transfer function details. They also described a method to reverse-engineer the behavior of external scrolling devices connected through USB. Subsequent work extended the methodology to touch scrolling devices, using a robotic arm to precisely control gestural input (Quinn et al., 2013). Reverse-engineering methods could prove helpful in precisely describing existing edge-scrolling designs, providing a basis upon which their features can be studied.

The previous methods have been designed for analyzing dedicated scrolling devices and touch-based techniques, but they do not extend to edge-scrolling techniques on desktop applications, which are controlled using indirect pointing input, and may have characteristic features to which the previous methods are oblivious. Moreover, the logging strategy in these methods are not always suitable to study edge-scrolling. Quinn et al. (2012) logged the behavior of device drivers by using low-level event APIs, and Quinn et al. (2013) used custom programs instrumenting specific widgets. Other methods are needed to study the viewports in closed-source applications, as edge-scrolling is implemented at the application level and is likely to vary between applications.

Indeed, there are different levels of developer guidance on how to implement edge-scrolling in custom widgets. Java's *Swing* documents how to comply with the default toolkit behavior

---

[2]For a more detailed review of scrolling transfer functions, see Section 7.1.

in the `setAutoscrolls` method of class `JComponent`[3] and in the `Autoscroll` interface[4]. In Cocoa applications, edge-scrolling is automatically present in any view placed inside a `NSScroll-lView`, but to be available during drag-and-drop it must be implemented in an ad-hoc fashion[5]. Other toolkits like the Windows Presentation Foundation (WPF), Qt, or GTK+, provide no recommendation or basic behavior to developers.

6.1.3. POINTING-BASED SCROLLING TECHNIQUES

A significant portion of existing scrolling techniques use a dedicated input stream distinct from pointing input: for example, scrolling is often performed by using the scrollwheel on desktop computers. These techniques are fundamentally different from edge-scrolling, which shares the pointing input stream and uses mode delimitation to switch between pointing and edge-scroll modes. Studies have suggested that dual-stream input eliminates costs related to sequential operation and mode switching in single-stream input (Buxton and Myers, 1986; Buxton, 1987; Leganchuk et al., 1998). However, it is unlikely that edge-scrolling presents these costs.

First, there is little evidence that pointing and scrolling are performed in parallel in compound tasks (Buxton and Myers, 1986). The biphasic model of Guiard et al. (2004) described earlier suggests that the pointing phase occurs once the scrolling phase has set the required reference frame. In addition, Zhai et al. (1997b) showed that the opportunity of parallel interaction does not guarantee superior performance to a sequential technique if other factors, such as the transfer function, are neglected.

Second, while switching to scrollbar operation remove attention and resources from the main task (Zhai et al., 1997a) because it requires focusing on and pointing to a relatively small widget, this is not the case when switching between pointing and edge-scroll modes. Activating edge-scrolling is less dependent on visual feedback because it is performed by entering a large dedicated area near a viewport edge, or by simply crossing the viewport edge. Moreover, taking the argument of Guiard et al. (2004) further, edge-scrolling can be regarded as a direct extension of pointing actions.

6.1.4. EDGE-SCROLLING

Few existing empirical works have tackled the design and evaluation of edge-scrolling techniques. This is all the more surprising when various forms of the technique are implemented in most desktop and mobile applications, supported by major GUI toolkits, and described in various patents. Here, we first present an analysis of the patents, revealing that edge-scrolling presents specific challenges that motivate a variety of different design choices whose consequences have not been examined yet. Then, we describe existing empirical comparisons of edge-scrolling to other scrolling techniques and show that they are not sufficient to characterize edge-scrolling performance.

---

[3] `http://docs.oracle.com/javase/7/docs/api/javax/swing/JComponent.html#setAutoscrolls(boolean)`. Last accessed 05/07/2015.
[4] `http://docs.oracle.com/javase/7/docs/api/java/awt/dnd/Autoscroll.html`. Last accessed 05/07/2015.
[5] `https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/NSScrollViewGuide/Articles/Scrolling.html#//apple_ref/doc/uid/TP40003463-SW3`. Last accessed 05/07/2015.

INVENTIONS

Various patents have described edge-scrolling techniques and have claimed to solve several specific issues, such as providing efficient control by pointing, disambiguating between cross-viewport manipulation and scrolling intentions, and reducing unintended scrolling. Note, however, that no public empirical work has validated the effectiveness of the following solutions.

Meier et al. (1993) described an "intelligent" scrolling technique allowing to scroll a viewport by dwelling inside a dedicated area placed along the inside edges of the viewport. The closer the pointer is to the edge, the faster scrolling occurs, as long as the pointer stays inside the scrolling area.

Bardon et al. (1997) disclosed a relative position-based technique and claimed that it was "more effective" than the rate control of existing edge-scrolling designs. When the pointer is outside the viewport, only pointer movements away from it cause scrolling movements in the same direction and with the same magnitude.

Kwatinetz (1996) commented that existing rate-based designs obliged users to overshoot their target more frequently due to high scrolling velocities, and proposed to improve rate-based edge-scrolling, which occurs in his invention when the pointer lies on the display edge during a dragging operation. Scrolling speed was set proportional to the acceleration of the pointer, as computed using the three last pointer positions recorded at a fixed interval, and it was decreased by three lines per second each time edge-scrolling mode was entered during a selection in order to facilitate the corrective phase.

While the previous designs focus mainly on scrolling performance, other issues with edge-scrolling have been mentioned, especially intention disambiguation with compound dragging and scrolling tasks, and risks of accidental scrolling when pointing. When a user drags an object and crosses the viewport edge, her intention is either to drop the object in another viewport, or to begin edge-scrolling with the viewport whose edge has just been crossed. This ambiguity is specific to situations where the scrolling area takes the whole space outside the viewport: short of defining a more limited scrolling area, a disambiguation mechanism must exist. Berry et al. (1991) proposed to use a mouse button press, noting that existing implementations disambiguated with a specific keyboard modifier that required both hands and competed with other existing uses of modifiers.

Instead, Li and Shrader (1998) proposed that position-based edge-scrolling be the default behavior when crossing the viewport edge while dragging an object, repositioning the pointer at the edge after each movement. With this technique, before being able to drag the object outside the viewport, the user must first explicitly hit a dedicated keyboard key. However, accidental activations, which can be frequent when close to the edge, could make this technique quite frustrating.

Belfiore et al. (1998) described the frequent accidental activations of edge-scrolling that occur when the dedicated scrolling area lies within the edges of the viewport. To reduce the risk of accidental scrolling, they proposed an activation method based on an "empirically proven" threshold on pointer velocity, calculated when the pointer is inside the scrolling area. In their invention, scrolling occurs at a fixed rate of four lines per second, only when pointer velocity stays below the threshold.

Altogether, these patents suggest that (1) edge-scrolling relies on new design dimensions, such as the geometry of a scrolling area, or the conditions leading to activation or deactivation of the edge-scrolling mode; and that (2) edge-scrolling presents specific and novel challenges, such as intention disambiguation. These dimensions and challenges have not been addressed by the existing scrolling literature, nor have they been mentioned explicitly in the following empirical studies comparing edge-scrolling performance to other techniques.

PERFORMANCE STUDIES

Johnson (1995) found that panning outperformed a rate-based edge-scrolling technique, *TouchEdge*, on touchscreens, however with very short scrolling distances, as the display was set to contain 70% of the entire document. TouchEdge did not feature any rate control mechanism and was activated by dwelling near the screen edge. While it minimized hand motion, it was not deemed suitable for fine positioning tasks.

Kulik et al. (2012) compared rate-based edge-scrolling to panning with the non-dominant hand during the manipulation of an object on a smartphone touchscreen with the dominant hand. Their rate-based edge-scrolling technique was active when the finger was less than 4 mm away from a screen edge. The bimanual method resulted in better performance in object dragging tasks with scrolling distances up to two fifths the size of the display.

Kaptelinin (1995) compared four navigation techniques based on indirect pointing, including a rate-based edge-scrolling technique. Edge-scrolling was active when the pointer was outside the viewport, and scrolling velocity was proportional to the distance to the viewport edge. Results showed that edge-scrolling outperformed scrollbar navigation, but was outperformed by two variants of a two-dimensional absolute scrolling technique.

There are two main reasons why the findings in the previous studies are not sufficient to characterize user performance with edge-scrolling. First, details on the scrolling transfer function are missing, preventing replication of the results (Quinn et al., 2012). Second, the short scrolling distances are not representative of the variety of possible scrolling tasks. And third, only rate-based variants were evaluated, not position-based ones which could be more adequate to short distances.

In summary, further investigation is needed to better understand edge-scrolling and assess its potential as a direct and expressive extension of pointing. The first step towards this goal is to be
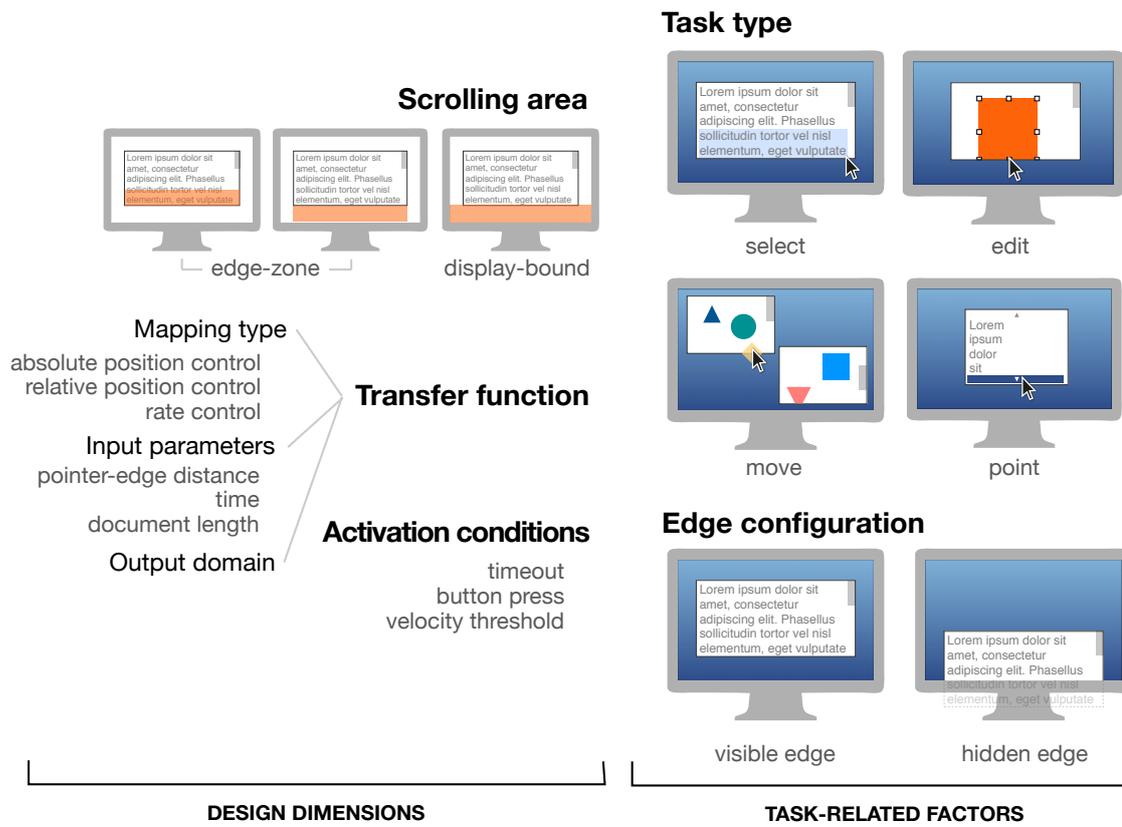
FIGURE 6.2: The design space of edge-scrolling provides three main dimensions (left) that can be used to describe a design, and two types of task-related factors (right) delineating its operating range.

able to precisely describe the design of a technique in terms of the dimensions discussed earlier. In the following section, we will describe a design framework that enables such descriptions. As edge-scrolling is already widely implemented, it will allow to precisely characterize and discuss the current design practice, user perception of potential usability problems, and user performance with edge-scrolling techniques, laying the groundwork for proper design and evaluation.

6.2. The design space of edge-scrolling techniques

The previous section has shown that previous literature on scrolling and edge-scrolling is not enough to characterize either the specific design aspects of edge-scrolling techniques, or the specific issues in interacting with them. As these aspects will be discussed throughout this chapter and the next one, it is important to first define and describe them more formally.

In this section, we define the dimensions of the edge-scrolling design space (Figure 6.2, left), describing possible options for each dimension. We also formulate task-related factors that may influence the design of an edge-scrolling technique (Figure 6.2, right). While the following discussion is limited to interactions with the bottom edge of the viewport, it should be understood that it applies for any given edge and scrolling direction (left, right, up, down). Figure 6.3 shows
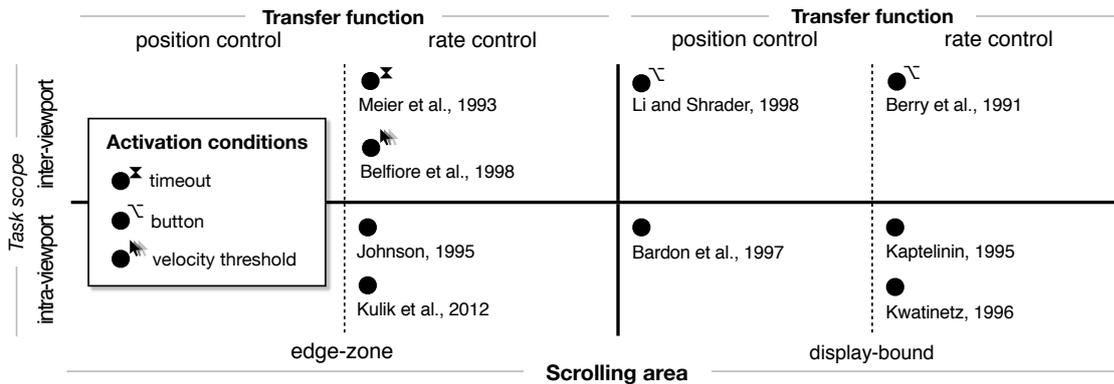
FIGURE 6.3: Existing edge-scrolling designs classified using the design space of edge-scrolling techniques from their descriptions in the literature.

the edge-scrolling designs from the literature discussed in the previous section classified according to the design dimensions and task-related factors defined in our framework.

6.2.1. DIMENSIONS

Edge-scrolling technique designers must make choices along three main dimensions: *scrolling area*, *transfer function*, and *activation conditions*. In edge-scroll mode, pointing inside a *scrolling area* relative to a given edge a viewport controls scrolling as specified by a *transfer function*. Entry into and exit from the edge-scroll mode both depend on *activation conditions*.

SCROLLING AREA

The *scrolling area* is the part of the screen that is used to activate and control edge-scrolling. It is defined relative to the edge of the viewport. Among the diverse possibilities, we focus particularly on *edge-zone* and *display-bound* areas.

An *edge-zone* scrolling area is placed near the considered viewport edge, has the same width as the viewport, and has a comparatively small height. While its vertical position can be anywhere relative to the edge, two specific cases can be further described. An inside edge-zone expands from a virtual line crossing the viewport to the considered viewport edge (or close to). Conversely, an outside edge-zone expands from the considered viewport edge (or close from) to a virtual line beyond the edge.

A *display-bound* area is a rectangle expanding from the considered viewport edge (or close from) and taking up the whole display space beyond the considered viewport edge. In contrast with edge-zones, which are finite and fixed, display-bound areas are virtually infinite and bounded by the size of the display only.

TRANSFER FUNCTION

An edge-scrolling technique manages the production of discrete scrolling displacements using a *transfer function* that defines a mapping from pointing input to scrolling output. Transfer functions may also use additional parameters, such as the length of the document (Cockburn et al., 2012), or the time spent inside the scrolling area.

There are three main *types* of input-to-output mappings. In *absolute position control*, the vertical position of the pointer in the scrolling area is mapped to the vertical scrolling position in the document. In *relative position control*, or simply *position control*, vertical pointing movements are directly translated into vertical scrolling movements. Scrolling amplitude is either derived from the amplitude of the pointing movement, or from the *pointer-edge distance*; that is, the vertical distance from the pointer to the edge. In either case, it is possible that only movements away from the viewport are taken into account. Finally, in *rate control*, the pointer-edge distance is mapped to the velocity of automatic scrolling.

The *input domain* is the set of possible pointing input values. It is limited by the size of the scrolling area. Correspondingly, the *output domain* is the set of possible output values that can be produced by the transfer function, be they positions, displacements, or velocities, depending on the type of mapping. Notable aspects of the output domain are its density—the continuous or stepwise quality of the evolution of successive output values—and its bounds—its minimum and maximum value.

ACTIVATION CONDITIONS

Switching to and from the edge-scroll mode requires fulfilling one or several *activation conditions*. By definition, the primary activation condition is that the pointer is inside the scrolling area. That is, the viewport scrolls as soon as the pointer enters the scrolling area.

This may be sufficient to initiate scrolling, but secondary activation conditions may also be used. For example, edge-scrolling may activate at the expiration of a timeout, after a button press, or if the velocity of the pointer when entering the area is above or below a predefined threshold. Activation and deactivation need not be symmetrical, as with the design of Li and Shrader (1998) described above.

6.2.2. TASK-RELATED FACTORS

Edge-scrolling can be used in various tasks which impose specific constraints on the design of the technique. Here we identify two types of *task-related factors* (Figure 6.2, right) that delineate the operating range of an edge-scrolling technique design: the high-level task type during which edge-scrolling is used, which defines its potential scope, and the configuration of the viewport (more specifically, of the considered viewport edge) inside the display.

FIGURE 6.4: Four types of target-directed tasks that may require edge-scrolling: (a) *select*, e.g. selecting a wide range of cells in a spreadsheet; (b) *edit*, e.g. resizing a shape in a drawing application; (c) *move*, e.g. moving a file in a subdirectory; and *point*, e.g. (d) pointing at a specific font in a large menu or (e) selecting a particular military unit on the map of a real-time strategy game.

TASK TYPE

We consider four types of target-directed tasks that may require edge-scrolling:

Select The user marks a part of the document for further action, specifying a starting point in the document space located inside the viewport by starting a dragging action, reaching a remote end point, and releasing the dragging mode to mark the final location of the selection. For example, selecting cells in a spreadsheet (Figure 6.4a).

Edit The user modifies one or more parameters of an object located inside the viewport by dragging, and reaching the desired values necessitates scrolling. For example, resizing a shape in a drawing application (Figure 6.4b).

Move The user displaces an object located inside the viewport by dragging, either to a remote point in the same document, or in another document. For example, moving a file icon in a subdirectory (Figure 6.4c), or moving a paragraph of within a text document by drag-and-drop.

Point The user reaches a remote point in a document and selects it. For example, selecting a font in a large menu (Figure 6.4d), or selecting a unit in a real-time strategy game (Figure 6.4e).

The nature of the user's task determines the potential scope of the interaction, which may turn to be an important consideration in edge-scrolling design. During the execution of *select* and *edit* tasks, there is no ambiguity about the viewport containing the user's intended target: it must reside within the viewport that contained the starting point. Hence, *select* and *edit* are considered *intra-viewport* tasks. However, *move* and *point* tasks are *inter-viewport* tasks: the target lies either within the initial viewport, or within a different viewport (for example, during drag-and-drop between windows). Designers must therefore carefully consider the risk that their design may interfere with the user's interaction with other viewports in inter-viewport tasks.

EDGE CONFIGURATION

The display layout also influences the nature of the user's task and her ability to effectively use different edge-scrolling techniques. Two different *edge configurations* form the variety of

possible positions of the viewport inside the display (Figure 6.2, bottom-right). In the *visible edge* configuration, the viewport edge lies within the display, and there is available splace for a scrolling area that extends beyond the viewport. However, in the *hidden edge* configuration, the viewport edge is either coincident with the display edge (e.g., with a maximized window), or lies outside the display. In the latter case, if the pointer is restricted by the bounds of the display, the user may not be able to access the totality of the scrolling area.

6.3. Reverse-engineering current designs

Edge-scrolling is currently implemented separately in each application or toolkit, therefore a typical user will face different implementations of the same functionality throughout the operating system. Although following the same general principle, edge-scrolling implementations may well behave differently, embodying drastically different design opinions. These designs can now be precisely described using the design space presented in the previous section, forming a base for comparison and evaluation.

In this section we detail a methodology for examining existing implementations and reverse-engineering their behavior by monitoring the scrolling output of a viewport in response to software-simulated pointing input. We then analyze the high-level characteristics of the edge-scrolling designs implemented in 19 common applications and GUI toolkits retrieved using our method. This analysis will highlight the most common design choices and help characterize the diversity and consistency of edge-scrolling.

6.3.1. METHOD

Each edge-scrolling technique has been examined using the following method: (1) the scrollable viewport implementing the technique was configured according to a predetermined setup (geometry, contents, and scrolling position); (2) the viewport was manipulated using several predefined interaction patterns; and (3) the scrolling response to these manipulations was then observed and measured. The resulting data allows to carefully describe a particular design, mainly on the scrolling area and transfer function dimensions, and determine how it fits in the design space.

INITIAL CONFIGURATION

We used the same initial configuration whenever possible. The viewport was 400 pixels high and located at the center of a 1440×900 display, leaving 250 pixels between the bottom edge of the viewport and the bottom edge of the display. This viewport contained a large document, but its contents varied depending on the type of viewport and behavior tested. Text documents used a *Lorem ipsum* filler text typeset in *Ubuntu Mono Regular 16px*[6]. List documents used similar font and size when possible. Graphical documents were 100,000 pixels high. Setting a unique document size across all tested viewports was not possible because of changing document margins and peculiarities of typesetting algorithms, but fortunately none of the tested designs depended on the size of the document.

---

[6]We ensured that the same physical font size was used across all the different operating systems by using pixels instead of points as the font size unit.

PROCEDURE

To determine the properties of an edge-scrolling technique design according to the design space, we combined simple observation and logging of the scrolling response to both human and computer-generated manipulations of the scrollable viewport implementing the technique. The following procedure is described for downwards scrolling using the bottom edge of the viewport, but it can be applied to any edge direction or input mode.

In all examined viewports, edge-scrolling occurred in response to dragging interactions. The objects (i.e., text, file icon, etc.) located underneath the starting point sp of the dragging interaction determined the type of task to be performed. For instance, on a viewport displaying vector graphics, dragging from an empty point started a *select* task, but dragging from any object in the document started a *move* task.

We first used ad-hoc dragging interactions, observations, and logging to determine the following: type of scrolling area, type of transfer function mapping, attendance to different input parameters (pointer-edge distance, time, pointer movements, font size, document length, and viewport height), and availability of the hidden edge configuration.

We then simulated a specific mouse input pattern with all behaviors (when possible) in order to precisely characterize the scrolling area and the remaining parameters of the scrolling transfer function. The pattern, described in Figure 6.5, consists of as many dragging sequences as there are pixels in the scrolling area. Each dragging action starts from sp, jumps directly to the relevant pixel, and ends after a ten second dwell. By logging the scrolling position every second for the ten seconds of each dragging sequence, we could measure the effect of pointer-edge distance on scrolling velocity (for rate-based functions) or distance (for position-based ones), as well as the limits of the output domain.

APPARATUS

We built the following two main software components: one simulated mouse input, and the other logged the scrolling displacements of the considered viewport. To simulate mouse input on Windows, we developed a tool in C# using the Win32 API. For event simulation on OS X, we developed another tool in Objective-C using Quartz Event Services.

To log scrolling output, we used various methods to meet the requirements of the target viewport. The edge-scrolling behavior of a standard widget from a GUI toolkit was logged in a custom program or script. The program configured a window containing the widget and filled it with the appropriate document. Then, it recorded mouse events and changes in the scrolling position of the document inside the widget. Mouse and scrolling event interception was done either by subclassing the class representing the scrollable widget and overriding internal event handling methods, or by subscribing to events. A similar method was also employed for web browsers: a custom HTML page was designed and a script subscribed to DOM events `mousedown`, `mousemove`, `mouseup`, and `scroll`.

Data:

p	the vertical position of the pointer relative to the bottom edge of the viewport
s	the vertical position of the viewport
sp	the vertical position of the starting point of the dragging interaction (task-dependent)
y_m	the top vertical position of the bottom scrolling area relative to the bottom edge of the viewport
s_{size}	the height of the bottom scrolling area

Result:

$ds_t[y]$	the recorded amplitude of total scrolling displacements on time t of a dragging interaction to point y of the scrolling area

$y_M \leftarrow y_m + \max\{s_{size}, 250\}$
for each y in $\{y_m, \ldots, y_M\}$ **do**
    $s \leftarrow 0$
    $p \leftarrow sp$
    mouse down
    $p \leftarrow y$
    for each t **in** $\{1, \ldots, 10\}$ **do**
        wait 1 second
        $ds_t[y] \leftarrow s$
    mouse up

FIGURE 6.5: Simulated mouse input pattern applied to a scrollable viewport in order to systematically extract transfer function details of the edge-scrolling technique it implements. The pattern describes discrete successive dragging actions, one for every pixel of the scrolling area.

To log edge-scrolling output in standalone applications, we used accessibility APIs when available. On Windows, scrollable viewports implement the `ScrollPattern` control pattern using Microsoft's UI Automation API. The scrolling position in pixels can be derived using properties `VerticalScrollPercent`, `BoundingRectangle`, and `VerticalViewSize`. With OS X's Accessibility API, scrolling position was provided by the `AXPosition` property of scrollable viewports implementing the `AXScrollArea` role.

In some applications, such as Photoshop, only the scrollbar accessibility object could be observed, not the viewport itself. In this case, the scrolling position y was approximated based on the normalized scrollbar value $v \in [0, 1]$, the height of the viewport h_v, and the height of the document h_d, as follows: $y \approx v(h_d - h_v)$.

6.3.2. RESULTS

Using the above method, we examined edge-scrolling in 19 common applications and GUI toolkits, spread across three operating system versions (Windows 7, 8, and OS X 10.9.2). Depending on the application, the inspected viewports displayed web pages, text documents, images, spreadsheets, or lists.

We tested each application for *select* and *move* tasks. We discarded *edit* tasks because they were allowed in only four applications. In the following, the combination of an application and a
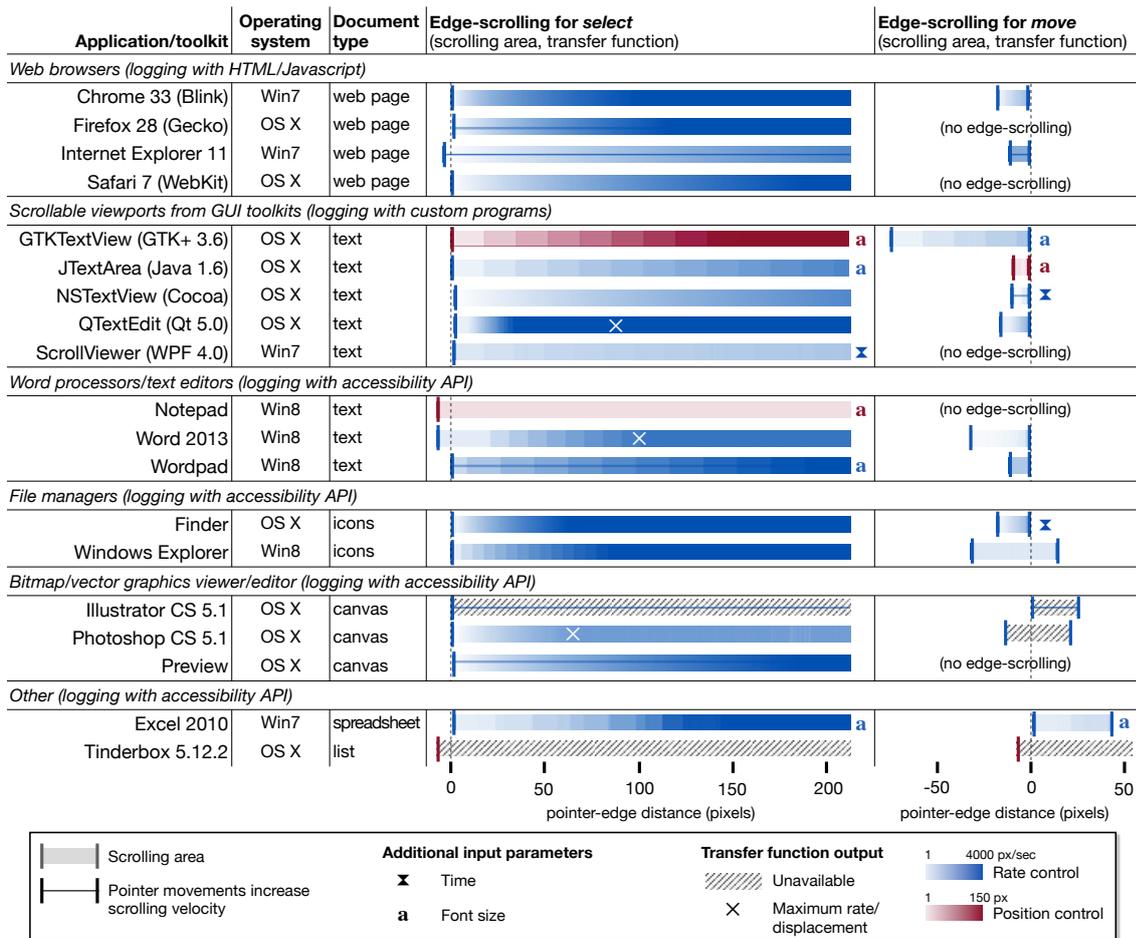
FIGURE 6.6: Overview of the 19 applications and toolkits examined across three operating system versions (Windows 7, 8, and OS X 10.9.2). Bars describe scrolling areas and transfer function output for *select* and *move* tasks. Color indicates rate control (blue) or relative position control (red), and opacity denotes either the average velocity (rate control), or the average displacement (position control) for a given pointer-edge distance. A central horizontal line in a scrolling area indicates that pointer movements generate scrolling displacements. Bars are hatched when transfer function data was not available due to incomplete or unreliable accessibility implementation in the application.

particular task type will be referred to as a *configuration* (e.g., GTKTextView$_{select}$). When no task type is specified (e.g., Tinderbox$_*$), we refer to both edge-scrolling configurations surveyed for that application.

Overall, we surveyed 19 *select* and 14 *move* configurations, making a total of 33 configurations, described on Figure 6.6. We focused on downwards scrolling using the bottom edge of the viewport. In a few applications, we found that the scrolling area was different depending on the edge direction, but additional examinations confirmed that other properties remained stable.

SCROLLING AREA

All *select* configurations used a *display-bound* scrolling area, and all *move* configurations used an *edge-zone* area (with the exception of Tinderbox$_{move}$ whose area is display-bound).

The twenty *display-bound* areas extended down to the bottom edge of the display, but while they all began close to the edge of the viewport, we observed variations in the exact vertical starting position. In 15 configurations the scrolling area started either exactly at the viewport edge or up to 2 pixels away from it. For the 5 remaining configurations, the area started a few pixels inside the viewport (4 pixels for InternetExplorer$_{select}$, 8 pixels for Notepad$_{select}$ and Word$_{select}$, and 10 pixels for Tinderbox$_*$), becoming the only part of the area available when the viewport is maximized.

There were thirteen *edge-zone* scrolling areas. Among them, there were nine inside edge-zones positioned along the bottom edge of the viewport or 1 pixel away from it, with sizes varying from 8 pixels (JTextArea$_{move}$) up to 75 pixels (GTKTextView$_{move}$). The size of the scrolling area of GTKTextView$_{move}$ was 20% of the height of the viewport. We also found two straddling edge-zones, WindowsExplorer$_{move}$ and Photoshop$_{move}$, which were not strictly inside nor outside, as they lied on both sides of the bottom edge of the viewport. The two remaining areas, Illustrator$_{move}$ and Excel$_{move}$, were outside edge-zones that did not intersect the viewport, and started at its bottom edge rather than ending at it. Interestingly, the space outside the viewport of these outside edge-zones overlapped a scrollbar, a status bar, or both. Because this space is likely to contain neutral interface components, it can be safely used to extend the edge-scrolling input domain without risking unintended scrolling.

These findings suggest a tradeoff between the amount of control that an edge-scrolling design affords and the possibility of ambiguous user action. With *select* tasks, most designs can maximize the input domain because moving the pointer outside the viewport cannot be interpreted as extending the selection to other viewports—selections spanning multiple viewports do not exist. In contrast, *move* tasks can span multiple viewports, therefore moving the pointer outside the source viewport is equivocal to the system: either the user wants to control edge-scrolling, or she wants to target another viewport for the *move* task. Designers chose to use fixed edge-zone areas in order to avoid ambiguous situations, regardless of the limited amount of control. The only exception was Tinderbox$_{move}$, which used a display-bound area nonetheless, discriminating between edge-scrolling and targeting another viewport by using an additional activation condition: edge-scrolling was activated when the pointer entered the scrolling area with a velocity lower than a predefined threshold.

Finally, scrolling areas were more often unavailable than not in the *hidden edge* configuration, with the exception of NSTextView$_{select}$, Preview$_{select}$, and Excel$_{select}$, which used fixed edge-zone areas starting above the display edge (50px for NSTextView$_{select}$ and Preview$_{select}$, 1px for Excel$_{select}$). Finder$_{select}$ also supported edge-scrolling in the hidden edge configuration, but rather than redefining the scrolling area based on the display edge, it smoothly moved the window upward until the entire viewport was accessible again, returning it to its original location upon completion of the dragging action.
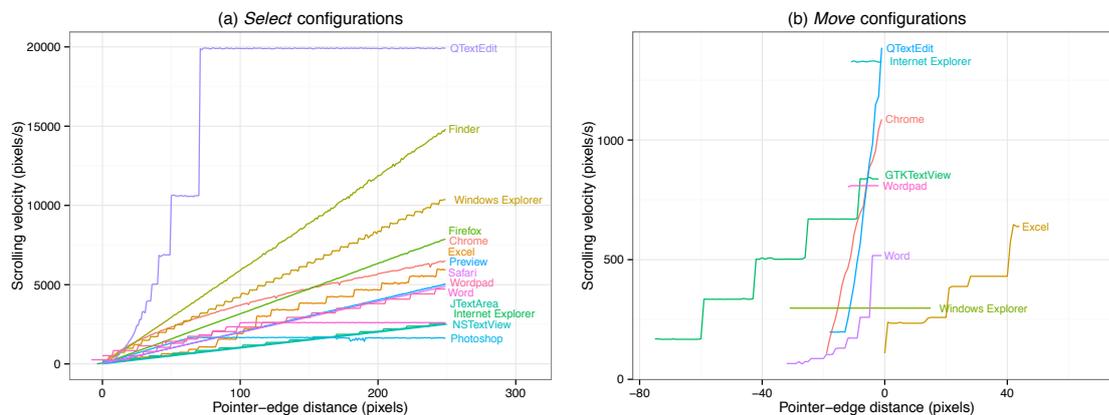
Figure 6.7: Scrolling velocity by pointer-edge distance for (a) 14 *select* and (b) 8 *move* configurations with rate-based transfer functions that did not depend on the time spent in the scrolling area.

TRANSFER FUNCTION

Among the 33 configurations, only five used relative position control (GTKTextView$_{\text{select}}$, JTextArea$_{\text{select}}$, Notepad$_{\text{select}}$, and Tinderbox*). The remaining 28 used rate control, most of which are shown in Figure 6.7. The Tinderbox*, Illustrator*, and Photoshop$_{\text{move}}$ configurations (shown hatched in Figure 6.6) could not be examined: Tinderbox and Illustrator did not implement the system accessibility APIs at all, and in Photoshop the accessibility objects representing scrollbars did not update reliably during drag-and-drop.

INPUT PARAMETERS

As shown in Figure 6.6, most of the transfer functions used pointer-edge distance as the primary input parameter. Additionally, some functions used font size, time, and pointer movements.

Font size was used in the transfer function of 9 configurations: GTKTextView*, JTextArea*, Notepad$_{\text{select}}$, Wordpad*, and Excel*. In these configurations, edge-scrolling is performed one line at a time. Interestingly, in applications displaying rich text documents, such as paginated documents (Word) or web pages (Firefox), transfer function did not depend on font size, presumably because their possibly complex layout precludes the assumption of a fixed-size base element.

Time was an input parameter in three rate control configurations. Scrolling velocity increased over time with Finder$_{\text{move}}$ and NSTextView$_{\text{move}}$, but it decreased over time with ScrollViewer$_{\text{select}}$.

Pointer movements in any direction generated additional scrolling displacements in 9 configurations: Finder$_{\text{select}}$, InternetExplorer*, NSTextView$_{\text{move}}$, Wordpad$_{\text{select}}$, Illustrator*, Preview$_{\text{select}}$. In these configurations, pointing movements within the scrolling area also sent scrolling events, momentarily increasing the frequency of scrolling output and giving the effect of faster scrolling. This may be useful when the input domain is small due to a constrained input area, for example when the window is maximized.

Finally, WindowsExplorer$_{\text{move}}$, Wordpad$_{\text{move}}$, and InternetExplorer$_{\text{move}}$ used constant transfer functions that took no input parameter.

Transfer function	Task type	Mapping type	Approximate model parameters			
Constant		$f(x) = a$	**a** (y-intercept)			
Notepad	select	position	20			
JTextArea			17			
Windows Explorer	move		297.58			
Wordpad		rate	809			
Internet Explorer			1328			
Linear		$f(x) = a + bx$	**a** (y-intercept)	**b** (slope)		
NSTextView			0.25	10		
Internet Explorer			49.96	10		
Preview	select	rate	-10.36	20.26		
Safari			19.51	19.59		
Firefox			-3.4	31.67		
Finder			-53.87	56.06		
Chrome	move		1134.44	54.18		
Regular staircase		$f(x) = a + b(1 + \lfloor \frac{x+\tau}{\sigma} \rfloor)\sigma$	**a** (y-intercept)	**b** (slope)	**σ** (step size)	**τ** (x-offset)
GTKTextView		position	0	1	17	0
JTextArea			0	10	17	0
Wordpad	select		521.16	16.66	18	-7
Word		rate	0	26	10	-10
Photoshop			60	28	2	-4
Windows Explorer			0	41	6	2
GTKTextView	move		670	9.85	17	8
Excel			35	10	20	0
Nonlinear			**Comments**			
ScrollViewer			time-based			
● Excel	select		irregular staircase			
● Chrome						
● QTextEdit		rate				
NSTextView			time-based			
● Word	move		irregular staircase			
● QTextEdit						
Finder			time-based			

FIGURE 6.8: The transfer functions of 28 configurations of edge-scrolling for which we could successfully measure scrolling data, organized into four classes, together with approximate models.

TRANSFER FUNCTION OUTPUT

Figure 6.8 arranges transfer functions in four classes depending on the nature of their scrolling output response $f(x)$ (in pixels for position control functions, in pixels/s for rate control ones) to pointer-edge distance input x (in pixels).

Constant functions (5 configurations) always produce the same output value a in response to any pointer-edge distance input ($f(x) = a$), although a changed with font size for Notepad$_{select}$ and JTextArea$_{move}$[7]. Therefore, their output domain has exactly one value, offering more limited control than what the scrolling area can afford (e.g., WindowsExplorer$_{move}$, 46 pixels).

---

[7]With these configurations, a was the line height, which differed between applications due to varying typesetting algorithms.
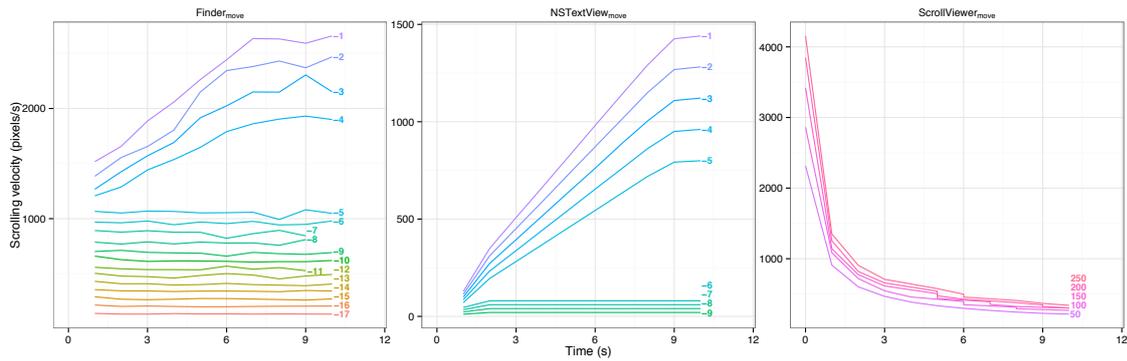
FIGURE 6.9: Scrolling velocity as a function of time at different pointer-edge distances (line labels, in pixels) for three edge-scrolling configurations that included the time spent in the scrolling area as an input parameter.

The untapped control potential is all the more substantial with display-bound areas such as Notepad$_{\text{select}}$.

Linear functions (7 configurations) map pointer-edge distance linearly to scrolling output ($f(x) = a + bx$), with slopes (b) varying from 10 to 56.06. All of them used rate control, and only one *move* configuration used a linear function (Chrome$_{\text{move}}$).

Regular staircase functions (8 configurations) increase scrolling output in progressive steps of fixed size σ as pointer-edge distance increases: $f(x) = a + b(1 + \lfloor (x + \tau)/\sigma \rfloor)\sigma$, given a (y-intercept), b (slope), σ (step size), and τ (x-offset). In the 5 configurations using a text document and a regular staircase function, step size was equal to the line height (which depended on font size), causing discrete line-by-line movements. Compared to linear functions, regular staircase functions basically downsample the output domain.

Other nonlinear functions (8 configurations) were grouped into the fourth class. In QTextEdit∗, Excel$_{\text{select}}$, and Word$_{\text{move}}$, the transfer function increased stepwise, but width and height were not constant across the input domain. In addition, the transfer function of Excel$_{\text{select}}$ depended on font size. The transfer function of Chrome$_{\text{select}}$ varied with the type of document and the hardware configuration (this will be discussed later on). Finally, Figure 6.9 shows three rate-based configurations where time was an input parameter. In NSTextView$_{\text{move}}$ and Finder$_{\text{move}}$, the time spent in the scrolling area had little effect until pointer-edge distance was above 5 pixels. The change was more continuous over time in ScrollViewer$_{\text{select}}$, but pointer-edge distance had virtually no effect.

OUTPUT DOMAIN

The class of transfer function output affected output domain density, and therefore the amount of control: for example, regular and irregular staircase functions had sparser output domains than linear functions.
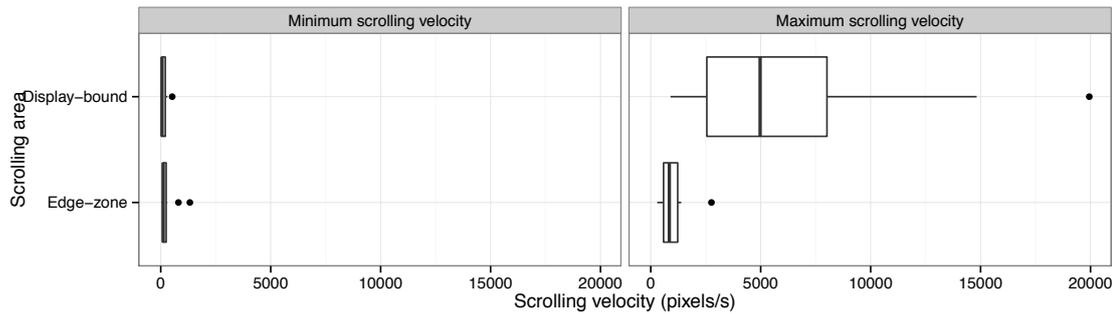
FIGURE 6.10: Distributions of the minimum and maximum reachable velocity across edge-scrolling configurations for display-bound and edge-zone scrolling areas.

Minimum and maximum output values within our setup[8] varied substantially for rate-based functions. Figure 6.10 shows the minimum and maximum scrolling velocities we measured with these functions for display-bound and edge-zone scrolling areas. No significant difference was found regarding minimum velocity (Mann-Whitney $U = 60$, $Z = 1.382$, $p = .178$) between edge-zone (median 112 px/s, IQR 161 px/s) and display-bound (median 60 px/s, IQR 167 px/s) scrolling area configurations. However maximum velocity was significantly lower ($U = 166$, $Z = -3.849$, $p < .001$) for edge-zone configurations (median 844 px/s, IQR 644 px/s) than for display-bound ones (median 4975 px/s, IQR 5463 px/s).

Therefore, while the minimum scrolling velocity that can be produced is similar between both tasks, the range of scrolling velocities is likely to be much smaller with *move* tasks than with *select* tasks, likely because *move* configurations used smaller edge-zone scrolling areas. However, concerning display-bound configurations, no factor could explain the wide spread of maximum measured scrolling velocities. This apparent variety may then reflect different design opinions on how transfer function output domains can optimally support user performance.

In addition, we found that the output domain was capped in only three rate-based configurations with display-bound scrolling areas, and that this capping had different effects on control. The upper bound for QTextEdit$_{select}$ was 20000 px/s (reached at pointer-edge distance 70px), while the highest maximum velocity in other configurations as 14823 px/s (reached at 250px, the maximum pointer-edge distance tested): the design of QTextEdit$_{select}$ sacrifices resolution to compress a wide range of scrolling output values in a small area. In contrast, Word$_{select}$ (max. 2600 px/s, reached at 100px) and Photoshop$_{select}$ (max. 1650 px/s, reached at 65px) simply prevent users from reaching higher scrolling velocities.

ACTIVATION CONDITIONS

Although we did not focus on this aspect, we experienced different activation conditions. During *select* tasks, edge-scrolling always felt instantaneous, while we sometimes experienced delayed

---

[8]The minimum and maximum output values discussed here refer to the scrolling velocities we have been able to measure using the procedure presented earlier. For example, with a display-bound configuration, the maximum velocity was that measured at a 250px pointer-edge distance.

edge-scrolling activation during *move* tasks. Tinderbox$_{move}$ was the only configuration where activation depended on the pointer entering the scrolling area below a predefined velocity threshold, but we could not measure it. Further refinements of our examination method would allow to precisely characterize activation conditions.

6.3.3. SUMMARY

The results demonstrate substantial variance in the design approaches to edge-scrolling, with only two areas of general conformity. First, rate control techniques are much more common than others. Second, *select* tasks typically allow larger *display-bound* scrolling areas than the *edge-zone* areas typically used for *move* tasks. This observation can be largely attributed to the ambiguity of dragging in *move* tasks: the user's target may lie within the current viewport, but it may also lie within a different one. This ambiguity does not exist for *select* tasks. Other than these two general trends, the variance in design approaches is substantial, with markedly different transfer functions and input parameters, leading to very different opportunities for control.

Edge-scrolling transfer functions are typically rate-based, with common input parameters including pointer-edge distance, font size (for text documents), and pointer movements. The time spent in the scrolling area is rarely taken into account. Concerning the relationship between pointer-edge distance and scrolling output, it can be constant, linear, staircase, or nonlinear. The density and limits of the transfer function's output domain influence control in different ways. Nevertheless, the amount of control is lower in inter-viewport tasks, where the range of reachable velocities is narrower. In intra-viewport tasks, this range is wider, but also much more inconsistent.

6.4. Surveying current edge-scrolling use

Reverse-engineering techniques provide us with ample information on the behavior of existing designs, but says nothing about how users perceive and use them. Thus, we designed and deployed an online survey to determine whether edge-scrolling is actually used in real-world situations, and which salient aspects of interaction with edge-scrolling relate to the design parameters we previously identified.

6.4.1. SURVEY DESIGN

The previous analysis of existing designs led us to suppose that edge-scrolling was primarily used in the desktop when performing *select*, *move*, and *edit* tasks, therefore we did not include *point* tasks in our questions. We found three alternative techniques that also allow scrolling during these tasks: concurrently using the mouse wheel or scroll wheel gestures (*wheel*), extending a selection while scrolling using arrow keys and keyboard shortcuts (*keyboard*), or scrolling before extending a selection by clicking while holding the Shift modifier (*shiftclick*). Given the implicit nature of edge-scrolling and the presence of alternative techniques, we were concerned that focusing on edge-scrolling from the start would confuse participants and undermine the
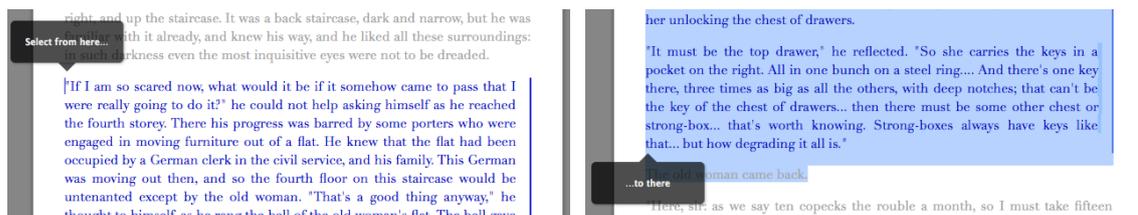
FIGURE 6.11: Text selection task for the second part of the online survey. Respondents were free to use any scrolling technique while selecting the paragraph containing blue text.

validity of the survey. Thus we decided to make text selection the initial topic of the survey, and to gradually shift focus to edge-scrolling along the questions. We checked that this design would allow respondents to understand the idea of edge-scrolling and recall prior usage during pilot tests with ten participants.

PROCEDURE

The survey featured six main parts (described below) which were displayed each in a separate web page. The participant could not continue to the next part until all the mandatory questions of the current one had been answered, and she could not go back to a previous part once it had been completed. All Likert-type items featured an additional "I don't know" item.

After a short introduction describing the general structure of the survey and exposing the estimated time required to complete it, part one asked general questions about computer use: principal operating system, web browser, pointing device, as well as dominant hand and time spent using a desktop or laptop computer daily.

Part two was a guided selection task designed to determine which scrolling technique the respondent chose when asked for text selection without specific constraint. The page displayed a 15-point 4932-word text along with a tooltip containing instructions (Figure 6.11). The respondent had to select a particular paragraph displayed in blue that was long enough to necessitate substantial vertical scrolling. The user could proceed with the next part once the entire paragraph had been selected.

Part three discussed the task that had just been performed and asked the respondent for feedback. She was then asked whether she could concurrently point and scroll using her pointing device. After being introduced to the issue of scrolling while dragging, she had to specify the frequency at which she scrolled while performing *select*, *edit* and *move* tasks using 5-point Likert-type items (from *never* to *very often*). A short text and an animation illustrated each task.

Part four focused on the four different scrolling techniques identified earlier (*edgescroll*, *wheel*, *keyboard*, and *shiftclick*). The respondent had to enter the frequency of use of each technique during a *select*, *edit*, or *move* task. A short text and an animation illustrated each technique. The order of presentation of each technique in the survey was randomized across respondents to limit bias. An optional free-form text item asked for details about the use of the techniques.

Part five focused on one of the four techniques, edge-scrolling. After a short description and an animation recalling the technique, the respondent was asked whether she was aware of its existence before taking the survey. Then, she had to answer questions on the difficulty to control edge-scrolling (from *very hard* to *very easy*), her general experience with edge-scrolling (from *very frustrating* to *very satisfying*), and the frequency at which the edge-scrolling behavior differed from her expectations in daily use (from *never* to *very often*), using 7-point Likert-type items. In addition, three optional free-form text items asked to describe particular problems encountered while using edge-scrolling techniques, personal strategies to counter specific issues, and expectations for future designs.

Part six was a simple demographic questionnaire asking for age, gender, nationality, and occupation, with age and occupation being optional. Upon completion, the respondent was thanked for her participation and was invited to share the survey on social networks.

PARTICIPANTS AND APPARATUS

Respondents were recruited through local university mailing lists and online social networks (Facebook, Twitter, Google+). No compensation was offered for completing the survey, and respondents were informed that their answers were anonymous.

The survey tool was implemented as a HTML and Javascript website communicating with AJAX to a home-made Node.js server using the ExpressJS library. During the selection task in the second part of the survey, the software logged pointing, keyboard, and scrolling events using the DOM Javascript API.

6.4.2. QUANTITATIVE RESULTS

In two months, 214 volunteers completed the survey (235 including those who abandoned). The majority of the respondents were male (71%) and the median age was 26 (IQR 23-32; 46 preferred not to answer). Most of the respondents estimated that they used a computer for more than five hours daily (84%). Of the 155 respondents who specified their occupation, most were university students (65), researchers (36), or engineers (25). More respondents were on Windows (58%) than on OS X (29%) or Linux (13%). The most used web browsers were Chrome (47%), Firefox (34%), and Safari (13%). Finally, almost all respondents operated their pointing device with their right hand (98%), and 66% were mouse users, while the others used a touchpad.

ANALYSIS OF THE SELECTION TASK

To analyze the selection task in the second part of the survey, we reconstructed the methods used to complete the task using the recorded timestamps and high-level event descriptions, such as "the page scrolled down" or "the pointer went outside the viewport". Consecutive events of the same type were coalesced in order to compute event durations (e.g., "the page scrolled down for 500 milliseconds"). Given a sequence of high-level events, we used a finite state machine to automatically determine which techniques had been used by a participant during the selection
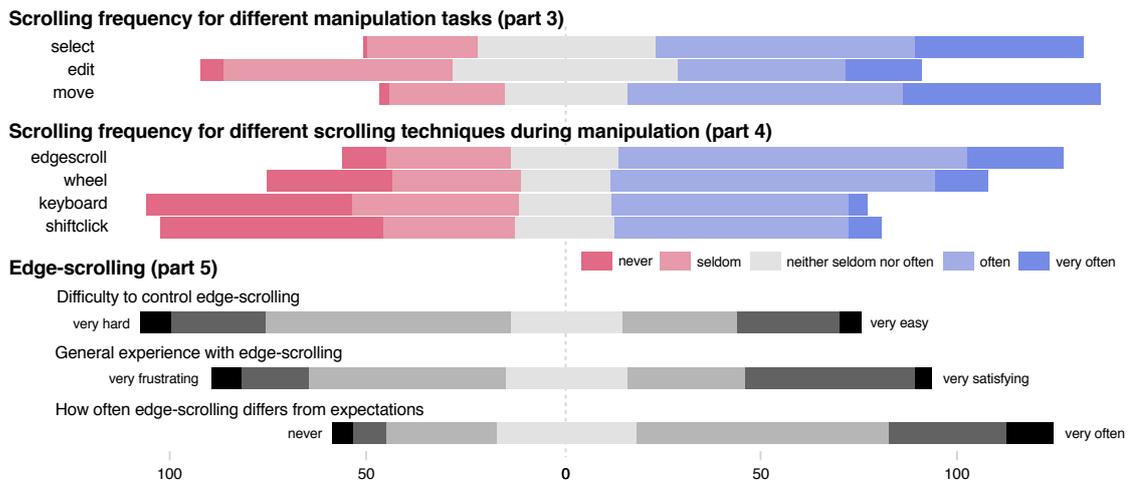
FIGURE 6.12: Responses to Likert-type items on parts 3, 4, and 5 of the survey.

task. For example, when a *mouse down* event is followed by a *pointer outside* event and a *page scroll* event (without a simultaneous *scroll wheel* event), then we consider that edge-scrolling has been used. Only 7% of the respondents used more than one method during the task, and for them we retained the method that had the longest duration. We double-checked this manually by reading through the event logs for each respondent of these 7%.

Overall, *edgescroll* was the most used technique in the selection task, with 57% of the respondents, while 34% used *wheel*. In comparison, *shiftclick* (8%) and *keyboard* (1%) were rarely used. This provides the first piece of evidence that edge-scrolling is a widely used scrolling method during text selection.

PERCEIVED FREQUENCIES OF SCROLLING FOR DIFFERENT TASKS AND TECHNIQUES

Part three of the survey concerned the perceived frequency of scrolling while performing *select*, *edit*, and *move* tasks. To determine a frequency order for the tasks, we compared the responses to the corresponding Likert-type items using a paired comparison approach[9] implemented in the R package prefmod (Hatzinger and Dittrich, 2012), and performed as follows: paired comparison patterns are derived from the Likert item responses and are then analyzed using a paired comparison pattern model (Dittrich et al., 2007). The model's parameter estimates are then transformed into *worth* values for each item, indicating the relative importance of the item on a preference scale. The sum of all worth values for a given scale is 1, and their order determine the frequency order of the items.

Part four concerned the perceived frequency of the four previously identified scrolling techniques (*edgescroll*, *wheel*, *keyboard*, and *shiftclick*) during a *select*, *edit*, or *move* task. We used the same

---

[9]This approach allows proper statistical inference on the data compared to more common methods that either were only descriptive (e.g., comparing medians), made assumptions that did not fit the nature of the data (e.g., parametric tests), or did not take into account the fact that Likert-type responses are relative (see Dittrich et al. (2007) for a full discussion).

Model 1 parameter	Estimate	Std. err.	Z-value	p-value
select	-.106	.060	-1.776	.076
edit	-.612	.069	-8.871	<.001
Model 2 parameter	**Estimate**	**Std. err.**	**Z-value**	**p-value**
edgescroll	.188	.055	3.455	<.001
wheel	.187	.054	3.434	<.001
keyboard	-.036	.054	-.662	.508
edgescroll:touchpad	.246	.099	2.468	.014
wheel:touchpad	.004	.094	.042	.966
keyboard:touchpad	.096	.095	1.012	.312
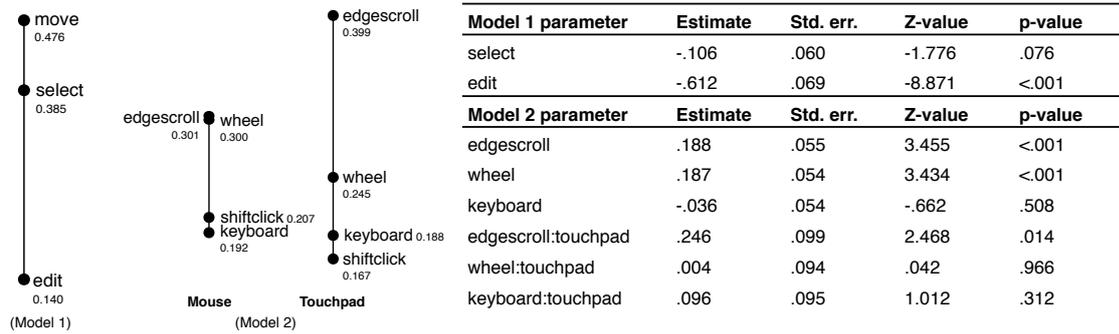
FIGURE 6.13: On the left, item worths derived from each paired comparison pattern model (Dittrich et al., 2007) used to determine the relative order of frequency of scrolling for *select*, *edit*, and *move* tasks (model 1; part 3 of the survey), and the relative order of frequency of use of different scrolling technique during these tasks (model 2; part 4 of the survey). On the right, parameters of interest for both models, used to determine item worths.

method as part three to derive a frequency order for the techniques. However, as we supposed that mouse and touchpad users would induce different technique usage patterns, we also included the preferred input device in the second model as a subject-specific covariate in order to determine its effect on the relative frequencies.

We found no noticeable understanding problem for these parts. In pilot tests, we found that the task and technique descriptions were generally well understood, and testers found that it was not too difficult to estimate frequencies of use. In the survey, only two respondents answered "I don't know" for the frequency of scrolling while performing *edit* tasks, and only one respondent did so for the frequency of scrolling using *edgescroll*.

Responses for parts 3 and 4 are summarized on Figure 6.12, and Figure 6.13 describes the parameters of interest and derived item worths for the models used to analyze parts 3 and 4.

RELATIVE SCROLLING FREQUENCY BY TASK

Scrolling is more often performed with *move* and *select* tasks than with *edit* tasks, according to item worths shown on Figure 6.13.

This result follows the findings of the reverse-engineering analysis, where we found that users had more opportunities to select and move objects than to do any other kind of direct manipulation. For example, any text widget affords selecting and moving portions of text around, but there is no opportunity to resize or rotate them. The same occurs with list widgets, such as those found in file browsers.

RELATIVE FREQUENCY BY SCROLLING TECHNIQUE

Edge-scrolling is perceived as one of the most often used scrolling techniques while performing *select*, *edit* or *move* tasks: overall, respondents found that they used *edgescroll* and *wheel* significantly more often than *keyboard* and *shiftclick* (Figure 6.13).

Mouse users indeed rated their frequency of use of *edgescroll* and *wheel* higher than *keyboard* and *shiftclick*. For touchpad users, similar patterns can be observed, but edge-scrolling is used significantly more frequently than for mice users, likely because the other techniques may not be as easy to use with a touchpad. In particular, two potential causes may discourage the use of scrolling gestures while dragging on a touchpad. First, the device may not allow concurrent pointing and scrolling, as reported by 24% of the touchpad users in our survey. A Chi-square test with Yates' continuity correction showed that significantly less participants could point and scroll concurrently with a touchpad than with a mouse (χ^2(1, N=201)=10.51, p=.001, ϕ=.24). Second, scrolling gestures may be physically difficult to perform while holding the dragging state on some touchpads. For example, the vertical scrolling zone to the right of a standard laptop PC touchpad might prove hard to reach with a finger while concurrently pressing the left button with the thumb.

SUBJECTIVE ASSESSMENTS OF EDGE-SCROLLING EXPERIENCE

Overall, the Likert-type items specific to edge-scrolling indicated that it is somewhat hard to control, that using it can be frustrating or satisfying, and that users generally notice diversity in existing behaviors. Most of the respondents declared knowing edge-scrolling before the survey (93%), and an overwhelming majority (98%) were able to answer the items of the fifth part related to their experience with edge-scrolling.

Likert-scale responses showed that respondents encountered difficulty controlling the velocity of edge-scrolling (median response "somewhat hard", Figure 6.12). 51% of the respondents answered between "somewhat hard" and "very hard" to the corresponding Likert-type item, while 32% answered between "somewhat easy" and "very easy", and 4 answered "I don't know". We found no evidence that the pointing device affects the attitude towards edge-scrolling control difficulty, as a Mann-Whitney U test showed no significant difference between mouse and touchpad users (U=4981, Z=.032, p=.975).

The general experience reported for edge-scrolling was almost equally divided between frustrated (41%) and satisfied (40%) users (median response "neutral"). Again, we found no evidence that this attitude is affected by the preferred pointing device (U=5064.5, Z=.327, p=.775).

Regarding how often edge-scrolling differs from expectations, 50% of the respondents answered between "somewhat often" to "very often", while only 19% answered between "seldom" and "never". With 31 participants (15%) answering "I don't know", this item was the most difficult to answer, probably due to its relative complexity, but results still suggest that respondents noticed the lack of consistency in edge-scrolling behaviors.

6.4.3. THEMATIC ANALYSIS

To investigate users' experiences with edge-scrolling and other scrolling techniques, we examined the comments left by 74 respondents (35%) in the free-form text items of the fourth and fifth parts of the survey using a bottom-up thematic analysis approach, as described by Braun and Clarke (2006)[10].

We started the analysis with the following question: what design factors are important to support expressive, direct, and effective edge-scrolling? Coding was performed single-handedly and the process resulted in five main themes articulated as follows[11].

Users may rely on the *context of use* to select a scrolling technique appropriate for the task at hand, but using other techniques may be the result of *compensating* several experienced weaknesses of edge-scrolling: difficult scrolling *control*, suboptimal *performance*, and too much *heterogeneity*.

CONTEXT OF USE

The choice of a scrolling technique during a *select*, *edit*, or *move* task depends on the context of use: *"it depends on my use"* (P55). The context of use includes various factors, such as:

- the current application (*"It depends on the application, operating system and how familiar I am with the application"*, P141);
- the type of object being manipulated (e.g., *"I only use shift click when selecting multiple items in a long list"*, P127);
- the device in use (e.g., *"I use the wheel only during a manipulation with a mouse"*, P222);
- the distance to travel (e.g., *"for small parts of text, I prefer to use keyboard shortcuts while when I need to select entire paragraphs, I prefer to use the edge-scroll technique"*, P160);
- whether the endpoint location is known (*"edge scroll when I am not sure where to stop and the Maj + Click when I know exactly"*, P175);
- or whether the user is already dragging (*"if I find out that I am already selecting I expect edge-scroll to work"*, P196).

COMPENSATING WEAKNESSES

Several respondents commented that edge-scrolling has low activation requirements compared to other techniques (e.g., *"I just click and keep dragging until I get to the bottom"*, P169). However, preceived deficiencies of current implementations may prompt users to consider other techniques (regarding edge-scrolling, P203 commented *"the only expectation is that it will be painful!"*), either as alternatives or as complements, or even to avoid edge-scrolling altogether.

---

[10]We chose thematic analysis over other methods for several reasons. Instead of merely acknowledging patterns, for example by counting and correlating coding units, we wanted to get a deeper understanding of current edge-scrolling use and the underpinning reasons for the observed patterns. However, we did not want to involve a much heavier procedure, such as grounded theory (Glaser and Strauss, 1967), which would require an iterative process and would not suit our data collection method and research goals. Thematic analysis provided a middle ground combining an accessible and flexible method with an interpretative analysis.

[11]Note that some of the extracts from respondent comments presented here have been translated from French to English.

When edge-scrolling is found inadequate for the task at hand, another method is sometimes used instead: *"when it goes too fast I use one of the keyboard based methods previously mentioned"* (P2). Inadequacy can stem from specific conditions, such as lack of control, making alternatives more compelling (*"the wheel is when I use the mouse and edge-scroll isn't fast enough"*, P175), but it can also occur when the implementation is deemed faulty (*"in some heavy PDFs or web pages, it doesn't work very well and it's simpler to use the keyboard"*, P110).

Users may also consider combining edge-scrolling with other techniques in the course of their action, for example to compensate the lack of resolution (*"complement mouse selection with keyboard selection"*, P18) or speed (*"if I realize that the end of the zone I want to select is really far away, then I switch to the wheel"*, P177).

AVOIDANCE

Repeated inadequacies may drive users away from edge-scrolling (*"it's something I often encounter but try to avoid where possible"*, P62; *"I stopped using it"*, P170). However, in specific situations, edge-scrolling remains the only option: *"I try and avoid it, but sometimes it's not possible to avoid, e.g., when moving a bookmark in my list"* (P203). Interestingly, zooming was mentioned as a way to avoid or reduce edge-scrolling: *"zoom out to scroll more easily"* (P17).

CONTROL

Generally, users noticed the lack of control over scrolling velocity: *"You need to have some control over the speed"* (P168); *"sometimes it scrolls incredibly quickly which is difficult to control"* (P66). Scrolling velocity was found inconvenient when too high (*"it's really annoying when it's too fast"*, P4) or too low (*"it is soooo slow …"*, P81). While overall descriptions of scrolling speed were not sufficient to pinpoint why users lacked control, more specific accounts allowed us to determine three main reasons: limitations to the transfer function, ambiguous activation conditions, and a limited scrolling area.

TRANSFER FUNCTION

The amount of control may be inadequate when the maximum attainable velocity is limited (*"the maximum speed is way too slow when you want to do a large selection"*, P82), or when the minimum velocity is too high (*"there is no way to limit the scrolling speed"*, P186), turning navigation into a slow, *"robotic descent"* (P61).

The failure to support a sufficient transition between scrolling speeds also indicates limitations in the transfer function: *"Always a big jump between very slow motion (before crossing the edge) and hyper fast motion (as early as I cross the edge)"* (P20).

ACTIVATION CONDITIONS

Delayed activation of edge-scrolling may be helpful to prevent undesired scrolling, but it can break the flow of interaction by making users wait too long for activation: *"Having to wiggle the mouse/pointer to get the application to realise that is what you want to do"* (P141). Anticipating when and whether a delayed activation will happen is not easy: *"it often takes some time for the scrolling to begin, but once it does its speed is too fast to easily control"* (P62). Indeed, when entering the scrolling area, there is no feedback allowing the user to discriminate between an ongoing activation and a failure to locate the area. Therefore, when activation finally occurs, the user may already be so far away from the edge that the system starts edge-scrolling at a high velocity.

SCROLLING AREA

The effective size of the scrolling area is critical to the amount of scrolling control: *"there isn't enough room left to use the technique (scrolling is ridiculously slow)"* (P26).

Small scrolling areas by design may make it difficult for users to operate within their boundaries without visual guidance: *"you have to find the edge to start scrolling in a single pixel height line, or in a very small area. If you pass that zone, you have to come back"* (P197).

A scrolling area can also become smaller due to, for example, a maximized window: *"when a window is full-screen, there's not enough room to use the technique"* (P71). In order to recover control, users may be compelled to configure window placement specifically to assist edge-scrolling: *"I tend to move the active window"* (P62); *"reduce the size of the window so that it is as far as possible from the edge of the screen and enjoy greater speeds"* (P15).

PERFORMANCE

Users have various strategies to optimize performance, including deliberately trading off speed for accuracy: *"my solution [...] is to try to edge-scroll closer to the window and try not to make it scroll too fast"* (P18). Such strategies are sometimes executed too conservatively, which then defeats the purpose: *"moving the mouse very slowly -> but loss of the utility of the functionality"* (P183). Suboptimal performance may be caused by overshooting and disorientation issues.

OVERSHOOTING

Due to its very nature, edge-scrolling offers limited foresight, making overshooting almost mandatory, which requires users to reverse actions once the target has been brought into view: *"I select too large a part, then I have to unselect some parts"* (P92). Large amounts overshooting are typically caused by excessive scrolling speed (*"too fast, too far"*, P102), but excessive slowness may also cause task disengagement: *"I eventually fall asleep and overshoot my target"* (P81).

SPATIAL AWARENESS

Lack of spatial awareness (*"completely losing myself in the page"*, P105) could also contribute to suboptimal performance due to the additional search time. The fear of getting lost may compel users into limiting their performance, for example by scrolling in fits and starts: *"you have to stop regularly to check where you are in the document"* (P213). Current systems have less than adequate support for spatial awareness, which can help avoid disorientation and anticipate the scrolling movement: *"I would also like to have some sort of preview of where I am in the document so that I don't overshot the target"* (P91).

HETEROGENEITY

In everyday life, users face large varieties of edge-scrolling behaviors and are generally sensitive to their details (*"different control of speed between applications"*, P55). For example, some respondents reported particular issues with Excel, some commenting it was too fast (*"too fast with Excel, I find myself at the end of the sheet without noticing"*, P2), and others stating it was too slow (*"in excel, when there are very long columns, it is often too slow"*, P74). This discrepancy might be explained by a change between versions of Excel, which was also noted by two respondents: *"Excel used to be very sensitive. It's not so bad any more."* (P229). We formed two secondary themes related to heterogeneity: consistency and familiarity.

CONSISTENCY

External inconsistency was mostly noted in general terms: e.g., *"too fast or too slow depending on the application/site/computer"* (P115). We found a general desire for *"more consistency, and adapted speed"* (P2) across the whole system.

Inconsistent behavior is indeed an explicit issue for users, mainly because because it works against their expectations: *"when I copy-paste between Word files and HTML pages the speeds are generally different, even null in some editors, and it becomes frustrating"* (P175). This leads to the development of systematic behavior testing strategies, such as *"moving backwards and forwards as how far you have to move the mouse is not predictable across applications"* (P141).

FAMILIARITY

The latter strategy could also prove helpful to build a mental model of an unfamiliar behavior. However, it might not be enough for some of them, which can be unfathomable for some users: *"I don't understand how it works"* (P170). Respondents requested more familiar designs—*"a more intuitive use"* (P183)—and referred to common transfer functions in their recommendations: *"I would like that the speed be systematically related to the distance of the cursor relative to the bottom of the window"* (P177).

6.4.4. SUMMARY

The survey results, drawn from the responses of generally experienced users, show that edge-scrolling is a widely used and important facility that is worth supporting and improving. Section 6.4.2 revealed that edge-scrolling was one of the most used scrolling techniques during manipulation, independently of the preferred pointing device. It was also largely used for the selection task (Section 6.4.2). Besides, most respondents knew the technique well enough to be able to produce subjective assessments (Section 6.4.2), as well as comment their use for some of them (Section 6.4.3). But beyond the question of whether edge-scrolling is used, it is also useful to know when it is used.

The first two themes of our thematic analysis (Section 6.4.3) explain how users select a scrolling technique during a manipulation task. Experienced users draw on their practical knowledge to adapt their strategy to particular applications, types of objects, devices, or distances, and change techniques accordingly. Interestingly, technique selection also depends on their initial knowledge of the endpoint location. As the first themes suggests, there is no overall "best technique" and context shapes users' strategies (Mackay, 2002). In addition, intrinsic weaknesses of a particular instance of the technique can compel the user to either switch to another technique along the way, or to consider not using it anymore. This was demonstrated for edge-scrolling in the second theme. The order difference in scrolling technique use between mouse and touchpad users (Section 6.4.2) also suggests this for *wheel*: as the scrolling gesture is hard to perform on a touchpad, users may rely more on edge-scrolling when using this kind of device.

This survey does not suggest any agreement on frustration, but subjective assessments (Section 6.4.2) revealed control issues and sensitivity to change of edge-scrolling behaviors. Furthermore, the comments left by respondents provided the basis for a more detailed account of interaction. Frustration was rarely explicitly elicited, but we could formulate three themes relevant to edge-scrolling design: control, heterogeneity, and performance (Section 6.4.3).

6.5. Discussion

Through our investigations of current design practice and use in this chapter, we have uncovered important insights on edge-scrolling. Edge-scrolling is supported in various ways, leading to very different opportunities for control across different applications, and most existing implementations share a similar stance towards equivocal *move* tasks, trading off expressive power for predictability. Accounts of edge-scrolling use show that perceived lack of control originates from all three dimensions of the design space (scrolling area, transfer function, and activation conditions), that inconsistent behavior throughout the system prevents transparent use of the technique, and that poor perceived performance can be linked to all these aspects. In this section, we will discuss the limitations of the reverse-engineering approach used to analyze existing designs, and we will propose an extension of the design space of edge-scrolling techniques based on the survey results.
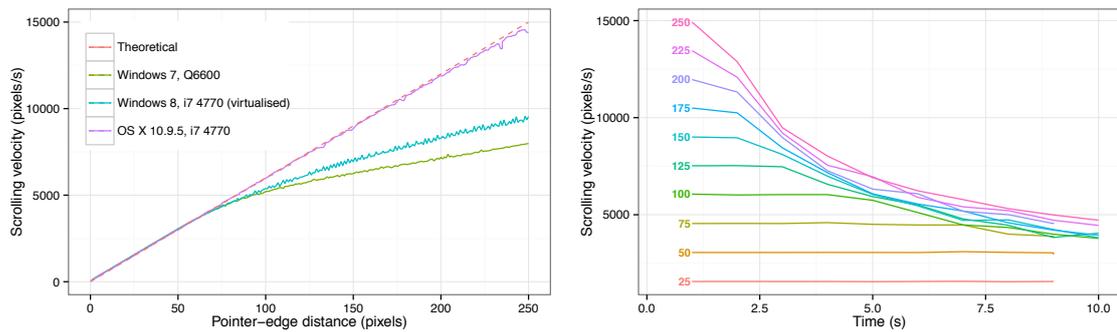
FIGURE 6.14: Variations of edge-scrolling velocity in Chrome$_{select}$. On the left, scrolling velocity as a function of pointer-edge distance for a 10 second edge-scrolling interaction, measured with different system configurations. On the right, scrolling velocity as a function of time, which varies with different pointer-edge distances (line labels, in pixels) due to accumulated system load (Windows 7, Q6600).

6.5.1. ISSUES IN REPLICATION AND REVERSE-ENGINEERING

Reverse-engineering the behavior of edge-scrolling viewports can bring valuable information when studying existing designs, but precise reproduction of the observed behaviors is rarely possible due to several limitations. First, using accessibility APIs is more amenable to imprecise measurements compared to scripts and programs due to the inter-process and non-critical nature of accessibility implementations (e.g., Finder or Photoshop in Figure 6.7, left). Second, parameters that possibly have an effect must be tested manually, therefore it is easy to overlook a critical parameter. For example, according to the source code of the `QTextEdit::timerEvent` method, edge-scrolling velocity in QTextEdit$_{select}$ does not always depend on the distance between the pointer and the bottom edge of the viewport. In fact, the input parameter is a pointer-edge distance, but the viewport edge that is considered for this distance is the farthest one from the pointer. Therefore, as we focused on downwards scrolling using the bottom edge of the viewport, we did not model this behavior completely.

The issues with this *black box* approach could indeed have been avoided by analyzing the source code of applications, but an overreliance on source code may be worse for proper understanding and replication of edge-scrolling mechanisms. This approach is seldom possible with commercial software, and requires to properly understand the interactions between subsystems in a large code base. Moreover, this source code is only part of the causes that engender an observed edge-scrolling behavior. Desktop functions typically use non real-time low-resolution timers to implement interface functions, including rate-based scrolling, which is achieved by performing discrete scrolling displacements of a given amplitude at a given frequency. But when frequency exceeds timer resolution, scrolling velocity becomes less reliable.

In some cases, there could even be larger differences between the theoretical transfer function extracted from source code and the actual behavior. To illustrate this, we analyzed the source code

of the Blink web engine powering Chrome and compared it to several recordings of Chrome$_{\text{select}}$ with different system configurations (Figure 6.14).

Discrete edge-scrolling displacements occur each time the `AutoscrollController::animate` method is called, calling in turn the `RenderBox::autoscroll` method that produces a scrolling displacement equal to the pointer-edge distance. The `animate` method is called by the rendering system at each frame. Theoretically, with a best case 60 Hz frame rate, scrolling velocity should approximate $f(x) = 60x$, x being the pointer-edge distance. However, as pointer-edge distance increases, there is more content to render, and when the system has not enough computational power to keep up, scrolling velocity decreases below the theoretical level (Figure 6.14, left), reflecting the accumulation of rendering load over time (Figure 6.14, right). Therefore, by influencing the edge-scrolling behavior in an inconsistent way, the system configuration and the contents of the page could indirectly affect user performance.

Thanks to the project's issue tracker, the engineering rationale for this implementation has been documented. Coupling rate-based edge-scrolling to frame rate allowed cleaner code by avoiding timer management, but more importantly, contributors felt that the previous implementation, updating at 20 Hz using a dedicated timer, was not responsive enough. Interestingly, the previous implementation was that of the WebKit engine, from which Blink forked, and is still used by Safari. In theory, the modifications made to Chrome$_{\text{select}}$ should provide three times faster scrolling than Safari$_{\text{select}}$, but in practice scrolling velocity is unstable. A solution to keep consistent behavior when achieving high scrolling velocities would be to use a dedicated timer with lower timer frequency and increase the gain factor on discrete scrolling displacements to compensate.

6.5.2. VISUAL GUIDANCE

Some of the problems identified through thematic analysis of the survey respondent comments can be solved by displaying appropriate visual information to indicate when edge-scrolling can be used, how to activate it, whether it is active or not, and how to control it. This information is critical for understandability and controllability, yet it remains largely unexploited in current designs. Scrolling movements, and possibly the scrollbar, already provide strong visual cues on changes in viewport position inside the document, but additional visual guidance can be offered. *Visual guidance* represents a new dimension in the design space of edge-scrolling technique that can affect *awareness*, *feedback*, and *anticipation*.

First, awareness of the edge-scroll functionality can be improved with signifiers and feedforward (Vermeulen et al., 2013). For example, awareness can be provided by visually highlighting scrolling areas when the pointer is in their vicinity, and by displaying arrows to indicate scrolling direction (Figure 6.15a). Note that a similar form of awareness is already used in long menus (Figure 6.4d), but it could be extended to any edge-zone to relieve users from guessing the extent of the scrolling area.
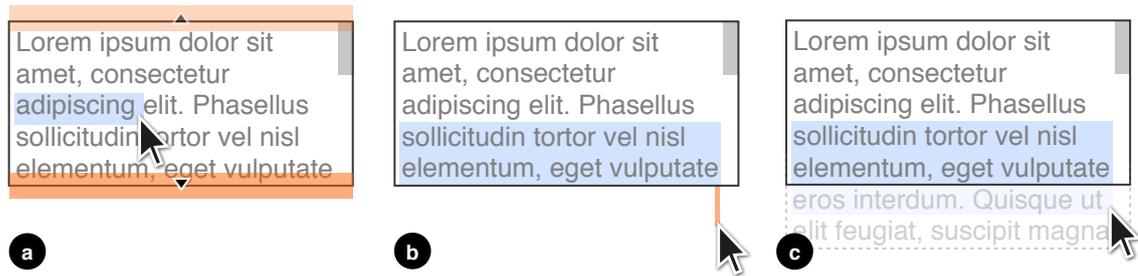
Figure 6.15: Edge-scrolling designs can use visual guidance: (a) to provide *awareness* of the availability of the technique and its properties, such as the scrolling direction and the extent of the scrolling area; (b) to provide *feedback* on control parameters (the length of the orange line represents scrolling velocity); and to support *anticipation*, for example by displaying parts of the document outside the viewport during edge-scroll.

Second, more visual feedback may help the user understanding what she is actually controlling. For example, when the pointer-edge distance is used to control scrolling velocity, displaying the distance using a line may provide a better visualization (Figure 6.15b), especially when the scrolling area is not adjacent to the considered viewport edge. This establishes a *display invariant* between user input and scrolling output that contributes to the *tight coupling* (Ahlberg and Shneiderman, 1994b) between the edge-scrolling technique and the document displayed in the viewport. In addition, visual feedback can provide confirmation that secondary activation conditions (e.g., a button press) have been correctly fulfilled.

Third, the user's ability to anticipate an upcoming endpoint can be augmented with proper visualization. During edge-scrolling, the height of the viewport determines how much of what is behind is visible, but the user has little to no visibility on what is ahead of her current position in the document. Therefore, any way of displaying the contents lying outside the viewport might help anticipation (Figure 6.15c). But support for anticipation is also welcome when a delay is used to activate edge-scrolling, especially if delays are inconsistent across applications. For example, gradually changing the color of the scrolling area can help the user foresee edge-scrolling activation in a predictable fashion.

6.6. Conclusion

In this chapter, we proposed and studied a design space for edge-scrolling, a technique for increasing the extent of dragging actions by scrolling a viewport when the pointer approaches its edge that has been widely implemented in desktop systems. By reverse-engineering these implementations, we exposed substantial variance in design approaches. Responses to an interactive survey demonstrated that edge-scrolling is broadly used, and that users are aware of problems stemming from lack of expressive power and perceived inconsistencies. After a discussion of the limitations of our reverse-engineering approach, we proposed a new design space dimension, visual guidance, based on our results.

Generating and evaluating edge-scrolling designs

The previous chapter presented a conceptual framework of the factors influencing the design of edge-scrolling techniques, and discussed the most prevalent choices in existing software, as well as their reported use. In this chapter[1], we propose two novel techniques that both are more direct and have more expressive power than existing ones, and evaluate their performance, as well as that of various designs.

Our main concern here is the evaluation of edge-scrolling techniques. Most of the edge-scrolling designs examined in Chapter 6 have two major limits regarding expressive power and directness. First, they offer unsatisfactory control of scrolling due to their scrolling area, which is either dependent on viewport and display geometry for display-bound designs, or restricted to a small portion around the viewport edge for edge-zone designs. Second, they are based on rate control, which is not as *referentially direct* as position control, but might be less difficult to use with small scrolling areas because they do not require as much movement. Regarding other factors, such as performance and perceived effort, little is currently known, but existing scrolling research (e.g., Hinckley et al., 2002) suggests that the transfer function has a major effect.

We propose two novel edge-scrolling techniques, *push-edge* and *slide-edge scrolling*, that eschew the above limitations, and discuss the evaluation of edge-scrolling. We will first discuss the scrolling literature dealing with the respective merits of position and rate control transfer functions. We will then expose the rationale of push-edge and slide-edge scrolling, as well as design details, and compare their performance with a standard rate-based edge-scrolling technique in a one dimensional text selection task combining dragging and scrolling using a mouse and a touchpad. We will further compare five transfer function designs, including push-edge, in another controlled experiment using only a mouse, with results showing the influence of several design parameters on task completion time and overshooting. Finally, based on the combined results of the previous chapter and this one, we will discuss other possible criteria for the evaluation of edge-scrolling, propose general design recommendations, and outline the limitations of our work.

---

[1]The work presented in this chapter has been carried out in collaboration with Sylvain Malacria, Philip Quinn, Géry Casiez, Andy Cockburn, and Nicolas Roussel.
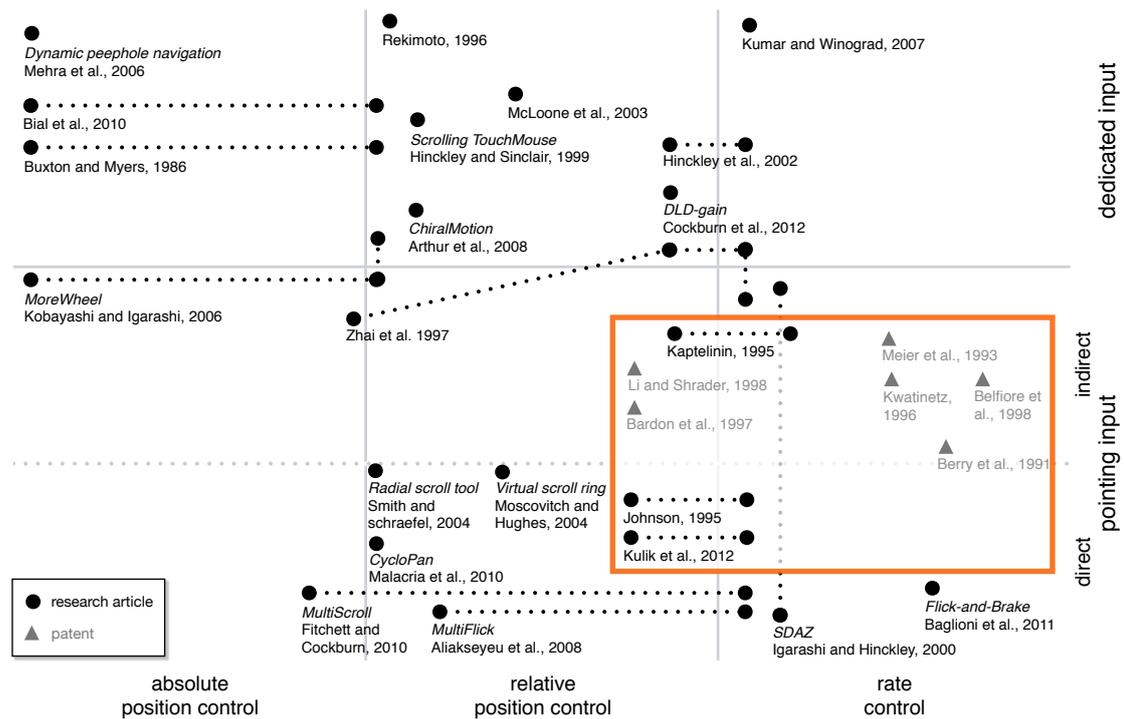
FIGURE 7.1: Overview of scrolling techniques and studies discussed in this chapter and the previous one, by type of input (pointing or dedicated input) and transfer function (absolute/relative position control or rate control). Grey triangles are patents. The orange frame delimits edge-scrolling.

7.1. Related work

The design of scrolling transfer functions has been extensively studied in the literature (Figure 7.1), and has brought many relevant insights that can be used for edge-scrolling technique design. In this section, we review existing works on absolute position control, relative position control, and rate control transfer functions. We analyze the merits of either type of transfer function mapping using the system-level model of Quinn et al. (2012). In this model, a transfer function processes user input to produce scrolling output based on three successive transformations: conversion of the input signal into a control signal in display space units (*translation stage*), modification of the amplitude of the control signal (*gain stage*), and application of effects, such as simulated friction, over time (*persistence stage*).

7.1.1. ABSOLUTE POSITION CONTROL

Absolute position control functions map a point inside an area in a physical or virtual coordinate space to the top left corner of the viewport in document coordinates—they map an input position to a scrolling position. Such functions have been used in dedicated devices (Buxton and Myers, 1986; Bial et al., 2010) and on touchscreens (Fitchett and Cockburn, 2010). They allow users to quickly jump to remote parts of a document, but their main shortcoming is that precise selections and short-distance movements are difficult to perform (Fitchett and Cockburn, 2010). Resolution problems occur at the *translation* stage of the transfer function, mainly for two

reasons: either the number of sampled elements in the input space does not match the number of elements in the document space, or the transfer function itself does not consider the whole input sensitivity (Chapter 4).

7.1.2. RELATIVE POSITION CONTROL

Relative position control functions offer a referentially direct mapping of the user's movements to scrolling displacements—input movement causes document movement. However, the physical or virtual footprint available for input is generally limited in comparison to the large range of scrolling movements, from a few pixels to hundreds of lines of text (Hinckley et al., 2002). Therefore, in long distance scrolling, the user must frequently clutch the device in order to compensate the limited input area. Repeated clutching can compromise the user's ability to maintain a steady scrolling speed, for example when reading (Zhai et al., 1997b). Clutching can also increase scrolling time (Zhao et al., 2014) and require more physical effort (Hinckley et al., 2002), but it can also be performed intentionally as part of the user's strategy to optimize device movements (Nancel et al., 2015).

Scrolling performance and undesired clutching can be improved by each of the three transformations composing the transfer function. At the *translation* stage, mapping circular motion to scrolling (Moscovich and Hughes, 2004; Smith and schraefel, 2004; Arthur et al., 2008) removes the need to clutch, but these mappings take up to two degrees of freedom in order to control one-dimensional scrolling. Instead of a circle, Malacria et al. (2010) proposed to use oscillatory motions along a line as two-dimensional scrolling input. However, this decreases movement compatibility and might be harder to learn compared to linear movements. At the *gain* stage, the manipulation of CD gain can reduce device footprint (Jellinek and Card, 1990), but high gains can decrease resolution unless using a non-linear transfer function that maps low input velocities to low gains and high input velocities to high gains, making both fast and precise movements possible. Non-linear transfer functions have been shown to decrease scrolling time (Hinckley et al., 2002) compared to a proportional gain function. Cockburn et al. (2012) showed that, in addition to velocity, incorporating document length as a key input parameter of the transfer function substantially improved long distance scrolling performance across a variety of scrolling input devices. Finally, at the *persistence* stage, simulated inertia and friction can allow scrolling to continue autonomously while the user is clutching (Quinn et al., 2013).

7.1.3. RATE CONTROL

With rate control, user input controls the velocity of scrolling. The *persistence* stage of rate control transfer functions therefore consists in sustained scrolling, whose rate is determined by the *translation* and *gain* stages. With rate control, scrolling velocity can be maintained smoothly and continuously throughout the scrolling movement (Zhai et al., 1997b; Hinckley et al., 2002). This is used, for example, on many desktop applications after a middle-click on a document (Zhai et al., 1997a), or with augmented scrollbar buttons (Beaudouin-Lafon, 2000). Alternatively,

some rate control functions are controlled by a flicking gesture that sets the initial scrolling velocity (Aliakseyeu et al., 2008), and subsequent presses may cause simulated friction, allowing continuous control of the scrolling speed throughout the scrolling movement (Baglioni et al., 2011). Regardless, controlling an acceptable range of velocities does not require as much footprint as relative position control. Rate control also requires less physical effort (Zhai et al., 1997b), but can be difficult to control precisely (Hinckley et al., 2002). In addition, Igarashi and Hinckley (2000) remarked that beyond a certain velocity, rate-based scrolling caused disorientation. They proposed to compensate the increase of scrolling speed by decreasing the scale at which the document is displayed in order to ensure a constant visual flow.

In summary, the existing literature on scrolling transfer functions has dealt extensively with the drawbacks and benefits of each type of control. Some works have even proposed or studied hybrid transfer functions combining absolute and relative position control (Buxton and Myers, 1986; Kobayashi and Igarashi, 2006), relative position and rate control (Aliakseyeu et al., 2008), or absolute position and rate control (Fitchett and Cockburn, 2010). Figure 7.1 recapitulates the scrolling techniques reviewed in this chapter and the previous one, organized by transfer function and type of input.

7.2. Push-edge and slide-edge scrolling

In the previous chapter, we have uncovered two major characteristics of existing designs. First, their transfer functions uses rate control, mapping the pointer-edge distance to scrolling velocity. This was the case for 84% of the surveyed designs. And second, the scrolling area was display-bound for *select* tasks, while it was a small edge-zone for *move* tasks in order to avoid unintended activation. In both cases, control is frequently suboptimal due to the small size of the scrolling area.

This section presents two novel designs, *push-edge* and *slide-edge scrolling*, that challenge those common choices in edge-scrolling design and propose that edge-scrolling be performed as if by pushing the pointer against the viewport edge to move the underlying document. We will discuss three defining aspects of these new techniques: the use of position control rather than rate control, the independence from scrolling area size, and the scalability to long distances.

7.2.1. POSITION CONTROL AND RATE CONTROL WITH DESKTOP INPUT DEVICES

As discussed in the previous section, the use of position control versus rate control in scrolling has been extensively debated in prior research. Rate-based methods are most desirable when the user's primary task is reading, because it is easier to maintain a constant scrolling speed (Zhai et al., 1997b), but edge-scrolling is used in more target-directed activities because it primarily exists as an extension of dragging actions. Rate control is also preferred when input is provided through an isometric device (Zhai, 1995), such as a joystick, essentially because the self-centering force of these devices facilitates the frequent reversal of movement direction required by rate

control. However, desktop and laptop pointing devices (mice and touchpads) are mainly isotonic, directly transforming device displacements into on-screen displacements, but the lack of self-centering effect requires explicit oscillatory movements for adjusting the rate (Kim et al., 1987). Thus, the omnipresence of rate-based edge-scrolling techniques is surprising, as there is support for the design of position-based techniques.

7.2.2. INDEPENDENCE FROM SCROLLING AREA SIZE

But in existing implementations, the size of the scrolling area limits the maximum amplitude of edge-scrolling movements in position control, or the maximum scrolling velocity in rate control. In *move* tasks, the scrolling area is an edge-zone to avoid unintended scrolling, but its small size limits expressiveness. And when the scrolling area is display-bound, as is frequently the case in *select* tasks, it is dependent on screen space, which is likely scarce when the window is maximized, or when using a small display. Either way, a small scrolling area is more likely to make edge-scrolling hard to control for the user.

Position control has an edge with respect to this issue. Because the movement of the pointer is made redundant by the visual scrolling feedback, the pointer can be blocked at the edge when entering the scrolling area, so that every subsequent movement will scroll the viewport instead of moving the pointer, effectively having a zero-sized scrolling area that does not impede scrolling control. This is similar to the strategy employed in the invention of Li and Shrader (1998), except that their design required explicitly hitting a key to stop capturing the pointer at the edge, which might be frustrating in case of accidental activation.

Push-edge and *slide-edge scrolling* (Figure 7.2) extend this design by only scrolling and blocking the pointer for movements towards the exterior of the viewport, and by releasing the pointer when the user moves the device back towards the interior of the viewport. When the pointer crosses the viewport edge, it is captured at the edge, and subsequent pointing input that would otherwise move the pointer is mapped to scrolling through the transfer function: further movements away from the viewport result in scrolling displacements of the document, while the pointer remains captured in a fixed position at the edge. The captured pointer is released either by moving it back towards the viewport, or by releasing the dragging state. In effect, the user pushes against the viewport edge to reveal other parts of the underlying document. Related metaphors have been employed successfully in the design of radial menus (Fitzmaurice et al., 2003), and of focus+context interfaces (Appert et al., 2010).

7.2.3. SCALING TO LONG DISTANCES

The range of a pointing movement, from tenths of a millimeter (Chapter 5) to the size of a screen, is much narrower than the range of scrolling movements, from a few lines to hundreds of pages (Cockburn et al., 2012). Consequently, scrolling by pointing is not likely to scale to long distance scrolling unless using a non-linear transfer function. In *push-edge scrolling*, a raw displacement of the pointing device in motor space (d, in mm) is converted into viewport
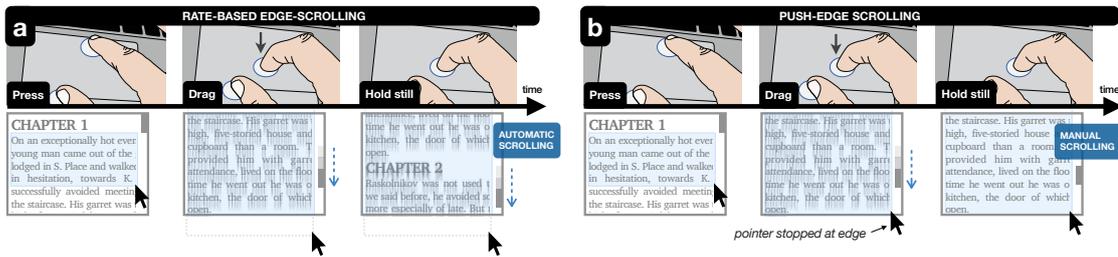
FIGURE 7.2: When selecting text with a touchpad, downward movements after crossing the edge of the viewport will (a) change the rate of automatic scrolling with rate control techniques; or (b) manually scroll the document, stopping the pointer at the edge, with *push-edge scrolling*, the proposed position control technique. Slide-edge scrolling performs similarly but applies inertia once the user lifts the finger.

scrolling in display space (D, in pixels) by multiplying d with the outcome of a transfer function g, and by converting the result in a pixel distance using the pixel density of the display. The non-linear transfer function g produces a unitless control-display gain from device velocity (v, in m/s):

$$g(v) = \begin{cases} g_{min} & : v \leq v_1 \\ g_{min} + (g_{max} - g_{min}) \times \frac{v - v_1}{v_2 - v_1} & : v \in]v_1; v_2[\\ g_{max} & : v \geq v_2 \end{cases}$$

As a touchpad and a mouse are controlled using different limb segments and perform different ranges of motion in motor space, we adapted the parameters of the transfer function for each type of device:

Mouse:	Touchpad:
$g_{min} = 1$	$g_{min} = 1$
$g_{max} = 10$	$g_{max} = 10$
$v_1 = 100\ mm/s$	$v_1 = 200\ mm/s$
$v_2 = 500\ mm/s$	$v_2 = 500\ mm/s$

However, even with a non-linear transfer function, we expected large differences in physical effort between rate-based techniques and push-edge scrolling, in particular due to clutching. The particularly long distances involved in scrolling compared to normal pointing are likely to inflate the impediment caused by repeated clutching. To try and reduce the amount of clutching, we devised *slide-edge scrolling*, a variant of push-edge scrolling that applies inertia to continue scrolling after the user stops operating the pointing device based on a simulation of

residual momentum (Quinn et al., 2013). When the motor-space velocity at liftoff v_l exceeds 150 m/s, friction progressively decreases velocity over time according to the following function: $f(t) = v_l \exp^{-4t}$, where t is the time elapsed since liftoff, in seconds. Thus, when clutching, the view continues to scroll but the user can instantly regain control of scrolling just by moving the device again, similarly to GlideCursor techniques for pointing (Beaudouin-Lafon et al., 2014).

7.3. Evaluation of push-edge and slide-edge scrolling

We performed a controlled experiment comparing performance, degree of control, and perceived workload between push-edge scrolling (PUSH), slide-edge scrolling (SLIDE), and the default OS X rate-based edge-scrolling technique (RATE), using a mouse and a touchpad on a one-dimensional text selection task. The experiment tested the four following hypotheses:

H1 Selection time is lower for PUSH and SLIDE than for RATE.

> Push-edge and slide-edge scrolling were designed to overcome the limitations of rate-based edge-scrolling discussed above, and are expected to permit quick and accurate scrolling for a wide range of scrolling distances.

H2 Inertia (SLIDE) reduces selection time compared to PUSH.

> To take advantage of the nonlinear transfer function of PUSH, the user must perform repeated ample movements with high velocity in one direction to navigate towards the target. However, once it is brought into view, a comparatively small pointing movement must be performed in the opposite direction, towards the target. This could result in longer selection times, because the user must reverse movement direction and shift her motor strategy from speed to accuracy. With inertial scrolling, user movement is much shorter because most of the scrolling motion occuring is autonomous. Therefore, less force is likely required to perform movement reversal when switching from scrolling to pointing.

H3 PUSH and SLIDE lead to less overshooting than RATE.

> Due to the lack of self-centering force in the mouse and the touchpad, rate-based edge-scrolling must be stopped by explicitly moving the pointer back to the viewport. During this movement, the viewport continues scrolling. In contrast, stopping position-based edge-scrolling is instantaneous.

H4 Inertia (SLIDE) reduces physical effort for position control, compared to PUSH.

> The repeated clutching that may occur with position control techniques requires more physical effort (Hinckley et al., 2002). The negative effect of inertia on the amount of undesired clutching has been demonstrated in pointing tasks (Beaudouin-Lafon et al., 2014). Introducing inertia could consequently reduce physical effort in edge-scrolling.

To avoid any ambiguity related to inter-viewport interaction, the experiment used an intra-viewport *select* task with display-bound edge-scrolling. The task has been repeated across a large range of scrolling distances to be representative of the variety of scrolling tasks.
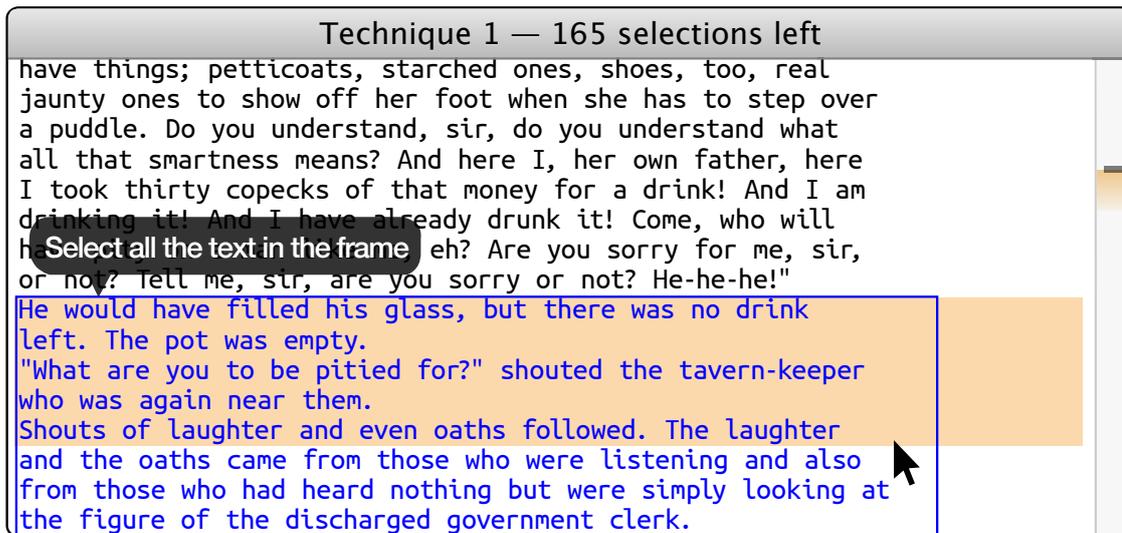
FIGURE 7.3: The experimental software used for the one-dimensional text selection task.

7.3.1. METHOD

The experiment was conducted on a 13" MacBook Pro Retina running OS X 10.9.4, with display resolution set to *Best* (1280×800). Depending on the condition, participants either used the embedded touchpad of the laptop, or Logitech M90 optical USB corded mouse (sensitivity 1000 CPI) on a plywood desk. The pointing transfer function was set to the fourth tick of the *Tracking Speed* slider in the respective touchpad and mouse system preference panels. The custom experimental software was written in Objective-C with Cocoa, and implemented push-edge and slide-edge scrolling as described above. The software monitored raw device input using the I/O Kit and Apple's private multitouch API, and it blocked the system pointer using the Quartz Event Services API. The baseline rate-based edge-scrolling technique (RATE) was NSTextView$_{select}$, which is the default technique in all Cocoa scroll views (defined in the `autoscroll:` method of the `NSClipView` class).

TASK

After completing a demographic questionnaire, participants were instructed to perform a sequence of top-to-bottom text selections as quickly and accurately as possible. The software displayed a 400 pixels tall viewport in the center of the display, containing 21,480 lines of text typeset with *Ubuntu Mono Regular* 13px on which selections were performed (Figure 7.3).

For each trial, participants had to successfully select a section of text framed and colored in blue, starting eight lines from the bottom of the viewport, and varying in size throughout the experiment. A gradient in the scrollbar cued the approximate size of the portion of text to select. As the task was one-dimensional, selecting anywhere on the line always selected the entire line. Every text selection required concurrently dragging and scrolling using the requested device and the requested edge-scrolling technique. Other scrolling methods and devices were disabled.

To perform a selection, participants had to position the pointer at the top of the blue portion of text, press the pointing device to start the selection, entering a dragging mode, scroll down the viewport using the requested edge-scrolling technique until they reached the bottom of the blue portion, position the pointer over its last line, and release the button to confirm the selection. If the selection did not exactly capture the blue portion of text, a selection error occurred, the viewport was scrolled back to its initial position, and the trial had to be started over.

DESIGN AND PROCEDURE

The experiment used a $2 \times 3 \times 3 \times 4$ within-subjects design, with the following factors: DEVICE (MOUSE and TOUCHPAD), TECHNIQUE (PUSH, SLIDE, and RATE), BLOCK (1-3), and SIZE (S_{15}, S_{45}, S_{135}, and S_{405} lines of text to be selected). The presentation orders for DEVICE and TECHNIQUE were counterbalanced across the participants. SIZES were presented from the shortest to the longest one for each BLOCK, with 5 consecutive repetitions for a given size within a block, making a total of $2 \times 3 \times 3 \times 4 \times 5 = 360$ trials per participant. Participants were encouraged to take a short break after each block, and they completed a NASA TLX form after each technique. The experiment lasted approximately 45 minutes.

PARTICIPANTS

Twenty-four unpaid volunteers (mean age 30.4, SD 7.0, 19 males) participated in the experiment. Most were heavy computer users (more than six hours daily for 75% of them), used a laptop as their primary computer (96%), and used a touchpad as their primary pointing device (58%).

7.3.2. RESULTS

The dependent variables were selection time, overshoot distance, and the number of clutches. For the analysis we excluded error trials (6.2%). We also excluded the first repetition of each SIZE, as size changes were not immediately noticed. For all repeated-measures ANOVAs, we corrected the degrees of freedom using Greenhouse-Geisser estimates of sphericity when the assumption of sphericity was violated, and pairwise comparisons used Bonferroni correction.

SELECTION TIME

Selection time is defined as the time taken to select the target, from the first device movement after the left mouse button was pressed, to the button release.

A repeated-measures ANOVA revealed a significant effect of BLOCK on selection time ($F_{2,46} = 28.0$, $p < .001$, $\eta_p^2 = .55$; block 1: 4.0s; block 2: 3.7s; block 3: 3.6s), with a significant TECHNIQUE $\times$ SIZE interaction ($F_{4,92} = 3.0$, $p = .023$, $\eta_p^2 = .11$) due to a learning of the new technique behavior during the first block. As we are concerned with user performance after familiarization, the remaining analysis discards the first block.

We found significant main effects of DEVICE ($F_{1,23} = 28.7$, $p < .001$, $\eta_p^2 = .56$), TECHNIQUE ($F_{2,46} = 6.9$, $p = .002$, $\eta_p^2 = .23$), and SIZE ($F_{1.1,25.6} = 360.8$, $p < .001$, $\eta_p^2 = .94$) on selection time, as well as a significant TECHNIQUE $\times$ SIZE interaction ($F_{2.4,55.3} = 4.2$, $p = .014$, $\eta_p^2 = .16$).
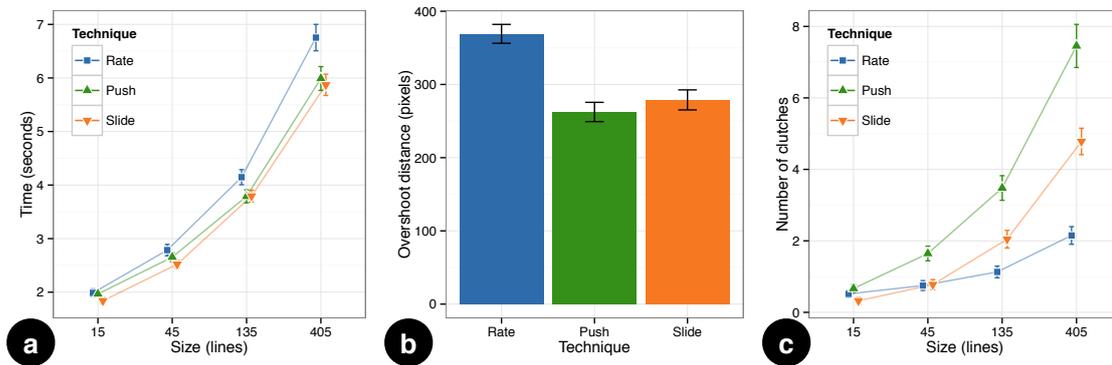
FIGURE 7.4: (a) Selection time by size and technique; (b) overshoot distance by technique; (c) mean number of clutches for the touchpad by size and technique. Error bars represent 95% confidence intervals.

Post-hoc tests revealed that selection time was significantly lower for SLIDE than for RATE with S_{15} ($p < .010$), and than for RATE with S_{45} ($p = .004$). RATE significantly increased selection time compared to both PUSH and SLIDE with S_{135} ($p < .015$) and S_{405} ($p < .030$).

Although inertia had virtually no effect on performance, thus undermining **H2**, PUSH and SLIDE consistently outperformed RATE by up to 13% in the longest size (Figure 7.4a), supporting **H1**.

OVERSHOOT DISTANCE

The overshoot distance is defined as the maximum scrolling distance reached during the trial, relative to the target.

A repeated-measures ANOVA revealed a significant main effect of SIZE ($F_{1.4,31.0} = 106.1$, $p < .001$, $\eta_p^2 = .82$) on overshoot distance. More interestingly, the effect of TECHNIQUE was also significant ($F_{2,46} = 66.7$, $p < .001$, $\eta_p^2 = .74$, Figure 7.4b), and we found no significant interaction effect. Pairwise comparisons showed that there was significantly more overshoot ($p < .001$) with RATE (369px) than with PUSH (262px) and SLIDE (279px).

In order to stop scrolling when the target has been found with RATE, one must travel back towards the viewport, while PUSH and SLIDE do not have this problem because the pointer is blocked at the window edge, giving more immediate control: **H3** is supported.

NUMBER OF CLUTCHES ON THE TOUCHPAD

We analyzed the number of clutches used on the TOUCHPAD, assuming that frequent clutching indicates high physical workload.

A repeated-measures ANOVA showed significant main effects of TECHNIQUE ($F_{2,46} = 47.3$, $p < .001$, $\eta_p^2 = .67$) and SIZE ($F_{1.3,30.7} = 172.4$, $p < .001$, $\eta_p^2 = .88$) on the number of clutches, as well as a significant TECHNIQUE × SIZE interaction ($F_{2.2,50.2} = 28.2$, $p < .001$, $\eta_p^2 = .55$).

Pairwise comparisons showed that PUSH yielded significantly more clutches than SLIDE with S_{15} ($p = .003$), and than SLIDE and RATE with S_{45} ($p = .002$). Additionally, with S_{135} and S_{405}, all differences are significant ($p < .006$), with the following order: RATE < SLIDE < PUSH. As RATE allows continuous scrolling to be maintained without movement, it is unsurprising that it has significantly less clutching than PUSH. However, as Figure 7.4c shows, the addition of inertia decreases clutching as selection size increases, thus giving support to **H4**.

PERCEIVED WORKLOAD

We used separate Friedman tests for either device to analyze the NASA TLX scores of the *physical demand* and *effort* subscales.

We found significant differences between techniques on *physical demand* with the mouse ($\chi^2(2) = 17.2$, $p < .001$) and the touchpad ($\chi^2(2) = 11.9$, $p = .003$). Pairwise comparisons showed that RATE was less physically demanding than PUSH and SLIDE for the mouse ($p < .001$), and than PUSH ($p = .013$) for the touchpad. This clearly reflects the difference between position and rate control.

For the mouse, we also found significant differences on *effort* ($\chi^2(2) = 7.7$, $p = .022$), with pairwise comparisons showing that SLIDE required more effort than RATE ($p = .019$). This probably stems from the difficulty of lifting the mouse while maintaining a high tangential velocity. Therefore, clutching and subjective data only partially support **H4**.

7.3.3. DISCUSSION

Our key finding is that push-edge and slide-edge scrolling both improved text selection time (**H1**) and control (**H3**) over NSTextView_{select}. The performance benefits were consistent for both the mouse and the touchpad. We did not find an effect of inertia on performance (**H2**), but it had an effect on physical effort commensurate with selection size, although slide-edge scrolling required more effort with the mouse (**H4**). Overall, our results tend to suggest that push-edge scrolling is best combined with mouse input, and slide-edge scrolling best with a touchpad.

It is likely that the actual performance benefits for push-edge and slide-edge scrolling over rate-based methods are understated with these results. The experiment intentionally used a relatively small window centered on the screen, which is a best-case scenario for traditional techniques because of the large scrolling area outside the window. When a window extends to the edge of the display (e.g., when maximized), part of the scrolling area becomes unavailable. This situation substantially impairs traditional methods, but there is no reason to anticipate any performance detriment with push-edge and slide-edge.

The baseline rate-based technique we chose for the experiment was the default one used in OS X, thus it is standard and present in many applications. But, as the previous chapter has shown, there is a variety of rate control transfer function designs, with no empirical data to assess their differences. The following experiment will show that while push-edge and slide-edge scrolling outperform the standard OS X behavior, some rate-based designs can compete with them in long selections.

7.4. Evaluation of common transfer functions

In the previous chapter, we found that the various edge-scrolling transfer function designs could impact reported user performance and impression of control in different ways. To further investigate this effect, we conducted a controlled experiment to measure and compare performance and perceived workload when completing one-dimensional text selection tasks using the previous push-edge technique, as well as four different transfer functions representative of diverse approaches to edge-scrolling.

7.4.1. METHOD

We used the same experimental software as in the previous experiment on a 13" MacBook Air using a 22" external monitor with resolution 1680×1050 pixels and pixel density 90 PPI. The input device was a USB corded Logitech M90 mouse with sensitivity 1000 CPI and was used on a varnished plywood desk. The pointing transfer function was set to the fourth tick of the *Tracking Speed* slider in the mouse preference pane of the System Preferences application.

TASK AND TECHNIQUES

The experiment used the same one-dimensional text selection task as described in the previous experiment (Section 7.3.1, Figure 7.3).

Participants completed the task with five different edge-scrolling techniques, including four with infinite scrolling areas[2] and the previous push-edge technique (PE) where the pointer is blocked at the viewport edge. Among the five techniques, three were replicated from the previous reverse-engineering study (Section 6.3): a position based technique, $GTKTextView_{intra}$ (GTK), and two linear rate-based ones with different parameters, $NSTextView_{select}$ (NSTV), and $Safari_{select}$ (WK, as it used the WebKit engine). We checked our implementations using both the reverse-engineering data and the source code from original implementations.

The fifth rate-based technique, NL (non-linear), was designed to study a wider range of scrolling velocities. The transfer function was defined as $f(x)$, in pixels/s, with x the pointing-edge distance in pixels, such that:

$$f(x) = \begin{cases} 10x & : x \leq 100 \\ 0.1x^2 & : x > 100 \end{cases}$$

Informal tests suggested that linear functions with slopes similar to $Firefox_{select}$ or $Finder_{select}$ provided an ample output domain, allowing to quickly scroll through long distances, but seemed too sensitive in the correction phase. We designed NL so that it kept a gentle slope in the first third of the scrolling area in order to facilitate scrolling on shorter distances, while being able to produce scrolling velocities up to 9000 px/s in the last third (Figure 7.5).

---

[2] A 300 pixels high scrolling area was available for edge-scrolling in this experiment.
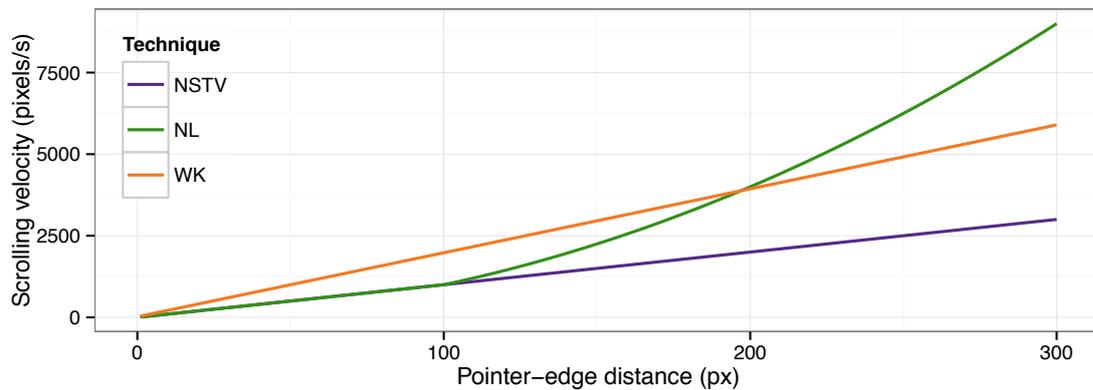
Figure 7.5: Scrolling velocity by pointer-edge distance for the three rate-based techniques used in the experiment.

PROCEDURE AND DESIGN

Participants completed 3 blocks of trials with each of the four techniques. Each block consisted of 20 selections, corresponding to 5 times 4 possible sizes of portion text to select, from the smallest (13 lines) to the largest one (508 lines). Participants were instructed to complete selections as quickly and accurately as possible.

The experiment used a 5×3×4 within-subjects design, with the following factors: TECHNIQUE (NSTV, NL, WK, GTK, and PE), BLOCK (1-3), and SIZE (S_{13}, S_{58}, S_{258}, and S_{508} lines of text to be selected). The order of TECHNIQUE was counterbalanced across participants. The experiment lasted approximately 40 minutes, with a total of 5×3×4×5 = 300 trials performed per participant.

PARTICIPANTS

Eighteen unpaid volunteers (mean age 29.8, SD 10.0, 14 males) participated in the experiment. Most were heavy computer users (more than six hours daily for all but one) and used a laptop as their primary computer (78%). Half of them used a mouse as their primary pointing device, while the others mainly used a touchpad.

7.4.2. RESULTS

The dependent variables were selection time, overshoot distance, and maximum pointer-edge distance. Error trials (3.0%) were filtered out for the analysis. The first repetition for each SIZE was also removed, as we observed that participants often failed to notice that the condition had changed. For all subsequent repeated-measures ANOVAs, we corrected the degrees of freedom using Greenhouse-Geisser estimates of sphericity when the assumption of sphericity was violated, and we used Bonferroni correction for pairwise comparisons.
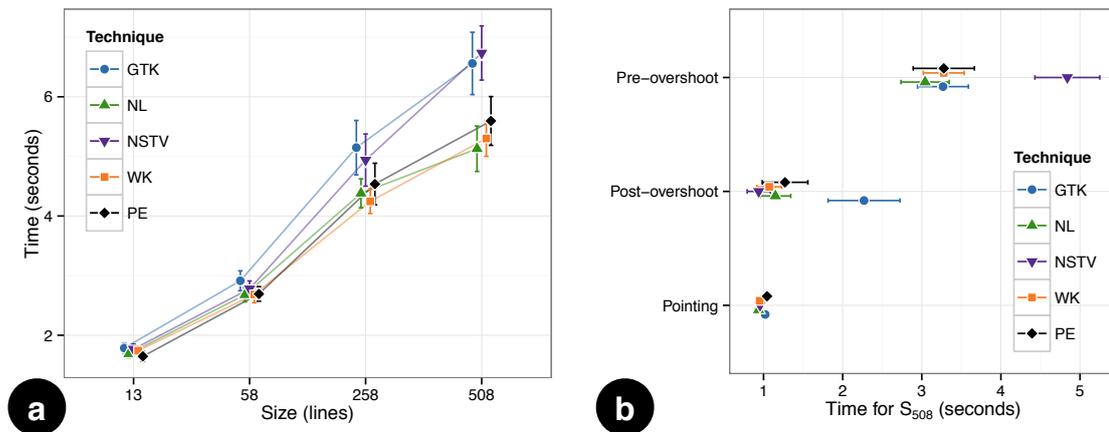
FIGURE 7.6: (a) Selection time by size and technique; (b) breakdown of selection time (for 508 lines) into pre-overshoot and post-overshoot scrolling times and pointing time, for each technique. Error bars show 95% confidence intervals.

SELECTION TIME

Selection time, the main dependent variable, is defined as the time taken to select the target, from the first time the pointer moved with the left mouse button pressed to the moment the left mouse button was released.

A repeated-measures ANOVA revealed a significant effect of BLOCK on selection time ($F_{2,34} = 29.9$, $p < .001$, $\eta_p^2 = .64$). Pairwise comparisons showed a significant decrease of selection time between the first block (4.2s) and the two remaining (block 2: 3.8s; block 3: 3.7s; $p < .001$), indicating a learning effect. Therefore, removed the first block from subsequent analyses.

We found a significant main effect of TECHNIQUE ($F_{4,68} = 6.9$, $p = .001$, $\eta_p^2 = .29$), of SIZE ($F_{1.1,19.4} = 109.7$, $p < .001$, $\eta_p^2 = .87$), and a significant TECHNIQUE × SIZE interaction ($F_{4.5,76.1} = 5.6$, $p < .001$, $\eta_p^2 = .25$) on selection time (Figure 7.6a).

Pairwise comparisons showed no significant difference in selection time between TECHNIQUE for S_{13}, S_{48}, and S_{258}. However, for S_{508}, selection time was significantly higher with NSTV (6.1s; $p < .001$), WK (5.3s; $p < .001$), and PE (5.5s; $p = .007$). It was also significantly higher with GTK (6.6s) than with NL ($p = .012$) and WK ($p = .046$).

When selecting short portions of text, all techniques have similar performance. This was expected only for NSTV and NL, whose transfer function was identical on the first third of the scrolling area. However, with long selections, WK, NL, and PE contrast with NSTV and GTK.

Despite implementing different mapping types, NSTV (rate control) and GTK (position control) exhibited a similar selection time. To understand why, we broke down the scrolling part of the selection time around the moment the user first overshot the target line, yielding *pre-overshoot* and *post-overshoot* scrolling times (Figure 7.6b). Selection time can then be expressed as:
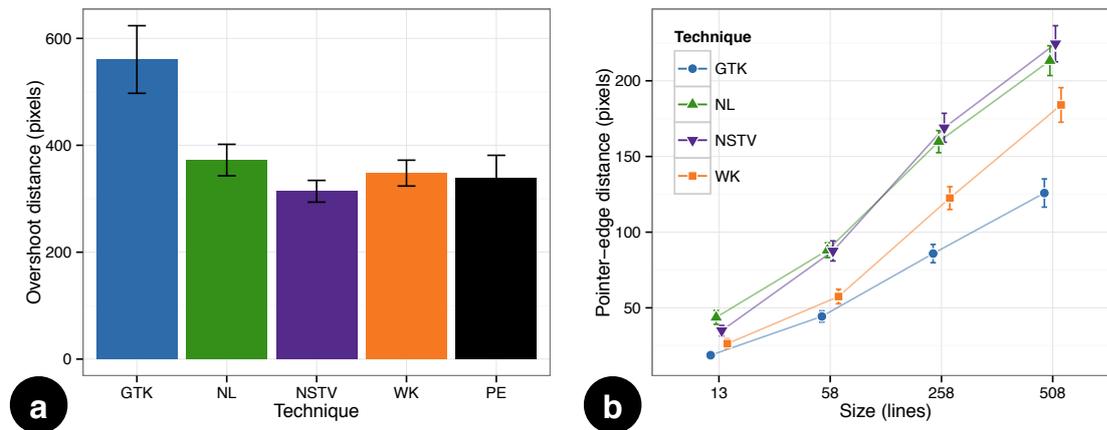
FIGURE 7.7: (a) Overshoot distance by technique; (b) Maximum pointer-edge distance by size and technique. Error bars show 95% confidence intervals.

$$T_{\text{selection}} = T_{\text{pre−overshoot}} + T_{\text{post−overshoot}} + T_{\text{pointing}}$$

As Figure 7.6b suggests, $T_{\text{pre−overshoot}}$ was longer with NSTV than with other techniques while $T_{\text{post−overshoot}}$ was similar to WK and NL. However, with GTK, the converse was true: $T_{\text{pre−overshoot}}$ was similar to WK, NL, and PE, and $T_{\text{post−overshoot}}$ was longer than with other techniques. The slow performance of NSTV in long selections is likely due to its limited maximum velocity (3000 px/s, compared to 6000 px/s with WK and 9000 px/s with NL, see Figure 7.5). However, the slow performance of GTK in long selections could be best explained by its tendency to induce overshooting.

OVERSHOOT DISTANCE

To better understand overshooting, we measured overshoot distance, the distance between the end of the target text and the farthest selection endpoint reached by the participant in a trial.

We found a significant main effect of SIZE ($F_{1.4,23.8} = 46.8$, $p < .001$, $\eta_p^2 = .73$), and more importantly of TECHNIQUE ($F_{1.7,29.3} = 5.9$, $p = .009$, $\eta_p^2 = .26$) on overshoot distance (Figure 7.7a). Pairwise comparisons showed that overshoot distance was significantly higher with GTK (560px) than with NSTV (313px; $p = .034$). It was also significantly higher with NL (372px) than with NSTV ($p = .020$).

This confirms that GTK is especially sensitive to overshooting, likely because pointer movements in any direction can issue scrolling movements: when the target line is brought into view, users move the pointer back to the viewport, and in the process generate more scrolling than necessary. In particular, with the longest selections, overshoot distance exceeded the height of the viewport with GTK, compelling participants to correct their movement by scrolling backwards before pointing to the target line.

MAXIMUM POINTER-EDGE DISTANCE

To analyze the use of the scrolling area, we measured the maximum pointer-edge distance reached by the participant in a trial for all techniques but PE, which blocks the pointer at the edge.

We found a significant main effect of TECHNIQUE ($F_{1.9,31.8} = 41.5$, $p < .001$, $\eta_p^2 = .71$), of SIZE ($F_{1.4,23.4} = 307.2$, $p < .001$, $\eta_p^2 = .95$), and a significant TECHNIQUE×SIZE interaction ($F_{3.4,56.9} = 11.4$, $p < .001$, $\eta_p^2 = .40$) on maximum pointer-edge distance (Figure 7.7b).

Surprisingly, with the rate-based techniques, participants did not maximize their use of the scrolling area. Pairwise comparisons showed maximum pointer-edge distance to be significantly lower with WK than with NL for S_{13} (26px vs. 44px; $p = .002$), with NSTV for S_{508} (184px vs. 225px; $p < .001$), with both techniques for S_{58} (58px vs. both 88px; $p < .001$), and with all other rate-based techniques for S_{258} (WK: 123px; NSTV: 169px, $p < .001$; NL: 160px, $p < .001$; GTK: 151px, $p = .008$). Participants seldom used the last third of the area, especially with WK, understandably because producing faster scrolling velocities requires less movement than other techniques when pointer-edge distance is under 200 pixels.

In addition, participants stayed closer to the viewport edge with GTK than with other techniques. For S_{13}, S_{258}, and S_{508}, maximum pointer-edge distance was significantly lower with GTK than with all other techniques ($p < .024$). Also, for S_{58}, it was significantly lower than with NSTV and NL (44px vs. both 88px; $p < .001$). Presumably, staying close to the viewport edge allowed participants to limit overshooting when going back inside the view, generally at the expense of scrolling speed. However, despite using only the first third of the scrolling area, some participants still managed to quickly advance scrolling position by oscillating the pointer horizontally.

7.4.3. DISCUSSION

The main finding of this experiment is that there are substantial performance differences between edge-scrolling implementations. We obtained similar results for the performance of NSTV and PE than in the previous experiment, but we also found no difference between the performance of PE and that of WK and NL, meaning that both rate control and position control transfer functions can be effective in long distances. However, unlike push-edge scrolling, overshooting is exacerbated when pointer movements in any direction, including towards the viewport, generate scrolling movements, as was the case with GTK.

The performance differences between NSTV and the other rate-based techniques also show that the slope of the transfer function has an effect on long distance performance. But it is only present up to a certain point: while NL is able to produce 50% higher scrolling velocities than WK, we found no significant difference in selection time between them.

Interestingly, the scrolling area was underused past 250 pixels, and the range of scrolling velocities that could be produced with the rate-based techniques was rarely used fully. However, further

experiments are necessary to determine whether higher scrolling velocities give benefits for even longer selections than the longest examined in our study.

In addition, notice that we did not find significant differences in overshoot distance between NSTV (NSTextView$_{select}$) and PE (push-edge scrolling) as we did in the previous experiment, which used shorter selection sizes. More work is needed to determine whether the overshoot advantage of PE disappears with longer selection sizes.

7.5. Discussion

In this section, we will first describe further criteria for the evaluation of edge-scrolling techniques based on the work on the previous chapter and the current one. We will then propose general guidelines for the use of edge-scrolling in applications, and discuss the limitations of our work as well as future interesting areas of edge-scrolling research.

7.5.1. CRITERIA FOR EVALUATION

Based on our results, we can formulate a number of desirable properties for edge-scrolling that can be achieved by manipulating the dimensions of the design space. They form criteria that can guide the evaluation or the comparison of several designs. However, all these criteria cannot be satisfied at once, and an edge-scrolling technique designer must carefully weight these tradeoffs to make an informed choice, as it is the situation at hand that determines which properties are most important. In the following, we will express these criteria, showing for each of them how they are affected by design dimensions and task-related factors, and describing their interrelationship.

EXPRESSIVENESS

In the case of edge-scrolling, *expressiveness* describes whether scrolling output reflects exactly and only the user's scrolling intentions, as expressed in pointing input. It is primarily determined by the amount of control given to the user in the scrolling area through the transfer function. But the expressiveness of a design also depends on whether expression of intentions is at all possible: in some configurations of the viewport, the user may not be able to do so.

CONTROL

The amount of control required for a given task and document may vary, but it is optimal when the output domain accommodates the user's scrolling intentions, and when enough of the output domain can be reached by the user. Control depends mainly on the *transfer function* and the *scrolling area* of a design.

Control can be limited when the output domain of the transfer function is artificially restricted to the point that it is impossible to produce the desired scrolling output. When the output domain is bounded, as with Word$_{select}$ or Photoshop$_{select}$, there is a risk that the task cannot be completed at the desired pace, or even that it cannot be completed at all. When the output domain is too sparse; that is, when the steps of the staircase function are too high (e.g. with

Excel$_{move}$, for which only three different values are available to the user), there is a risk that the task cannot be completed with the desired resolution.

Besides the output domain itself, control also depends on the scrolling area, which can change depending on the viewport geometry: it is possible that not enough values of the output domain can be produced given the current input domain. This happens when the window is maximized, which compels some users to resize the window before performing their task, or when a small edge-zone is used, for example during a *move* task. To work around the small input domain when the scrolling area lies on the display edge, the scrolling area could be virtually extended (past the display edge), despite the pointer being blocked at the edge. But this would require additional *visual guidance* to preserve the display invariant that pointer-edge distance is related to scrolling velocity. In the case of small edge-zone inside the display, an obvious solution to improve control is to increase the size of the scrolling area, but this could increase the risk of unintended activation. Short of enlarging the scrolling area, one can increase the range of possible output values, but this could limit the level of detail that can be attained in this range. This tension can be relieved by allowing subpixel interaction (Chapter 4), which increases the input domain (i.e., the number of possible pointer positions), while keeping the same scrolling area size. Also, note that pointer-blocking techniques such as push-edge and slide-edge do away with scrolling areas, and allow input domains limited only by the capabilities of the user.

AVAILABILITY

To guarantee that edge-scrolling is available in all situations without requiring the user to readjust the window, it is also necessary to pay attention to the *edge configurations* that impede interaction with the scrolling area. To make edge-scrolling available in the hidden edge configuration, the *scrolling area* can be changed so that the display edge is the relevant edge to consider for edge-scrolling: the scrolling area then lies within the display along its edges, as in NSTextView$_{select}$ and our push-edge and slide-edge techniques.

EFFECTIVENESS

While expressiveness determines whether an edge-scrolling technique allows the user to perform scrolling exactly as intended, *effectiveness* determines "how well" the user performs with the technique. Expressiveness does not automatically imply effectiveness, which can be characterized in a specific task by completion time or the amount of effort required, and by activation costs.

SCROLLING PERFORMANCE

The four tasks we identified (*select*, *edit*, *move*, and *point*) are composed of scrolling and pointing actions, are target-directed, and are arguably performed with approximate knowledge of target location. Such tasks might exhibit a speed-accuracy tradeoff similar to pointing tasks (Hinckley et al., 2002; Guiard et al., 2004). Task-related factors influencing performance may include target distance and size, but also viewport geometry (Zhao et al., 2014), although debated in the case of non-collocated input (Cockburn and Gutwin, 2009; Guiard et al., 2004). The effect of the

transfer function on scrolling performance has been established in the literature (e.g., Hinckley et al., 2002; Cockburn et al., 2012). However, in the case of edge-scrolling, other dimensions of the design space might impact performance as well: a small *scrolling area* with few opportunities for control is likely to have a negative effect on performance, *activation conditions* are especially important if target distance is short, and the feedback and anticipation aspects of *visual guidance* may help the user optimize her performance. Considering the effort required to perform the task, it is primarily related to the type of mapping used in the transfer function. Relative position control is prone to repeated clutching, unless oscillatory movements, non-linear control-display gain, or simulated inertia are used. On the other hand, rate control requires comparatively less movements.

EASE OF ACTIVATION

Another important aspect of performance not explicit in other scrolling techniques is ease of activation. The time and effort required to enter and exit edge-scroll mode may be particularly important depending on the amount of control provided by the technique, as edge-scrolling must be activated as many times as the user must change scrolling direction. This is typically the case in the corrective phase of scrolling. For example, the user may overshoot the target so much that she needs to edge-scroll again backwards to bring the target back into view, requiring another activation of edge-scrolling when entering the opposite scrolling area.

In edge-scrolling, activation involves two design dimensions from which activation costs originate: the *scrolling area* and the *activation conditions*. There are several possibilities for the scrolling area, and these can greatly influence the time required to acquire it. Reaching an edge-zone is a pointing task whose difficulty decreases with its size. Instead, if the scrolling area is display-bound or a pointer-blocking technique is used, then its acquisition is a much faster crossing task with an orthogonal movement variability constraint (Accot and Zhai, 2002). Besides, additional activation conditions can increase the cognitive and motor costs of entering edge-scroll mode, for example if holding another mouse button is required. In addition, with more activation conditions, edge-scrolling might not be perceived as a direct extension of pointing actions anymore.

EASE OF LEARNING

In all other criteria we assume that the user has already developed an appropriate mental model of the technique and is proficient in its use, but getting to this point requires learning, and the technique can be designed to accelerate the learning process.

The amount of practice required to reach expertise depends on whether the user is already familiar with the particular design of a technique. This is especially important with edge-scrolling, as it has been implemented in various software for at least two decades. Whenever a technique employs an unusual *transfer function* mapping or novel *activation conditions*, familiarity with the design cannot be assumed and the user has to discover the technique as well as its activation

and control mechanisms. In addition, the less consistent the designs encountered throughout the system, the more the user will deliberately limit her performance if she is not confident in anticipating the effects of each design. A design that closely resembles familiar interactions, like slide-edge scrolling with a touchpad, might help.

Because edge-scrolling is not a visible feature and may not be available in all contexts, it is important to help the user discover its presence. Moving the pointer to the viewport edge appears as a relatively natural action when the target lies beyond it, but the user might overlook the presence of edge-scrolling, if the *scrolling area* is too small or if its *activation conditions* require dwelling. As discussed in Section 6.5.2, appropriate *visual guidance* can increase awareness of the feature, and can also shorten learning time by providing better information on how it is activated and controlled. However, at some point, visual guidance can impose unwanted feedforward that can be disruptive when the user has no intention to edge-scroll and wants to manipulate other interface functions.

LOW RISK OF UNINTENDED ACTIVATION

In some situations, the user can perform ambiguous actions where it is unclear whether she wants to control edge-scrolling or perform another interface function, often because both functions use the same display area for different and conflicting purposes.

It is possible that edge-scrolling obstructs other interface functions because the scrolling area overlaps elements that the user may want to use. For example, when dragging an object to a point near the inside edge of the viewport, the user risks entering an edge-zone, thereby unintentionally activating edge-scroll mode. Another ambiguous situation occurs when crossing the edge of a viewport while performing an inter-viewport interaction, such as dragging a file. It is sometimes unclear whether the user wants to edge-scroll the current viewport, or whether she wants to continue the dragging action in another viewport. As a result, the viewport might start scrolling regardless of the user's real intentions.

Unintended activation was absent from survey respondents' accounts of their edge-scrolling experiences. Indeed, most existing designs avoid unintended activation by using a small edge-zone *scrolling area* interior to the viewport, so that what is outside it can be used for the dragging operation. However, as we have seen, such a small area impair expressiveness. As an alternative, some designers used an additional *activation condition* to disambiguate intentions (Berry et al., 1991; Li and Shrader, 1998), but these designs might be more difficult to discover and learn unless *visual guidance* is used for awareness.

7.5.2. DESIGN RECOMMENDATIONS

The knowledge gained throughout the studies in the previous chapter and the present one increase the generative, descriptive, and evaluative power (Beaudouin-Lafon, 2004) of the design space of edge-scrolling techniques. Figure 7.8 shows a revised version of this design space that
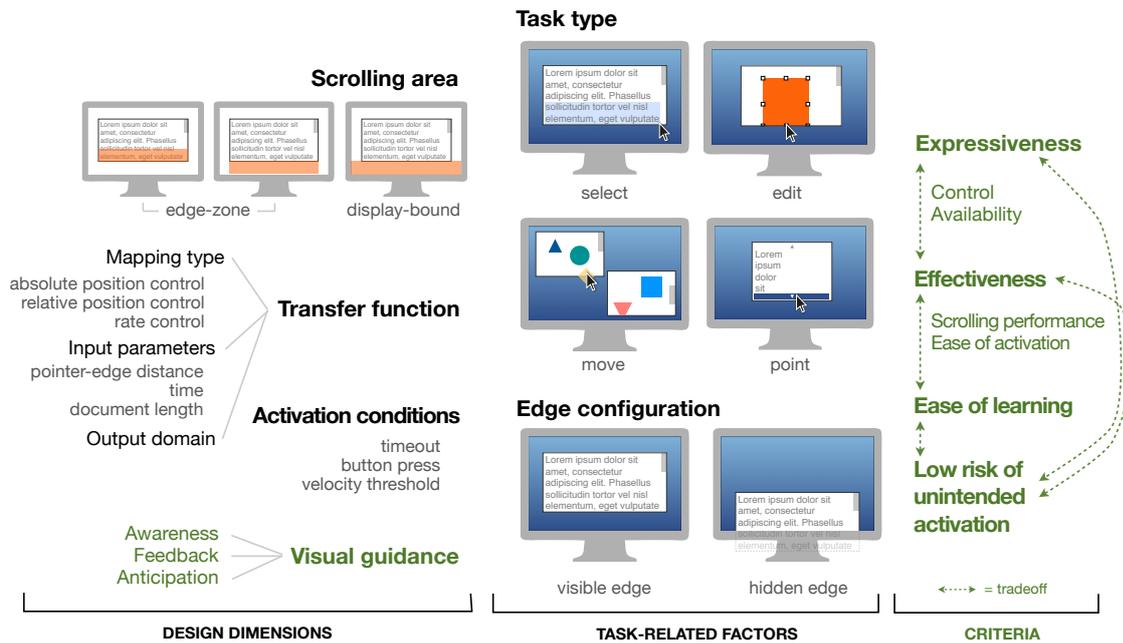
FIGURE 7.8: The revised design space of edge-scrolling, completing the one presented in Figure 6.2 using insights from our empirical results: an additional design dimension (*visual guidance*) and criteria for evaluation (in green).

integrates the new dimension (*visual guidance*) defined in Section 6.5.2 from our thematic analysis of survey respondents, as well as the criteria for evaluation discussed previously. In addition, a number of recommendations can be made on control, performance, and consistency.

Analysis of survey comments (Section 6.4) showed that all dimensions of edge-scrolling were involved in user's perception of control.

R1 A small *scrolling area* can negatively impact control, both because it does not offer a large range of controllable values, and because pointing at them is difficult when their boundaries are not clearly *visually* delineated.

Note that, however, participants in the experiments seldom moved further than 250 pixels (70 mm).

R2 Ensure that the *transfer function* permits an adequate range of output values to be attained in all possible configurations of the scrolling area.

Highly constrained transfer functions that allow only a small range of output values to be produced impede perceived user control.

R3 *Activation conditions* based on a delay can be detrimental to the sense of control when there is no *visual guidance* to help anticipate activation.

R4 Always provide *visual feedback* on the user's location in the document during edge-scrolling to mitigate some of the spatial awareness issues reported by survey participants.

> This could happen when the scrollbar is not visible, for example because it is placed behind another window or lies outside the display. In the experiments, both the current scrolling position and the target text were clearly displayed in scrollbars.

Insufficient control may distract users from their task and increase the risks of overshooting and disorientation, leading ultimately to suboptimal performance. Performance is also influenced by the transfer function.

R5 Linear rate control *transfer functions* with slopes less than 10 (NSTextView$_{\text{select}}$) may not yield optimal performance for long distances.

> The results of the second experiment suggest that users are not likely to move the pointer far enough away from the viewport edge to attain suitably high scrolling velocities. A linear rate-based transfer function with a slope constant of 20 (Safari$_{\text{select}}$; WK in the experiment) apparently provides a good compromise between accessibility of high scrolling velocities and degree of control.

R6 If deploying a position control *transfer function*, pointer movements towards the viewport should not produce scrolling displacements. Push-edge and slide-edge techniques do not suffer from this liability.

> As seen in the second experiment, this behavior is likely to cause overshooting, decreasing performance and requiring more user attention. Movements towards the viewport in the diagonal direction cause the same problem, but lateral pointer movements have been used by our participants with GTK to momentarily accelerate scrolling.

Examination of existing implementations showed marked inconsistencies that were sometimes problematic, as commented by survey respondents (Section 6.4).

R7 Unfamiliar or inconsistent *activation conditions*, *scrolling area*, or *transfer function* are noticed by users and can induce general confusion or distrust of edge-scrolling.

> Consider using *visual guidance* to help users understand and control idiosyncratic behaviors.

R8 Rate control mechanisms are far more common than position control, so consider the consistency impact before deploying a position control technique.

> Slide-edge scrolling is a novel technique, but note that it behaves similarly to two-finger touchpad scrolling in modern operating systems.

7.5.3. GENERAL LIMITATIONS AND FUTURE WORK

The work presented in the previous chapter and in the current one combined several methods to provide a detailed description of the design space of edge-scrolling, but it only explored part of it, and our results have left aside some of the complexity of real-world edge-scrolling. Although the revised design space can help generating, comparing, and evaluating designs, accurate description is out of its scope due to the intricacies of existing designs (e.g., the pointer-blocking aspect of push-edge and slide-edge scrolling, or the alternative definition of pointer-edge distance of $QTextEdit_{select}$). Reverse-engineering existing implementations is useful to extract and analyze high-level details, but the current methodology does not cover the design space thoroughly and forbids exact replication. Our survey of user experience with edge-scrolling pinpointed several problematic aspects of existing designs, but much could be gained from the analysis of these aspects with more focused methods, such as interviews. Finally, our controlled experiments only dealt with the unambiguous intra-viewport *select* task and display-bound scrolling areas, and the range of studied transfer functions was limited. But despite these limitations, our results have raised important questions and laid out the groundwork for future edge-scrolling research.

Almost none of the existing designs used visual guidance, and while we have suggested how it could improve edge-scrolling interaction in several ways, empirical assessment of the effects of visual guidance remains to be done.

Another unsolved challenge is the design of edge-scrolling behaviors for inter-viewport tasks. The additional complexity stems from ambiguity about the target viewport (*"is the user dragging the object to a new location within the current viewport, or does the target reside in another viewport?"*). Pointer-blocking techniques like push-edge and slide-edge and behaviors using display-bound areas will likely have to resort to explicit or sophisticated deactivation mechanisms. With other behaviors, this ambiguity constrains the extent of the scrolling area, and increases the potential of accidental activation of edge-scrolling. However, our results suggest that the scrolling area might be underused past 250 pixels, and that the slope of linear rate control transfer functions might be superior to 20 px/s without harming control. Determining a minimum pointer-edge distance above which acceptable performance and control can be preserved would help designing better edge-zone techniques.

The empirical studies in this work focused on desktop applications, where edge-scrolling was primarily performed while dragging using an indirect pointing device. However, edge-scrolling with collocated input (e.g., mobile devices) remains an uncharted area with regard to reverse-engineering methodologies and performance evaluation. With touchscreen interfaces, it is also common to edge-scroll when moving items in a list, or application icons between the "pages" of a home screen. But the often single-windowed environment poses hard constraints (intra-viewport interaction only, small scrolling area) that surely impact usability and make the design of efficient edge-scrolling techniques even more challenging.

7.6. Conclusion

In this chapter, we presented two novel edge-scrolling techniques, push-edge and slide-edge scrolling, that allow scrolling by "pushing against the viewport edge". The results of two controlled experiments on a text selection task suggest that they provide comparable performance to the best rate control techniques tested, but unlike them, they provide maximal control of scrolling movement with all possible viewport geometries. Moreover, the results showed that there are marked differences in user performance between current solutions, highlighting possible effects of the pointer movement directions that generate scrolling displacements for position control techniques, and of transfer function slope for rate control ones. Finally, from the combined results of the previous chapter and this one, we outlined other aspects of edge-scrolling evaluation, proposed design guidelines, and discussed possible areas for further edge-scrolling research.

This concludes the third part of this dissertation. We have established a base for the design of edge-scrolling techniques (Chapter 6) and empirically assessed the performance of several designs (this chapter). The two methods proposed, push-edge and slide-edge scrolling, contribute to increasing the expressive power and directness of extended dragging by avoiding two limitations identified in existing techniques. First, by blocking the pointer at the viewport edge (or the display edge if closer), they provide an area limited by the operating range of the device only, thus expressive power is independent from current viewport geometry or ambiguity considerations. Second, by allowing position control, they provide a more *referentially direct* control of scrolling movement. In addition, on touchpads, slide-edge scrolling promotes skill reuse by its resemblance to existing two-finger scrolling gestures. Finally, the limitations and future work discussed in Section 7.5.3 suggest that inter-viewport situations remain an important unsolved challenge concerning the directness and expressive power of edge-scrolling.

Increasing dimensionality in musical expressions

Office music: musical performance with a standard laptop

We now enter the final part of the dissertation, which is concerned with the dimensionality of user actions, and focuses on a specific application domain, the real-time control of musical processes.

With a growing portion of music equipment being available in software form, desktop and laptop computers never had a more important role in digital music production activities. According to the 2014 annual report of the National Association of Music Merchants (NAMM, 2014), the market for computer music products has grown by over 13% in the last 10 years, with increasing use and interest in commercial digital audio workstations (DAWs). Their graphical interfaces allows to easily edit recorded performances and navigate through soundtrack, audio effect, and virtual instrument parameters. Used with a laptop, they provide an opportunity for amateur and professional musicians to create digital music "in the box" (Adams et al., 2014) using an affordable and transportable package that fits in a backpack.

Recording and live musical performance, on the other hand, often require to simultaneously control dozens of sound parameters in real-time, a task to which graphical—and mostly time-multiplexed—interaction is not well adapted. DAWs usually allow musical note entry using the computer keyboard (Figure 8.1), thus the musician can quickly preview the effect of their modifications to sound synthesis tracks. But control is inevitably limited: the range of playable notes is generally around an octave and a half, and the musician cannot nuance intensity or timbre when playing.

Due to this, the laptop is usually accompanied with acoustic and electric instruments, audio input devices, and various physical controllers that provide more dimensionality in musical control but must be transported, configured before use, and maintained (Fiebrink et al., 2007). Throughout a session, musicians repeatedly reach for them and go back in front of the laptop for rapid experimentation. The result is twofold. First, the "laptop in a backpack" studio loses its appeal as an affordable and mobile enabler of musical expression. And second, switching between the laptop and physical instruments incurs a large amount of *turn-taking* that is arguably detrimental for the musician. This point is even used as a commercial argument for modern all-in-one hardware controllers[1].

---

[1] For example, the press release for Ableton Push, a dedicated controller and instrument for Ableton's DAW, Live, presents it as a way to "play and compose musical ideas without the need to look at or touch your computer": `https://www.ableton.com/en/press/press-archive/press-archive-release-9-push`. Last accessed 06/26/2015.

Figure 8.1: Musical note entry "cheat sheet" interfaces for the computer keyboard in digital audio workstation software: Apple Garageband (left), Steinberg Cubase (middle), and Propellerhead Reason (right).

In this chapter[2], we argue that a unequipped laptop can, in fact, allow high-dimensional musical expression "in the box" in a way that closely integrates with other musical activities, and we propose three "office music" prototypes where the laptop can be played like a traditional instrument. We will first review the research related to musical performance, expression, and controller design, and describe previous attempts to enable musical performance with standard computer input. We will then consider several design aspects related to input devices, musical gestures, and physical support, and compare several alternatives of computer keyboard layouts for note entry. Resulting from these considerations, we will detail the design and implementation of our three laptop musical instruments—the Pianotant, the Soufflant, and the Tapant—and the facilities they offer to switch between the "performance mode" they establish and normal graphical interaction. Finally, we will discuss our successful use of these instruments in various projects, and present examples of physical uses of the laptop in other contexts than musical interaction.

8.1. Related work

Musical performance encompasses various types of musical contexts and behaviors. Malloch et al. (2006) proposed a classification of digital musical instruments inspired from the SRK taxonomy (Rasmussen, 1983, detailed in Section 2.2.2), and arranged them along a spectrum, based on the amount and rate of control given by the instrument to performer. At one extreme, live coding (Collins et al., 2003) and algorithmic music are examples of practices exhibiting knowledge-based behavior: the performer interacts with symbolic structures and relies on problem solving and planning skills. Further along the spectrum, rule-based behavior is typical of sequencing and live looping (e.g., Berthaut et al., 2010): the performer selects and triggers high-level processes. At the other extreme, performance with traditional instruments is an example of skill-based behavior: the performer is in charge of the performance at the lowest level and relies heavily on perception, motor control, and coordination skills. In Section 3.3.5, we briefly described the many contexts in which the standard laptop has been used due to its versatility, and contended that it can span the entire spectrum of musical performance. The perspective of bringing several types of performance behavior together is appealing, because it would allow a musician to switch between various levels of control with great ease, while keeping the convenience and transportability of the standard unmodified laptop.

---

[2]The work presented in this chapter has been carried out in collaboration with Ludovic Potier.

The opportunity of integrating these behaviors is also promising because it would facilitate circulation between different types of creative processes. Based on existing models from cognitive psychology (Kahneman, 2011) and computational creativity (Wiggins, 2006), Tubb and Dixon (2014) proposed a four-strategy model of creativity with digital musical instruments, framing creative behavior as exploration in a conceptual parameter space (e.g., that of the controllable musical parameters) in the search for a suitable solution. Convergent strategies explore local solutions and are guided by an evaluation function that encompasses the creative expectations of the performer: the *analytic* strategy consists in systematically testing nearby solutions one parameter at a time, and the *tacit* strategy leverages well-learned, automatic behavior to quickly arrive at a good enough solution. Divergent strategies instead look for original solutions outside the known territory: the *exploratory* strategy consists in generating a large number of solutions by combination or random processes, and the *reflective* strategy transforms the conceptual space or the performer's expectations, for example by metaphor (Morse and Reynolds, 1993) or by metonymy (Bertelsen et al., 2007). The large differences between the "analytic mode" promoted by traditional graphical user interfaces and the "holistic mode" usually attributed to musical instruments (Hunt and Kirk, 2000) suggest that the former are well equipped for analytic and reflective strategies, but the latter rather enable tacit and exploratory ones. Therefore, the promise of the laptop to integrate various types of performance behavior might be important for supporting the four types of creative strategies, as well as the transitions between them.

The focus of this chapter is mainly on enabling advanced forms of skill-based performance behavior, but we will try to keep this integrated perspective in mind. In the rest of this section, we will first discuss the important notion of expressiveness in the context of musical performance, as well as the various associated considerations. We will then detail how graphical interaction, and more generally laptops, have been used in the design of digital musical instruments.

8.1.1. EXPRESSIVENESS AND MUSICAL PERFORMANCE

Musical expression is a complex phenomenon involving many facets of human behavior, such as emotion, movement, and communication. The GERMS model (Juslin, 2003) separates five main functions of expression in musical performance:

Generative rules Through changes in acoustic features, which may be codified by rules inherent in some musical styles, the performer conveys a musical structure. For example, deceleration (*rubato*) is frequently used in classical music to signal the ending of a musical phrase.

Emotion In addition to structure, the performer may communicate emotion through her performance. For instance, reliance on major or minor modes can be used to convey positive or negative emotions.

Randomness Slight random variations in rhythm, articulation (continuity in the transition between a sequence of notes), or intonation (accuracy of pitch) are impossible to avoid in human performance (Lehmann et al., 2007) but contribute to the feeling of liveness.

Motion Musical expression also conveys a sense of motion that can originate from the performer's aesthetic intentions, and whose dynamics are closely related to the bodily movements needed to produce the music.

Stylistic unexpectedness Musical styles establish a set of expectations for the listener that the performer can play with—for example, postponing the resolution of a harmonic progression by introducing an unexpected chord or rhythmic pattern.

Thus, expressiveness with a musical instrument cannot be condensed to the number of different parameters that can be manipulated in real time and the extent and granularity of parameter control afforded by the instrument (Dobrian and Koppelman, 2006; Malloch et al., 2006). But such variables, among others, make up expressive power (Section 3.2), which greatly influences how a performer chooses to fulfill the above functions.

MUSIC, GESTURE, AND INPUT CONSIDERATIONS

What does this entail in the context of musical performance? Among the determining properties of an instrument, Settel and Lippe (2003) consider *instrumental expressive range*; that is, the range of musical styles the instrument can be used for. Another important consideration is the *musical range* (Blaine and Fels, 2003) an instrument offers; that is, how limited is the set of possible notes or sound qualities that can be created by the performer.

Gesture acquisition is also important, and many properties of gestures and of the input devices that sense them can be considered. The three functions of instrumental gestures (Cadoz, 1988; Cadoz and Wanderley, 2000), which we have already discussed in Section 3.1.1, can be further detailed as follows. Excitation gestures can be *instantaneous* or *continuous* depending on whether energy is provided in a single event or throughout sound production. Instantaneous excitation can be further broken down into *percussion* and *picking*—when sound production only begins when physical contact ends. Modulation gestures are *parametric* if the discrete or continuous parameter that is controlled varies continuously (e.g., vibrato), and they are *structural* if they concern a qualitative change (e.g., the sustain pedal of a piano). Selection gestures can be either *sequential* or *parallel*. In addition, musicians perform *ancillary gestures* (Wanderley, 1999) that constitute part of their expression: for example, wind instrumentalists move their instrument in a way that accompanies the sound being produced. The visual information ancillary gestures convey fulfills most of the above functions of expression (Wanderley et al., 2005).

Considering input devices, much of the general HCI knowledge (e.g., Buxton, 1983; Card et al., 1991; Jacob et al., 1994) applies, but in some cases the specificity of the musical context warrants more detailed models. For example, Vertegaal et al. (1996) propose a classification of input devices according to the physical property sensed and sensitivity of the device, the type of feedback (tactile, kinesthetic, or visual) provided, and the suitability of the device to fulfill musical functions. They consider three categories of musical functions: *absolute dynamical* (e.g., selecting a pitch), *relative dynamical* (e.g., varying amplitude to perform a tremolo), and *static*

(e.g., shifting between different tunings). The above considerations on gestural functions and input devices have been applied by Wanderley et al. (2000) in the selection of mappings for the control of sung voice synthesis using a Wacom tablet equipped with a stylus.

MAPPING

In addition, Rovan et al. (1997) insist on the importance of the mapping between gesture and sound and discriminate three mapping strategies that present different implications for expressiveness. Mappings where each gestural dimension is linked to a separate musical parameter (*one-to-one mapping*) are commonly found unsatisfactory and are not deemed sufficient to allow the development of virtuosic[3] performance (Wessel and Wright, 2002). Successful instruments often present more complex mapping strategies (Hunt and Kirk, 2000) and may combine *divergent mapping* (or one-to-many), where one gestural dimension to several musical parameters, and *convergent mapping* (many-to-one), where one musical parameter is changed through the coupling of several gestural dimensions.

In order to provide more generality in mapping strategies, multi-layer models have been proposed (Wanderley and Depalle, 2004). For example, Arfib et al. (2002) consider three layers of mapping: first between gestural input and a perceptual space related to gesture, then between the gesture-related perceptual space and a perceptual space related to sound, and finally between the sound-related perceptual space and the parameters of the model used for synthesis. This approach has two benefits: on the one hand, perceptual spaces permit the explicit description of high-level considerations that may be more semantically direct (e.g., considering the amount of vibrato instead of fundamental frequency), and on the other hand, the modularity of the three layers allows input devices and synthesis models to be easily replaced.

EFFICIENCY AND CONTROL DIVERSITY

Jordà (2005) gives a higher-level view of these considerations with the notion of *efficiency*, which represents the relationship between the complexity of controllable musical output and the amount of effort needed to produce it. Schematically, he conceives efficiency as the ratio of two properties, *musical output complexity* (MOC) and *control input complexity* (CIC), multiplied by another property he calls *diversity control*. MOC depends on the richness and variability of aspects of the sound structures, at any level of detail, that an instrument can produce. This encompasses high-level aspects, like the range of pitches that can be produced, as well as lower-level ones, like the possibilities for nuancing timbre. If MOC relates to the "contents", CIC would be its "expression" counterpart: it depends on aspects of the control structure such as its integrality and the number of DOFs, but it also concerns aspects related to mapping. While the definitions of MOC and CIC are not clear-cut, they encapsulate the two separate spaces (music and gesture) brought together by the musical instrument.

---

[3]According to Aho (2013), a gesture is virtuosic if it is executed relatively effortlessly but defies the audience's tacit experience of how the musical instrument can be played.

But the notion of efficiency is not complete without asking what aspects of the musical output can be accounted for by the performer: this is the question asked by the diversity control property, which Jordà (2005) describes at three levels. *Micro-diversity* relates to the fine control of parameters, such as nuances in timing, loudness, pitch, and timbral properties. When an existing piece is played, micro-diversity allows subtle variations in the performer's expression that constitute her interpretation, but may not change the identity of the piece. *Mid-diversity* relates to the variety of performances an instrument allows to produce in a given context: a guitar or a piano can be used to produce a very wide range of different pitches, hence they have more mid-diversity than a drum kit or an MP3 player. Finally, *macro-diversity* concerns the extent of different contexts the instrument can fit in: for example, a double-bass can fit fewer musical roles and styles than a piano, thus it has a lower macro-diversity. To Jordà (2005), all three levels of diversity contribute to the performer's expressiveness, but mid-diversity and micro-diversity are especially important, for the first one allows "music gamers" to become "music performers" while the second one allows performers to become "virtuosi".

8.1.2. PERFORMING WITH STANDARD DESKTOP AND LAPTOP INPUT

Computer music appeared almost as early as interactive computers (Roads and Mathews, 1980), and now the use of laptops on stage is commonplace. Laptop ensembles have been thriving for the last 30 years with various forms of social and technical organization (Knotts and Collins, 2014). A typical example of laptop ensemble is the Princeton Laptop Orchestra (PLOrk), an ensemble of 15 performers, each equipped with a laptop, an independent sound system, and various external controllers. Smallwood et al. (2008) describe how, through several compositions, they explored the sound spatialization and networking possibilities offered by such a setup. Their focus was more on the orchestra as a *macro-instrument* rather than on individual performance. While at first they relied mostly on native laptop input for instrumental control (detailed later), they were dissatisfied with the limitations and progressively introduced external devices. And indeed, in computer-equipped performances, the laptop often supplements or is supplemented with other instruments and devices. With the exception of live coding and other knowledge-based and rule-based uses, the unmodified laptop is rarely "played" as a musical instrument in itself.

GRAPHICAL INTERACTION AND INSTRUMENTAL PERFORMANCE

There is a plethora of graphical interactive musical software in the form of sequencers, samplers, synthesis engines, and more general programming environments such as Max or Pure Data. Most were designed to support knowledge-based and rule-based behavior in off-line activities like editing compositions, consigning skill-based instrumental performance to external devices, such as MIDI keyboards or wind controllers, that are arguably better suited for this task. However, several designs have shown that a computer can be sometimes seen as the instrument itself, and that graphical interfaces can support real-time performance despite the input limitations. As Couturier (2006) showed, the instrumental interaction model (described in Section 2.3.5) can be adapted to the description and evaluation of such designs.

An early example of graphical interface for real-time performance is Music Mouse[4], a program for the original Macintosh where notes and chords are played by moving a "polyphonic cursor" around a grid representing pitches, while specific keyboard keys are assigned to changes in parameters such as tempo, transposition, selection of scale or chord voicing. A more recent example of mouse-controlled instrument is FMOL (Jordà, 2002), whose interface displays six vertical lines that can be plucked and fretted like guitar strings—by dragging them—and that vibrate to provide visual feedback on sound synthesis. Horizontal lines cross the strings and can be dragged to modify various parameters of the synthesis modules associated to each string. FMOL has been used in public live performances and its use has been taught in several workshops. Couturier (2006) has also used the string metaphor to drive a scanned synthesis engine or a bank of filters applied to a noise generator.

Other projects use drawing as the main metaphor for interaction, treating the display as a graphical score, thus often departing from real-time performance. This approach finds its roots in the UPIC system designed in 1977 by Iannis Xenakis (Marino et al., 1993). In its simplest mode, UPIC plays what the user draws on the screen by assigning the horizontal axis to time and the vertical axis to frequency. Modern descendants of UPIC like HighC[5] and IanniX (Jacquemin et al., 2012) leverage various interaction instruments usually found in vector drawing programs, such as line, Bézier curve, or transformation tools. Related designs regard the display canvas as an "inverse spectrogram" from which sound is resynthesized (e.g., MetaSynth[6]). However, they do not generally support real-time performance, unlike Yellowtail (Levin, 2000), where the user draw marks that animate and move according to the shape and speed of the original drawing. In Yellowtail, marks inside a predefined square area, which can be dragged around, are converted into sound when crossed by a horizontal line that regularly sweeps the area.

Another idea exploited in another prototype of Levin (2000), Loom, is that a marking gesture can be played back and that its dynamics can drive synthesis parameters. In Loom, the instantaneous velocity at a point of a mark is mapped to the amplitude of a frequency modulation (FM) synthesizer (the quicker one draws, the louder it sounds), and local curvature controls the brightness of the sound (angles in a drawing result in a momentarily harsh and percussive sound). The idea is further explored in Different Strokes (Zadel and Scavone, 2006), a live mouse-controlled performance system where small white particles travel along a mark that has just been drawn, with the same dynamics as the original gesture. The position and speed of a particle drives the playback of a prerecorded sound associated with the mark, and before drawing a mark, various sounds can be selected by using keyboard keys. A particle replicates once it crosses an intersection of two or more marks, and each newly created particle continues its own path. This simple rule can give rise to complex behavior: for example, looping can be achieved by drawing a loop, and chaining different sounds is done by making the end of one mark intersect near the other's starting point.

---

[4]http://www.musicmouse.com. Last accessed 06/07/2015.

[5]http://highc.org. Last accessed 06/08/2015.

[6]http://www.uisoftware.com/MetaSynth. Last accessed 06/08/2015.

Graphical interfaces also offer the opportunity to interact with dynamic graphical representations that exhibit autonomous behavior. Magnusson (2005) describes the investigations of the ixi software group on alternatives to standard GUI widgets for the design of "abstract graphical interfaces" to sound engines and programming environments like Pure Data and SuperCollider. Starting in 2000, the ixi group developed over 17 different interfaces for real-time musical performance, and more recently provided a library for SuperCollider named ixiQuarks (Magnusson, 2007) containing 15 other "instruments". In most of these designs, the user interacts with graphical objects with independent but controllable behavior. For example, in the SpinDrum (Magnusson, 2005) interface, the user interacts with a set of "wheels" dropped on a canvas, each one composed of small squares associated to a sound sample and arranged in a circle. Wheels rotate at different speeds, and once a square reaches the top, the associated sample is triggered. The user can change the position of the wheel in the canvas (mapped to pitch and panning), as well as its size (mapped to volume). In the Predators (Magnusson, 2007) interface, the user controls the parameters of a simulation of autonomous agents that can be either prey or predator; when a predator eats a prey, a sample or a tone is triggered with sound parameters related to the state of the agents involved.

Games have also been adapted to allow real-time collaborative musical performance. In first-person shooter games, the player navigates a 3D world using the keyboard, collects items such as weapons or health items, and aims and shoots at other players using the mouse. Musical modifications of Quake III Arena (Hamilton, 2008) and of the engine behind Unreal Tournament (Hamilton, 2011) have been developed that augment the shooter metaphor with musical expression capabilities. We will detail another example, *Couacs* (Berthaut et al., 2011), which was briefly mentioned in Section 3.3.2. Couacs provides each player with her own sound process, which is excited by shooting (clicking the mouse) and modulated by player movements, each weapon determining a different mapping. Couacs introduced imaginative workarounds to the expressiveness limitations of desktop input devices. For example, mouse buttons usually provide only binary input, but the energy of instantaneous excitation gestures can be specified by successively clicking a varying number of adjacent buttons at different speeds in different directions. However, allowing these gestures might hinder the execution of more advanced percussive techniques, such as drum rolls. Other types of games might prove interesting when applied to musical performance; for example, a "sonification" of the real-time strategy game StarCraft 2 has been recently developed by Cerqueira et al. (2013) and its use has been explored in a musical composition.

PUSHING THE LIMITS OF STANDARD INPUT

The designs described above leverage the possibilities of graphical interaction and the adequacy of standard input devices for it. But the lack of parallel pointing input imposes sequentiality, which renders a performance style similar to traditional musical instruments difficult to achieve. A promising solution would be to establish a "performance mode" where input is decoupled from its standard functions (e.g., the pointing device from the pointer, and the alphanumeric

keys to the production of letters or the activation of commands) and directly serves musical purposes.

While they usually control a single pointer, some laptop touchpads can sense multiple fingers in order to allow gestures such as two-finger scrolling. The opportunity to access individual finger information has enabled bimanual designs for tasks like color space exploration (Gonzalez and Latulipe, 2011). Adams et al. (2014) describe the design of SonicExplorer, an application allowing the bimanual and integral exploration of four audio parameters. SonicExplorer was designed to facilitate the exploration of a parameter space and lead to better creative choices compared to usual analytic interfaces that provide one widget per parameter that must tediously adjusted sequentially.

Other examples of the direct use of laptop input in musical performance come from the work of Fiebrink et al. (2007) on several musical compositions for PLOrk. In some pieces, alphanumeric keys of the keyboard were used to trigger sounds with different pitches. In another one, relative motion of a finger "bowing" on the touchpad controlled the loudness of a synthesized sound while the absolute position of the finger allowed sound to flow from one speaker to another. Another piece also used the integrated microphone by switching between two filter frequencies depending on whether the user was blowing into the microphone above a certain level. However, these designs are ad-hoc, they do not consider the interplay of the different input devices, and there is little guidance on the design of laptop-based musical instruments with standard laptop input devices.

The work presented in this chapter pushes further away the limits of standard input and will demonstrate that (1) real-time instrumental performance with high levels of micro-diversity control is possible with a standard laptop if its physical and input constraints and opportunities are taken into account in design, and that (2) with a laptop alone one can easily switch between skill-based, rule-based, and knowledge-based musical behavior during a performance.

8.2. Enabling musical expression using standard laptop input

The first step of our approach to designing laptop-based musical instruments consists in devising appropriate guidelines given the possibilities and limitations of standard input devices, the possible gestural functions they can fulfill, the physical constraints of the laptop, and integration with other uses of the laptop during a performance, such as live coding or graphical interaction with sequencer software. At the end of this section, we will also tackle the problem of pitch selection using the keyboard and describe the tradeoffs involved in the design of a musical keyboard layout. These considerations are independent from musical output and will drive the design of the instruments described on the next section.

8.2.1. INPUT DEVICE PROPERTIES

To determine the possible uses of a laptop's input devices in skill-based musical expression, we devised a set of properties based on the discussion of Hunt and Kirk (2000) of the attributes of instrumental real-time control systems that allow musical performance:

Number of parallel inputs Several body limbs are engaged simultaneously in control through instrumental gestures. But while control can be distributed over distinct devices, some actually allow more than one parallel input of the same kind.

Degrees of freedom Performance with traditional instruments often entails integral control of multiple parameters using the same device.

Control granularity Depending on the musical parameter to control and the degree of micro-diversity required, a threshold level of granularity might be expected for a particular degree of freedom. Kvifte and Jensenius (2006) point out that most general musical parameters (pitch, loudness, timbre, duration) can be both conceived as continuous or discrete, but that a discrete input rarely controls a continuous parameter in acoustic instruments. They also consider important that the level of measurement[7] of an input at least matches that of the musical parameter. However, in control granularity we are only concerned with the level of detail at which a physical property is sensed. Control granularity is constrained by the sensitivity of the input device, and is closely linked to the following property.

Reliability While a certain degree of randomness can be interesting in performance, similar gestures are expected to produce similar results to provide the performer with a sense of control. The reliability with which a physical property is sensed or an input is inferred can be important in this case. This includes the device's static stability (Bérard et al., 2011)—the standard deviation of the measure when there is no ongoing physical action—as well as the amount of variation of latency (Wessel and Wright, 2002).

Responsiveness Performance with traditional musical instruments relies on a tight coupling between perception and action, thus the latency between gesture and audio output should stay as low as possible.

Along this section we will illustrate the proposed method using the input devices (Figure 8.2) provided by the "unibody" Macbook Pro (further referred to as MacBook), a line of Apple laptops first released in October 2008[8] and still commercialized to this date. Figure 8.3 summarizes their properties, which are discussed in the following.

The laptop keyboard is more compact than standalone 104-key models used for desktop computers, and often lacks a numerical keypad. The MacBook keyboard has 79 keys in its ISO version. Each key can be either pressed or released, and few non-modifier keys can be simultaneously

---

[7]Stevens (1946) defines four levels of measurement with a gradual amount of mathematical properties: nominal (identity), ordinal (order), interval (distance between values), and ratio (presence of a zero value, multiplication and ratios of values).

[8]http://en.wikipedia.org/wiki/MacBook_Pro. Last accessed 06/09/2015.

	LINEAR			ROTATIONAL			
	X	Y	Z	rX	rY	rZ	
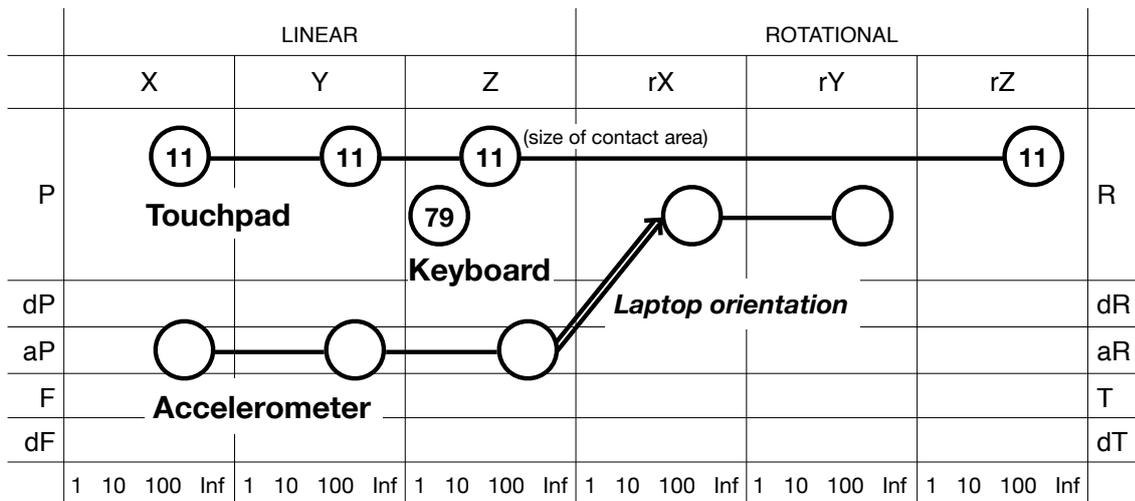
FIGURE 8.2: Input devices of the MacBook plotted against the taxonomy of input devices of Mackinlay et al. (1990). The webcam, microphone, and ambient light sensor are not represented. Black lines represent merge compositions, and the double-lined arrow represents a connection composition.

pressed. Most keyboards use a matrix design to register key presses (Zhang and Li, 2014). Each press of a key makes contact at an intersection of a row and a column wire on the matrix. The presence of absence of a contact on each row wire is checked for each column wire successively, which allows the keyboard to recognize at least two simultaneously pressed keys. However, this design is unable to detect particular three-key press combinations, due to an issue known as "ghosting"[9]. On the other hand, the USB specification for Human Interface Devices (DWG, 2001) sets a limit of six non-modifier keys in keyboard report descriptors for "boot devices"—those that can be used before the operating system is loaded. While these limitations can be circumvented, this is not the case for MacBook keyboards, which can handle between two and six simultaneous presses.

A laptop touchpad typically senses the relative position of one fingers on its surface, but most touchpads provide more information that is largely unexploited. Synaptics touchpads commercialized after 2001 can provide the absolute position of a finger with a sensitivity up to 2000 CPI, together with the approximate size of the contact area (Synaptics, 2001). Newer touchpads also provide the same information for a second finger (Synaptics, 2011). MacBook "trackpads" can sense the absolute position of up to 11 fingers, along with the approximate size and orientation of each contact area, when obtained using the API of the `MultitouchSupport` private framework.

Provided subsequent processing, microphone and webcam inputs can also be used. Audio signal from the mono or stereo microphone can be obtained using low-latency audio drivers

---

[9]The following page provides an explanation for ghosting phenomena and includes a tool allowing to find undetected key combinations: `http://www.microsoft.com/appliedsciences/antighostingexplained.mspx`.

	Parallel inputs	Degrees of freedom	Control granularity	Reliability	Responsiveness
keyboard	2 to 6	1	binary	++	+
touchpad	1 to 11	2 to 5	continuous	++	+
microphone	1 to 2	variable	continuous	+	+
webcam	variable	variable	variable	-	-
accelerometer	1	3	continuous	+	+
ambient light sensor	1	1	continuous/discrete	-	--

FIGURE 8.3: Properties of the standard input devices of a "unibody" MacBook Pro for musical performance.

(e.g., ASIO4ALL[10] for Windows, or CoreAudio for OS X) and processed in various ways, such as envelope following (averaging the amplitude of the signal over time) or pitch detection (estimating the fundamental frequency of the signal). The reliability of such techniques might vary depending on the environment and the properties to be estimated. Webcams can drive computer vision techniques, like fiducial marker tracking (Wagner and Schmalstieg, 2007), but they typically run at frame rates below 30 frames per second and their reliability highly depends on lighting conditions.

Laptops feature other sensors that might provide additional input for musical performance. MacBooks with hard disk drives are equipped with a three-axis accelerometer intended to detect impacts and disengage the heads of the hard drive to protect it from damage. Accelerometers sense linear acceleration, from which approximate pitch and roll angles can be derived (Pedley, 2013). Singh (2006) used the MacBook accelerometer to approximate the laptop's orientation at 60 Hz. Another sensor reports the amount of ambient light (at an approximate rate of 5 Hz on our machines), which is normally used to automatically adjust the brightness of the display. However, the value reported by an ambient light sensor changes tremendously depending on the lighting conditions. Some libraries, like Qt[11], report the amount of ambient light as a categorical variable instead.

8.2.2. MATCHING DEVICES AND INSTRUMENTAL GESTURES

An important goal of this section is that the guidelines developed here should result in designs allowing real-time performance similar to traditional musical instruments. Short of committing too early to a particular type of musical output, there are general aspects of instrumental performance that each of the laptop's inputs may or may not satisfy. Our focus is on the gestures that can be supported, and how to support them best with the input devices of a standard laptop.

GESTURES FOR TONE CONTROL

At the most basic level, the targeted designs must support the production of low-level musical events (*tones*, or notes) and the control of their unfolding in time. With acoustic instruments,

---

[10]http://www.asio4all.com. Last accessed 06/09/2015.
[11]http://doc.qt.io/qt-5/qambientlightreading.html. Last accessed 06/11/2015.
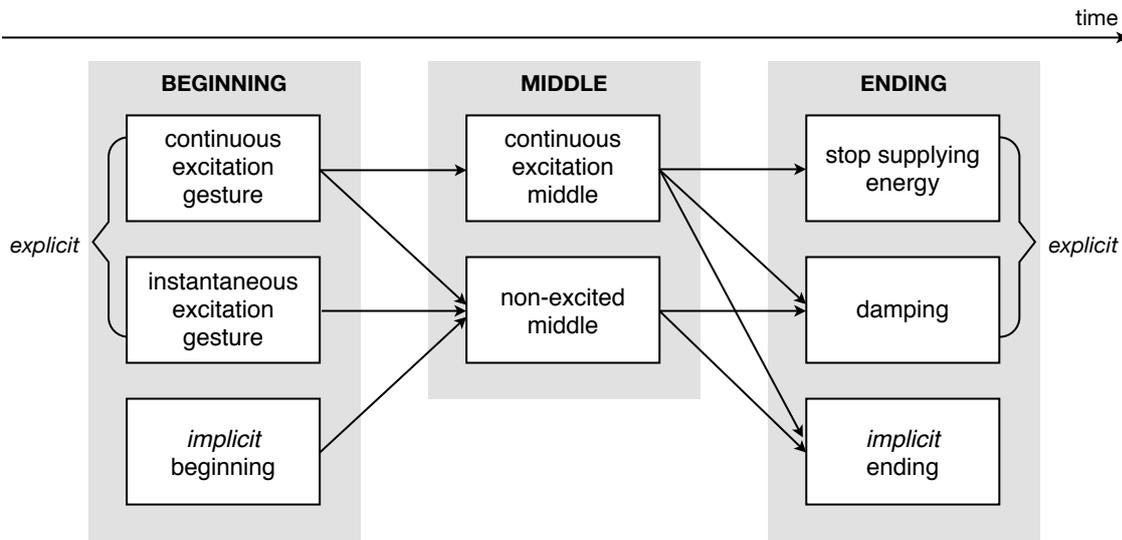
FIGURE 8.4: Simplified version of the model of temporal control of low-level musical events of Levitin et al. (2002).

one generally uses various gestures that have different roles with regard to the temporal structure of a tone. Levitin et al. (2002) proposed a model for gestural control at this level where a tone is made of three consecutive phases: *beginning*, *middle*, and *ending*[12]. The beginning and ending phases can be implicit if they are performed in the middle of a musical phrase. The model consists, for each of the three phases, in detailing how it can be controlled from a gestural point of view, with decreasing degrees of abstraction. Figure 8.4 presents a simplified version highlighting the different types of phases relevant to our discussion and the possible transitions between each phase according to the model.

This model, and the examples of gestures it provides, can be useful to determine the possible contribution of each of the laptop's devices at the three phases of a tone (Figure 8.5). Different gestures control the beginning of a tone depending on the type of excitation: bowing, rubbing, scraping, blowing, shaking, or swishing are used for continuous excitation beginning, while striking, plucking, or pressing are used for instantaneous excitation. Examples of gestures controlling the middle of a tone in acoustic instruments include deformation by bending, stretching, or squeezing, hand movements such as wiggling, sliding, with variations of speed and pressure, and mouth and tongue movements (lipping and tonguing). Finally, while the ending of a tone can result from the performer stopping supplying energy, it may also be due to damping.

GESTURAL FUNCTION ANALYSIS OF LAPTOP INPUTS

To complement these considerations, we will take a more general view on possible gestures and detail how each input provided by the devices of a laptop can be used to support them. The

---

[12]Levitin et al. (2002) prefer these neutral terms over the acoustically-tainted *attack*, *steady-state*, and *decay*. They also include a prior process involving cognitive planning (*selective preconditions*) and an end state (*terminus*), that we omit for our discussion.

	Keyboard	Touchpad	Microphone	Webcam	Accelerometer	Ambient light sensor
Explicit beginning (continuous excitation)						
Bowing		X		X		
Rubbing/scraping		X		X		
Blowing			X			
Shaking/swishing				X	X	X
Explicit beginning (instantaneous excitation)						
Striking	X	X	X	X	X	
Plucking	X	X		X	X	
Pressing	X	X		X	X	
Middle						
Bending/stretching/squeezing				X		
Wiggling/sliding/modulating speed		X		X	X	
Lipping/tonguing			X			
Modulating pressure	X	X			X	
Explicit ending						
Damping		X	X			

Figure 8.5: Examples of gestures controlling the beginning, middle, or ending of a musical event from the model of Levitin et al. (2002), and the input devices supporting them.

typology of instrumental gesture of Cadoz and Wanderley (2000)[13], together with the notion of ancillary gesture of Wanderley (1999), provides a high-level classification that we will use to assess the suitability of a particular input for particular instrumental functions. The outcome might differ between two projects, as it depends on the designer's priorities and goals. In our case, we determine suitability based on the device properties relevant to real-time control (Figure 8.3) and other considerations that will be detailed in the following paragraphs. The result of the analysis, shown in Figure 8.6, will be used to guide the allocation of the different inputs to complementary gestural functions prior to designing mappings with a particular synthesis method.

To support the ergotic function of excitation gestures, the energy transmitted through the instrument must be apparent in the sound phenomena produced. Input devices that sense excitation gestures should then be responsive—so as not to break the sense of causality between the user's action and the resulting sound—and have enough control granularity to give the performer the impression that sound is related to amount of energy transmitted through the gesture. Traditional instruments also give energy back to the performer through tactile and kinesthetic feedback, which help the performer control the instrument (O'Modhrain and Chafe, 2000). While current laptop devices lack actuators, the passive feedback they provide can be leveraged. For instance, contact with the touchpad surface or key presses might be better suited to percussion or picking than moving the finger past a threshold velocity. However, due to their limited control granularity, keys fail to transmit excitation energy in a satisfactory manner.

---

[13]Note that we do not include the *biasing* function—the need to maintain a part of the instrument in a certain state in order to produce sound—despite its necessity in some acoustic instruments. We will rather consider biasing in the design of convergent mappings (Rovan et al., 1997).

| | Excitation | | | Modulation | | | |
| | Instantaneous | | | | | | |
	Percussion	Picking	Continuous	Parametric	Structural	Selection	Ancillary
Keyboard							
Key press/release	3	2			1	1	
Touchpad							
XY position		2	2	1	2	2	
XY velocity	2		1	1		3	
Contact area size	1	1	1	1			
Finger orientation				2	2	3	
Microphone							
Amplitude	2		1	2			
Discrete pitch					2	2	
Webcam							
XYZ pos. of fiducial marker	3	3	3	3	3		2
Accelerometer							
Roll/pitch laptop orientation	1		1	2		3	1
Ambient light sensor							
Amount of ambient light				3	3		3

FIGURE 8.6: Suitability of usable inputs of the MacBook to the instrumental functions identified by Cadoz and Wanderley (2000). Microphone and webcam inputs are suggestions. Numbers must be read per column: a smaller number in a cell indicates better suitability of the input to the instrumental function. Blank cells are possible but less desirable choices.

Modulation gestures have a non-ergotic relation to sound: the energy expended does not contribute to the resulting sound (Cadoz and Wanderley, 2000) as it does for excitation gestures. The importance of responsiveness varies depending on the particular role of a modulation gesture in sound. Other criteria differ depending on their parametric or structural nature. With parametric modulation, the emphasis is on control granularity and reliability. Devices with a high number of degrees of freedom provide an opportunity for the integral control of several parameters. With structural modulation, timing accuracy and a sense of control are important: these properties are even critical when using a sostenuto pedal, which sustains only the notes that are played at the moment the pedal is activated. Thus, the emphasis in the suitability of laptop inputs for structural modulation gestures is rather on responsiveness and reliability.

Selection gestures do not provide energy to the resulting sound, nor do they modify the properties of an instrument. They consist in choosing between equivalent parts of an instrument (e.g., strings in a guitar, or keys in a piano). Responsiveness, reliability, and feedback are the most important properties to support selection gestures. In addition, selection can be parallel, in which case the number of parallel inputs a device can handle is important too.

Ancillary gestures concern the movement of the laptop in space, and only a limited number of devices can provide this information more or less directly: the accelerometer, the webcam coupled with motion estimation techniques (such as optical flow measurement), and the ambient light sensor. Wanderley (1999) showed that ancillary gestures influence the amplitude of the first partials (frequency components of the sound, as obtained by spectrum analysis) in the recorded
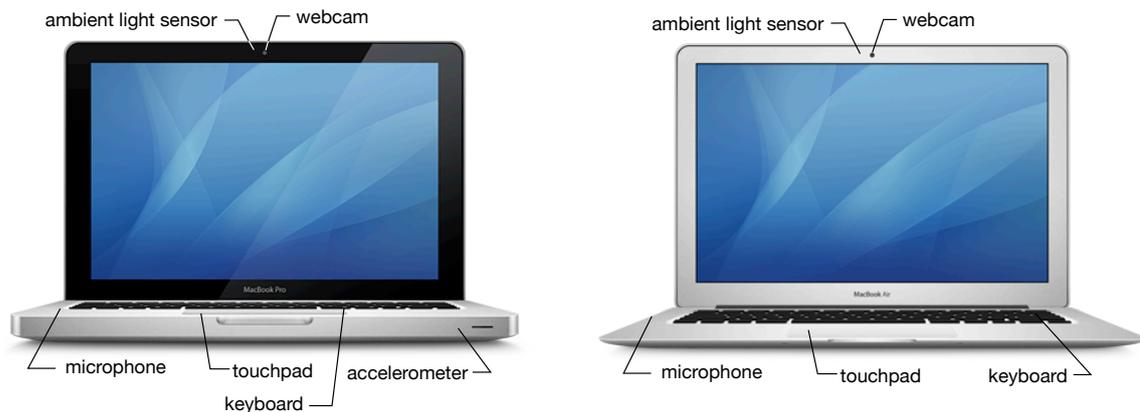
Figure 8.7: Location of the input devices in two considered MacBook models: a MacBook Pro 15" 2009 (left), and a MacBook Air 13" 2011 (right).

sound of acoustic wind instruments. If simulated, these effects must be tightly coupled to the performer's actions, which a device with high control granularity and responsiveness can best provide.

Note that this analysis is concerned with the relative suitability of inputs for each instrumental function, but this does not mean that a particular input should be assigned to a single type of gesture. Indeed, gestures often combine multiple functions: for instance, the bowing gesture on a cello both selects a string and excites it (Cadoz and Wanderley, 2000). The outcome of our analysis (Figure 8.6) is partially in line with the choices of Fiebrink et al. (2007). Using the keyboard to select between different pitches, bowing on the touchpad more or less rapidly to continuously excite sound synthesis, and changing finger position as a parametric modulation of the resulting sound location in space prove to be adequate choices, but Figure 8.6 would suggest other inputs than microphone amplitude for switching between two filter frequencies (structural modification).

8.2.3. PHYSICAL CONSTRAINTS AND SUPPORT

The physical location of input devices and the way the performer is holding the laptop form another important set of important considerations in the design of a laptop-based musical instrument. In the MacBook for example (Figure 8.7), the devices are laid out in such a way that the performer is only be able to reach some of them simultaneously (e.g., the ambient light sensor cannot be attained when both hands are on the keyboard). In addition, the particular posture of the performer may prevent her from interacting with certain parts of the laptop, limit her ability to express herself with ancillary gestures, and prevent the audience to understand what is going on.

We consider four postures that we have used extensively when practicing with the designs presented in the next section:

	Limb availability for interaction	Support for body movement	Display visibility
Desk	++	-	+
Lap	+	-	+
Held	-	+	+
Held-Mouth	-	++	-

FIGURE 8.8: Tradeoffs between desirable properties for different laptop postures.

Desk This position is similar to that of a desktop computer user. The performer is sitting or standing, with the laptop in front of her on a stable horizontal surface, and the forearms supported by the desk.

Lap Instead of being on a desk, the laptop rests on the laps on the sitting performer.

Held The performer holds the laptop with the left hand, its right edge being blocked by the fingers of the left hand forming a hook grip, while the forearm, elbow, and palm of the left hand support the back of the laptop. The right hand is free to interact with the keyboard and touchpad.

Held-Mouth This position is similar to the previous one, except that only the left thumb or the palm of the left hand support the back of the laptop. The left edge (where the microphone is, Figure 8.7) is put in front of the performer's mouth, and the bottom-left corner may rest on her chest.

As we will detail in the following, each posture entails different tradeoffs between *limb availability for interaction*, *support for body movement*, and *display visibility* (summarized in Figure 8.8). While other considerations such as comfort might be involved, we limit ourselves to the three aforementioned properties.

LIMB AVAILABILITY FOR INTERACTION

Depending on its location, a single non-moving hand can act on several devices at a time. When the fingers are above the keyboard, the performer can still press the touchpad using the palm, and the thumb around the thenar eminence (at the base of the thumb) and the first metacarpal. When the fingers are above the touchpad, they can still easily reach the bottom keys of the keyboard (modifier keys and SPACE). The left hand alone can simultaneously rub the microphone (if placed on the left edge or left side of the laptop, as is the case for MacBooks), hit the leftmost keys of the keyboard, and touch the upper-left area of the touchpad.

The least constraining posture is one where both hands are freely movable and available for interaction (*Desk*), but this is not always possible because some part of the limbs may already serve other functions: in the *Held* posture, the left hand must support the laptop, which leaves few opportunities for interacting with the input devices. The BiTouch design space (Wagner et al., 2012), initially designed for hand-held tablets, might help rationalize possible choices.

BiTouch is based on the kinematic chain model of Guiard (1987), who likened asymmetric bimanual action to the behavior of two contiguous elements of a kinematic chain, such as an
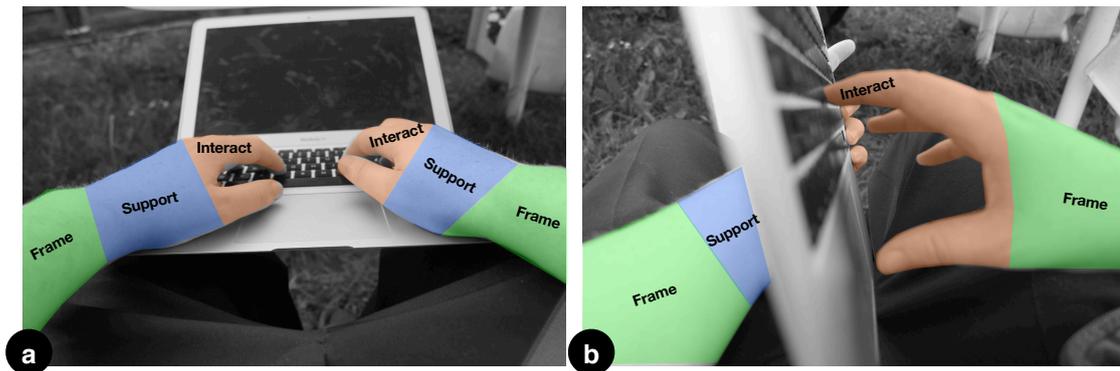
FIGURE 8.9: Framing, support, and interaction functions of the kinematic chain formed by upper limbs visually described in the BiTouch design space (Wagner et al., 2012) for two postures: (a) *Lap*, and (b) *Held-Mouth*.

arm: the proximal element precedes the distal one in action, it establishes its reference frame, and operates at a larger spatial and temporal scale. It ensues that both upper limbs have their place between more proximal (e.g., shoulders) and more distal (e.g., mouth) elements in the kinematic chain, which is allocated to successively refined functions. Wagner et al. (2012) consider three functions: establishing a spatial reference frame (*framing*), securing and stabilizing the device (*support*), and interacting with the device (*interaction*). Remembering that both limbs take part in the kinematic chain (the non-dominant before the dominant one), a description of its allocation to the three functions in the BiTouch design space will at least detail, for each function, its *location* in the kinematic chain (framing occupies the most proximal ones, interaction the most distal ones, and support, if any, is in the middle) and its distribution (one or more body parts, symmetrical or not). Limb support can be complemented with or replaced by independent support, as is the case with our *Desk* posture.

Figure 8.9 illustrates upper limb allocations for the *Lap* and the *Held-Mouth* postures. With *Lap*, the laptop may be unstable and the performer might have to press the palms against the wrist rest areas to secure its position, leading to less hand movement freedom. Once the framing and support functions have been allocated, a useful characterization of the remaining distal interaction function is in terms of the degrees of freedom available for movement. For example, in the *Held* and *Held-Mouth* postures with a MacBook, the only remaining possibility for left hand interaction is to press the rightmost keyboard keys ($\leftarrow$, $\hookleftarrow$, $\Uparrow$, and $\rightarrow$).

SUPPORT FOR LAPTOP AND BODY MOVEMENT

While interaction is constrained when distal elements of the kinematic chain are "single-handedly" in charge of the support function, this leaves the more proximal elements free to sway. What the performer loses in hand and finger movement freedom, she gains on opportunities for bodily expression and visibility to the audience. This turns out to be a notorious issue in laptop performance (Dahl, 2012). In the words of @collins03generative:

An artist using powerful software like SuperCollider or PD cannot be readily distinguished from someone checking their e-mail whilst DJing with iTunes.

Performing "ancillary" gestures on the *Desk* might seem forced and awkward. In addition, it demands energy and movement expenditure (e.g. leaving the touchpad to grasp and shake the display frame). *Held* and *Held-Mouth* leave the performer free to sit, stand, or dance during the performance, allowing a greater variety of laptop and bodily gestures.

DISPLAY VISIBILITY

A final issue to be considered is that of display visibility. While we have left out dynamic visual feedback in the designs presented in the next section, a display is a valuable resource anyway. It can bring peripheral clues for important events (e.g. "when the screen turns blue, it is your turn to solo"), and offers the opportunity to glance at various documents relevant to the ongoing musical performance, such as a setlist or a musical score. The *Held-Mouth* position, where the left side of the laptop faces the performer, is limited in this regard.

In addition, the versatility of the laptop in musical performance calls for more advanced uses that require a display. If the design allows it, the performer might be able to momentarily break out of "performance mode" and engage with knowledge-based or rule-based musical behavior in order to adjust the sound-producing structure (e.g., adding an instrument or an effect) by programming or graphical interaction, launch the recording of the next couple of measures and control previously recorded loops, or direct and negotiate privately with other performers through network communication.

8.2.4. KEYBOARD LAYOUT DESIGN

The previous gestural function analysis has identified the keyboard as a potential device to support selection between a large number of equivalent elements. As selection is likely to concern pitches, an important concern for design is their spatial distribution over the laptop keyboard. Spatial note layout design has a long history and concerns not only the traditional Western twelve-tone tuning system (12-TET) but also various non-standard and "microtonal" tunings (Keislar, 1987; Milne et al., 2007). A noteworthy property of some layouts is their *transpositional invariance* (Keislar, 1987); that is, the fact the fingering for any interval (two different pitches) maintains the same spatial relationship independently of the base key: the layout of a bass guitar is transpositionally invariant, while that of the piano is not. Transpositional invariance has been explored in a variety of "isomorphic" interfaces (Maupin et al., 2011; Milne et al., 2011; Bigo et al., 2012; Park and Gerhard, 2013). Other desirable properties may factor in, such as economy of movement and the opportunity to reuse existing musical skills. In the following, we will describe some of the tradeoffs involved in the design of a musical laptop keyboard layout in the 12-TET tuning.

We have experimented with four layouts (depicted in Figure 8.10 over an AZERTY keyboard) that occupy the four middle rows of the keyboard containing alphanumeric keys:
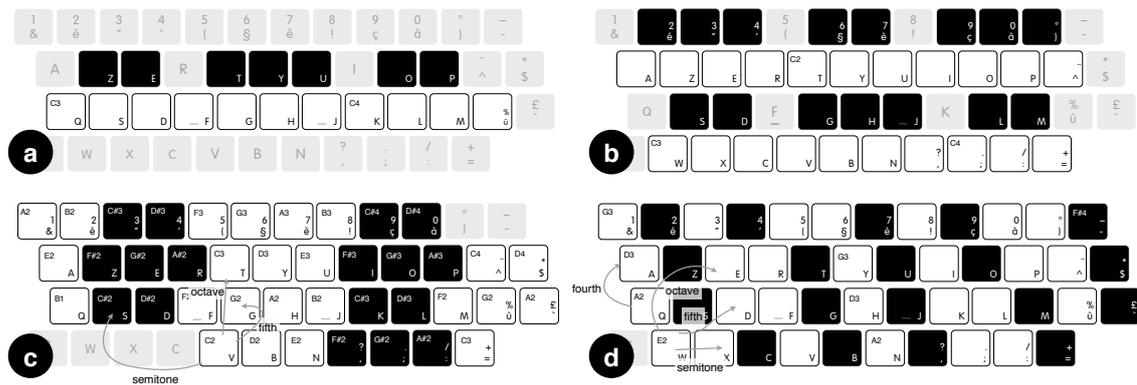
FIGURE 8.10: Four spatial note layout designs: (a) Piano-2line, typically found in sequencer software; (b) Piano-4line; (c) Wicki-Hayden, an isomorphic layout; and (d) Guitar.

Piano-2line The first one (Figure 8.10a), is commonly found in sequencer software, such as Garageband[14], Logic Pro[15], or Ableton Live[16], as an aid for musical composition and quick testing of "virtual instruments" (synthesizers). It mimics the layout of piano keyboards, with white keys laid out on the home row, below the black keys, giving a limited musical range of one octave and a half. In sequencer software, surrounding rows are often used to trigger changes in musical parameters, such as transposition, pitch bend, or the generic "modulation wheel" parameter defined in the MIDI specification (MMA, 1995).

Piano-4line The previous layout can be doubled (Figure 8.10b) to use the entire space offered by alphanumeric keys, offering twice the musical range. However, the tactile landmarks of the home row are not aligned anymore with a row of white keys.

Wicki-Hayden This isomorphic layout (Figure 8.10c) has been used in several commercial controllers such as the Thummer[17] and Musix (Park and Gerhard, 2013), and examined by Milne et al. (2007) and Maupin et al. (2011). Consecutive keys in a row advance by two semitones, and the fifth of any key is the immediate upper-right key.

Guitar This layout transposes the fingering of a bass guitar (Figure 8.10d), where consecutive keys in a row advance by one semitone, and the fourth of any key is the immediate upper-left key.

These different layouts can be compared in several respects (Figure 8.11). A first figure of merit is the *musical range* (Blaine and Fels, 2003) offered by the layout: although transposition keys allow the performer to dynamically adjust its position on the pitch continuum, an extended musical range it is certainly desirable. *Piano-2line* only offers 18 semitones from the lowest to the highest playable note, *Piano-4line* doubles to 36 semitones, and the two other layouts are in the middle with 27 semitones. In comparison, a guitar fretboard spans about 46 semitones and a piano keyboard spans 88 (more than seven octaves).

---

[14]https://www.apple.com/mac/garageband/. Last accessed 06/14/2015.

[15]https://www.apple.com/logic-pro. Last accessed 06/14/2015.

[16]http://www.ableton.com. Last accessed 06/14/2015.

[17]https://en.wikipedia.org/wiki/Thummer_keyboard. Last accessed 06/12/2015.

	Musical range	Economy of movement	Transpositional invariance	Skill reuse
Piano-2line	-	+	-	++
Piano-4line	++	-	-	+
Wicki-Hayden	+	+	+	-
Guitar	+	++	+	++

FIGURE 8.11: Tradeoffs in the design of a musical keyboard layout.

Economy of movement can also be an important factor, especially if the hands are already assigned to support functions and cannot move freely. One aspect of economy of movement is related to redundancy: layouts where several keys are assigned to the same note (*Wicki-Hayden* and *Guitar*) might reduce the distance between the current note and the next one. Another aspect to consider is that each layout facilitates or hinders the execution of different melodic and harmonic structures. The *Piano-2line* layout makes the one-handed play of some neighboring tones difficult because the piano keyboard is split into two parts. With the *Wicki-Hayden* layout, chromatic progressions require ample movements compared to *Guitar* and *Piano-1line*.

Finally, *transpositional invariance* and *skill reuse* mostly concern learning. Transpositional invariance, described above, is a property of isomorphic layouts like *Wicki-Hayden* and *Guitar*, and may facilitate learning because any fingering always produces the same interval or chord. A layout can also reuse, to some extent, the existing motor and cognitive skills developed by a performer with another instrument, which is what *Piano-2line* and *Guitar* do.

8.3. Office music

Based on the previous considerations, we designed three laptop-based musical instruments that demonstrate the potential of our approach and have been used successfully in several public live performances. These designs unite in the notion of *office music*, a vision of "laptop music" that emphasizes the main role of the laptop as an instrument that can be "played" in a way similar to traditional acoustic instruments. Our practice of office music relies on the stability of control mechanisms, so as not to hinder learning and mastery (Dobrian and Koppelman, 2006), the conceptual and physical similarity of instrumental gestures to other instruments and everyday interactions (Jacob et al., 2008), the opportunity of micro-diversity control (Jordà, 2005), despite the limitations of the laptop, and the compatibility with other uses of the laptop during performance.

In this section, we will describe the design and implementation of our three instruments: the Pianotant, a piano-inspired polyphonic instrument, the Soufflant, a monophonic wind instrument, and the Tapant, a non-pitched percussion instrument for use with modern drum kit simulations.
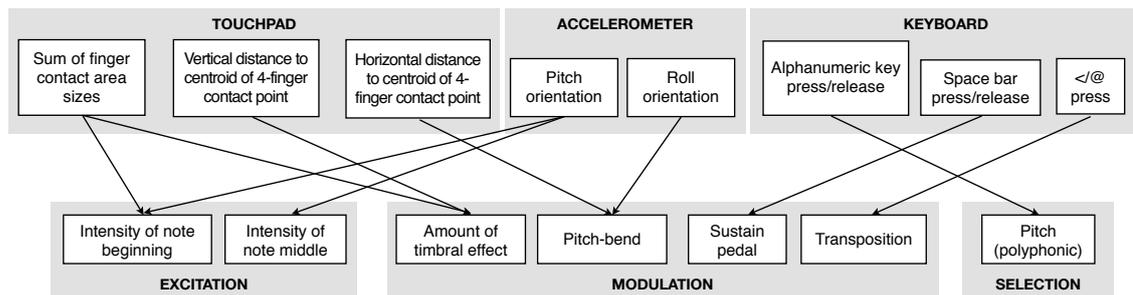
FIGURE 8.12: Mapping between input and intermediate parameters for the Pianotant.

8.3.1. PIANOTANT: A POLYPHONIC PIANO-LIKE INSTRUMENT

The Pianotant is a polyphonic instrument with which the performer produces and controls synthesized tones mainly by pressing keyboard keys. It is designed to be used mainly with both hands in the *Desk* and *Lap* postures, although it can be also used in the *Held* posture with the right hand. Figure 8.12 shows the mapping between laptop inputs and the intermediate parameters that are used to drive synthesis. We will not detail the mapping between intermediate and sound parameters as they largely vary depending on the type of synthesis.

In a typical resting posture, both hands are above the keyboard, their base lying above the touchpad without touching it. The keyboard is mainly used to to select pitches with the left hand, the right hand, or both, using the *Guitar* layout described earlier. Just pressing one or more keys triggers the beginning of the corresponding notes with low intensity. The notes resonate until the keys are released. The location and length of the SPACE bar makes it easy to press with a thumb while playing the rest of the keyboard, thus it is used as a sustain pedal: the notes triggered and activated when the space bar is held are sustained until the space bar is released. In addition, pressing the leftmost keys in the first and fourth alphanumeric rows (@ and < in the AZERTY layout we used) transposes the whole keyboard to the upper or lower octave.

The base of the hands being typically above the touchpad, they are used to define the intensity of the beginning of a note (using the sum of contact area sizes). This way, the performer usually produces a single tone using an excitation and selection gesture consisting in selecting and pressing the relevant keyboard key while also pressing more or less on the touchpad with the base of the hand. The sum of contact area sizes can also be used to drive the intensity of an effect on the timbre of the middle of the notes currently played. When four fingers are simultaneously on the touchpad[18], their centroid point is calculated, and its distance from the centroid of the contact point controls the intensity of another effect (vertical distance) and the relative continuous variation of pitch (pitch-bend, horizontal distance).

If the laptop has an accelerometer and the performer uses the *Lap* posture, a hard key press will change the pitch orientation sensed, which controls the intensity of the note beginning. Inclining

---

[18]We chose four fingers in order to limit the risk of accidental activation.
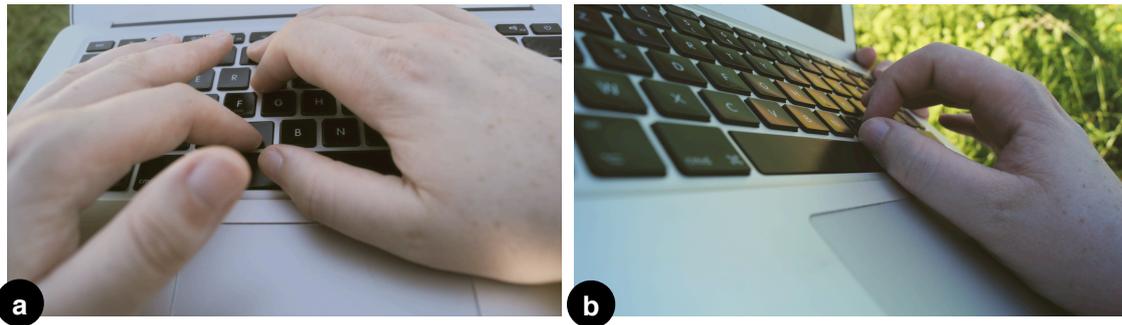
FIGURE 8.13: Playing the Pianotant: (a) executing a chord in the *Lap* position; (b) executing an arpeggio with sustain in the *Held* position (see text for details).
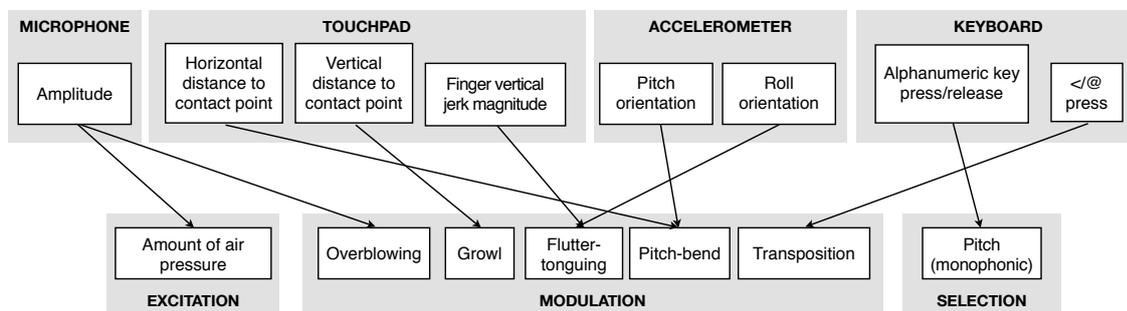
FIGURE 8.14: Mapping between input and intermediate parameters for the Soufflant.

the laptop to change its roll orientation (e.g., by lifting one leg) provides another way to produce pitch-bend effects, such as vibrato.

Figure 8.13 shows two close-ups of a Pianotant performance. In Figure 8.13a, the performer executes a loud G minor 7 chord by pressing the [V], [T], and [( ] keys (in the AZERTY layout) with both hands while strongly pushing the touchpad using the thenar eminence of her right hand. During the chord, the performer alternates pressure with both hands to shake the laptop on his laps and produce a vibrato effect. In Figure 8.13b, the performer is in the *Held* posture, with only his right hand free, and gently arpeggiates a A# minor chord with keys [,], [=], [L], and [O], exerting a light pressure on the touchpad, while holding the [SPACE] bar to sustain the tones.

8.3.2. SOUFFLANT: A MONOPHONIC WIND INSTRUMENT

The Soufflant is a monophonic instrument similar to wind instruments: energy is continuously transmitted to the instrument through the performer's breath, instead of being the result of an initial hit. Due to its reliance on breath, the Soufflant is mostly played in the *Held-Mouth* posture in order to position the microphone in front of the performer's mouth. Figure 8.14 shows the mappings between laptop inputs and intermediate parameters.

To perform a note, the performer selects a pitch on the keyboard using the right hand. The keyboard uses a *Guitar* layout, like the Pianotant, but only the highest of all selected pitches
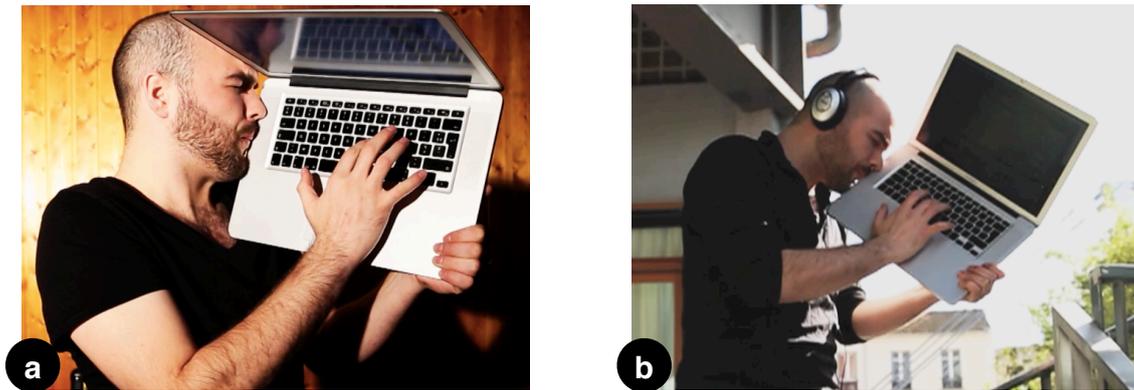
FIGURE 8.15: Performance with the Soufflant: (a) producing a single sustained tone with gradual intensity; (b) simultaneously controlling growling, pitch-bend, and flutter-tonguing effects (see text for details).

is taken into account at a given moment, which allows the performer to articulate a sequence of notes without separation (*legato*). Like with the Pianotant, the leftmost keys in the first and fourth alphanumeric rows are transposition keys.

However, until the performer is blowing into the microphone, selecting a pitch with the right hand produces no sound, maybe with the exception of the sound of a key clapping over a tone hole in the case of a realistic simulation of a woodwind instrument. The amplitude of the microphone input is computed using envelope following and continuously drives the amount of air pressure or the intensity of the synthesized sound. Thus, to produce any tone the performer has at least to blow into the microphone and to select a pitch on the keyboard. When blowing harder, the detected microphone amplitude crosses a certain threshold, leading to overblowing in a way similar to simple woodwind instruments: the notes currently played shift to the upper octave until the amplitude comes back under the threshold level, giving the performer a way to extend the musical range of the instrument without using transposition keys.

The performer can also use the base of her right hand or the right thumb to perform various effects. Modulating the horizontal distance from the original contact point with the surface controls a pitch-bend effect (a similar effect can also be produced by orienting the laptop towards the left or right), while modulating the vertical distance controls a growling effect. Finally, quick vertical oscillations on the surface simulate a more or less pronounced flutter-tonguing effect[19], which can also be controlled by orienting the laptop downwards.

Figure 8.15 shows two examples of a Soufflant performance using a 15-inch MacBook Pro, with the microphone on the left side of the keyboard. In Figure 8.15a, the performer holds the laptop in a neutral position and produces a soft D#3 tone by pressing the J key and softly blowing into the microphone, then achieves a crescendo effect by gradually blowing harder. In Figure 8.15b, the performer produces a D3 tone, pushes the top of the touchpad with the base of the

---

[19]In wind instruments, flutter-tonguing is acheived by quickly rolling the tip of the tongue.
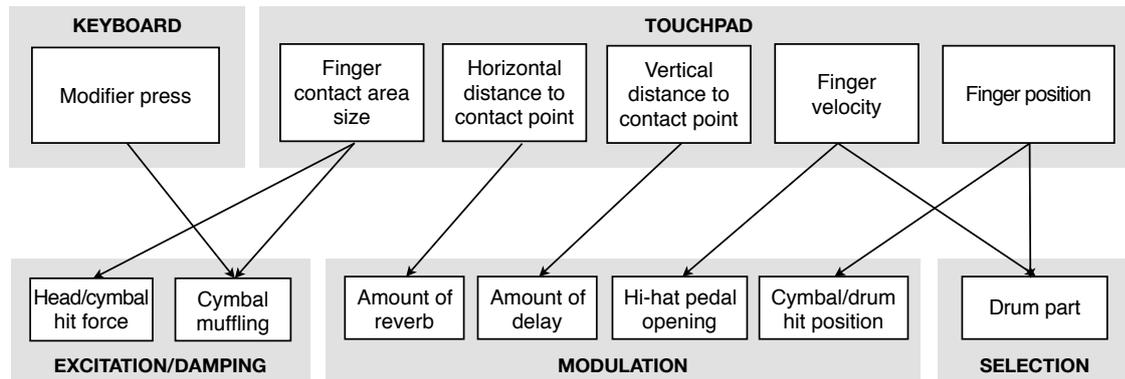
FIGURE 8.16: Mapping between input and intermediate parameters for the Tapant.

right hand, and moves it to the bottom of the touchpad in order to obtain a growling effect. In the same time, he orients the laptop towards the lower right to produce a slight variation of pitch and a flutter-tonguing effect, resulting in a harsh and dissonant sound.

8.3.3. TAPANT: A DRUM INSTRUMENT

The Tapant is an unpitched percussion instrument with which the performer can play drum parts for contemporary popular music in a way similar to modern drum kits. It is designed to be used in the *Desk* and *Lap* postures, with the fingers of both hands hitting the touchpad. Figure 8.16 shows the mappings between laptop inputs and intermediate parameters.

Both hands are placed around the touchpad so that the tips of the index and middle fingers rest above the upper-left and upper-right corner, while both thumbs are placed near the bottom edge. The Tapant divides the touchpad in nine areas, each assigned to a different part of the drum kit (Figure 8.17). Due to their prominence in popular music, the bass drum, the snare drum, and the hi-hat have been made the most accessible parts and occupy larger areas than other parts of the kit. This way, drum rolls and other percussion techniques can be easily performed with the fingers.

To hit a drum part, the performer strikes the surface of the touchpad in the relevant area. The intensity of the beginning of the drum sound produced is proportional to the size of the contact area: gentle touches with the tip of the finger produce soft drum sounds, while surface contact with a greater amount of skin area results in louder, harder hits. The position of the hit on the area has an influence on sound for the hi-hat and the ride cymbal, allowing to simulate cymbal hits closer to the edge or to the bell. A similar mapping occurs with the position of a hit on the snare drum area, simulating a stroke more or less close to the rim of the simulated snare drum.

The tangential velocity at which the finger strikes the surface is also taken into account in the mapping. For most of the areas defined, a "swipe" hit with tangential velocity above a certain threshold produces an accented hit sound, often a "rimshot" one[20], and a hit on the bell for the

---

[20]With acoustic drums, a rimshot is produced by striking the head and the rim of the drum at the same time with a drum stick.

hi-hat *(open hi-hat)*	crash cymbal *(splash cymbal)*	hi-hat *(open hi-hat)*
ride cymbal *(ride bell)*	snare drum *(rimshot)*	cross-stick *(cowbell)*
high tom *(high tom rimshot)*	floor tom *(floor tom rimshot)*	
bass drum		
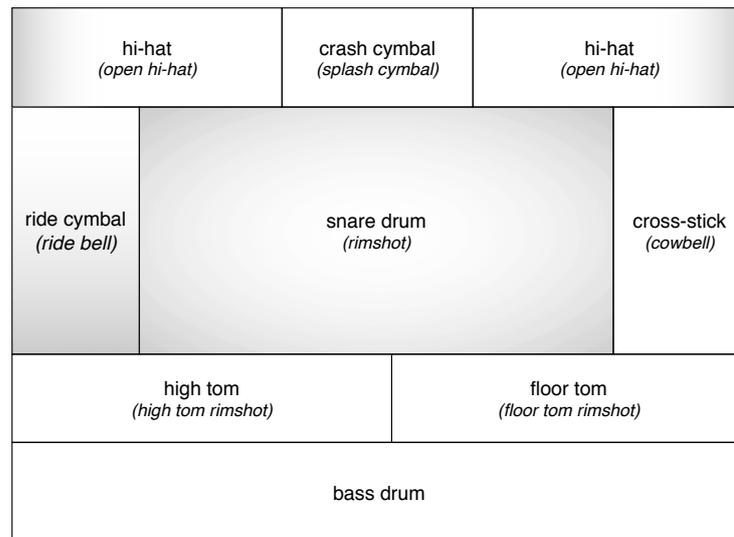
FIGURE 8.17: Mapping of drum parts on the touchpad surface for the Tapant: normal text indicates the drum part assigned to an area when hitting the surface with a near-zero incidence angle, text in italics indicates the drum part assigned when hitting the surface with a high tangential velocity, and a gradient shows that the position of the hit on the area is taken into account in sound.

ride cymbal. For the snare cross-stick and crash cymbal areas, a swipe selects and excites another part of the drum kit: for example, the area assigned to the cross-stick sound can also be used to produce a cowbell sound. This allows to select a larger number of drum parts despite the touchpad being relatively small. For the hi-hat, the tangential velocity determines sound in a more subtle way, as it defines the extent to which both cymbals composing it are close together when the hit occurs, allowing a wide range between the "closed" and "open" hi-hat sounds.

The above interactions all concern the moment where a finger enters in contact with the surface of the touchpad, as it usually leaves the surface immediately afterwards. We tried to exploit prolonged contacts to increase the amount of mid-diversity control of the instrument. The performer controls the amount of reverberation and delay effects applied on drum sounds by sliding a finger, receding from its original contact point. The effects cease to apply when the finger leaves the surface. In addition, when the finger hits a cymbal area and stays in contact with the surface, lifting the finger muffles the ringing cymbal. The keyboard can also be used to muffle cymbals, each modifier being assigned to a different one.

Figure 8.18 shows three close-ups of a Tapant performance. In Figure 8.18a, the performer executes the beginning of a simple rock drum pattern, simultaneously hitting the bass drum with his right thumb and the hi-hat with his right middle finger, and preparing the left middle finger to strike the snare drum at the next beat. In Figure 8.18b, the performer has already placed his thumb on the surface and moved it to the left so that the cymbal hit performed by his left middle finger resonates with large amounts of reverberation. In Figure 8.18c, the performer is in the middle of a drum fill and performs a snare drum roll pattern with the left index and the right index and middle fingers.

FIGURE 8.18: Performance with the Tapant: (a) during a simple rock pattern; (b) control of a timbral effect using the right thumb; (c) execution of a snare drum droll (see text for details).

8.3.4. LIVE LOOPING AND HIGHER-ORDER INTERACTION

The three above instruments are available after entering a "performance mode" by clicking a button or hitting a dedicated hotkey. When in performance mode, the performer can switch between the three instruments using the arrow keys. We also implemented a series of hotkeys to create and control loops from the performance, change the overall tempo, and undo and redo the previous changes. For example, to trigger or stop the recording of a loop, the performer must press ⌘SPACE, two consecutive keys that can be easily reached in any of the normal playing positions for our three instruments. Holding the ⌘ modifier while pressing arrow keys allows to navigate through the recorded loops, which can be restarted using ⌘SPACE, stopped using ⌘↵, and deleted using ⌘←. To accomplish more complex tasks, the performer must return to the normal mode of operation by hitting the ESC key during performance mode. In addition, the Fn modifier at the bottom right of the keyboard provides a quasimode to momentarily interact with the graphical interface, allowing quick adjustments during the performance.

8.3.5. IMPLEMENTATION

While early prototypes were implemented using Pure Data, our software is currently implemented using the Objective-C language on OS X. Normal keyboard and touchpad input are blocked using the Quartz Event Services. Keyboard input is monitored through the I/O Kit, touchpad input is handled using Apple's private multitouch API, and microphone input is provided by the CoreAudio API. Our software uses CoreMIDI to communicate with other applications that provide sound synthesis. We have mainly worked with Ableton Live 8 Lite, which provides the mixing and live looping functions controlled by our software using a "MIDI remote script" programmed in Python. Live also hosts the various virtual instruments we used that are responsible for sound synthesis. For the Pianotant, we mainly used MrRay73 Mark II[21], a simulation of electric piano based on physical models. For the Soufflant, we used brass and woodwind instrument simulations based on samples and physical models from Sample Modeling[22]. Finally, for the Tapant, we used XLN Audio's Addictive Drums[23] drum kit synthesis software.

---

[21]http://www.genuinesoundware.com. Last accessed 06/14/2015.
[22]http://samplemodeling.com. Last accessed 06/14/2015.
[23]http://www.xlnaudio.com. Last accessed 06/14/2015.

Because we sometimes did not have access to enough laptops to perform the tests and live performances described in the next section, we also built other implementations of the Soufflant for mobile phones and web browsers. The first implementation was an iOS application designed to function on an iPhone 3GS and a fifth-generation iPod Touch with a Bluetooth external keyboard. The mobile device was placed and secured under the keyboard thanks to a custom 3D-printed mount. The side of the device featuring a microphone barely protrudes from the left side of the keyboard, so as to enable a performer to play similarly to a laptop Soufflant. This implementation was functionally similar, however the touchscreen of the device was placed underneath the keyboard instead of being next to it, and performers had the tendency to hold the "phone Soufflant" by the top side of the keyboard. The second implementation was a HTML and Javascript web page using Web Audio APIs (for microphone input and sound output) that allowed a person with any kind of desktop computer to practice an impoverished Soufflant, due to the lack of web-based accelerometer APIs.

8.4. Discussion

In this section, we will detail the history of this project and the successful uses of the three prototypes presented above in various situations. We will then propose several examples of other physical uses of the laptop outside the domain of musical expression.

8.4.1. DESIGN PROCESS AND USE

The "office music" project started in December 2012, mainly out of discontent with existing ways of recording and testing musical ideas for jazz and electronic music in commercial sequencer software. We felt that too much time and unnecessary operations were involved between the appearance of an idea for a musical phrase or structure and its execution in an acceptable audible form, leading to a reluctance to try out ambitious ideas and to explore the space of musical possibilities. On the one hand, using external controllers provided us with rapid and effective ways to "play" with our ideas, but the constant need to get back to the laptop and the graphical interface to achieve any complex or high-level operation, and the continuous switch between the two separate worlds of skill-based and rule-based performance—which were more than an arm's length apart, in the case of the electronic drum kit—proved tiring and inefficient. In addition, the complex setup and the total size and weight of the equipment involved negated the advantages of the "all-in-one" mobile laptop that would allow us to discuss and test musical ideas anywhere and at any time. On the other hand, a laptop alone seemed to offer insufficient input dimensionality to try out musical ideas by directly playing them, and the process of testing variations of an idea parameter by parameter, in a sequential fashion, was even more frustrating. In sum, we were faced with a tradeoff between directness and expressiveness in a situation were the addition of input devices was unwelcome.

We first experimented with prototypes developed using the Pure Data environment. Our earliest efforts were dedicated to playing the drum kit. With one prototype, a performer could produce

drum hits with one or two mice with rapid and abrupt movements, the orientation and the chording of mouse buttons selecting the drum part to be hit. With another, the sound of various parts of the hand and objects hitting the laptop was analyzed in real-time using a vocoder algorithm and transformed the intensity and timbre of drum hit sounds in a way similar to the commercial Korg WAVEDRUM instrument: a stroke with the nail resulted in a brighter sound than a stroke with a softer part of the body, and gestures such as scratching and rubbing would produce interesting timbral variations on the original sound. Our use of the microphone led us to explore the control of simple synthesizers and wind instrument simulations with breath. We progressively converged towards early versions of the Soufflant and the Tapant, and discarded other designs as they would not provide enough mid-diversity to be useful as compositional aids or live performance substitutes for a particular class of instrument.

Rewriting the software in Objective-C was an important step that both provided enough robustness for prolonged use and permitted the development of a mobile phone version of the Soufflant, allowing us to accompany novice users during training sessions. During the last months of 2013, we tried to assess the potential of the Soufflant and the Tapant in a *transcription* activity (in the sense of the Cognitive Dimensions framework, Blackwell and Green, 2003) by reproducing a "brass band" style song (Greb and Dugnus, 2009) with drum and brass parts. This allowed us both to refine the mappings of the instruments and to begin to identify some of the situations where being able to quickly switch between musical performance and sequencer operation would be useful. As a result, we produced a short video demonstrated at the CHI'14 Video Showcase (Aceituno and Potier, 2014b).

Around February 2014, we also organized several individual practice sessions with three volunteer musicians: a trumpeter, a drummer, and a pianist, all three being exposed to the Soufflant and the Tapant. They were enrolled in the local music academy and had advanced knowledge of music theory and instrumental practice, allowing us to focus strictly on the learning and appropriation of our instruments. The sessions were interactive and informal, and alternated between free exploration, focused training (e.g., execution of drum fills), and two-person improvisation. We continued to revise the mappings of our instruments between sessions based on the participants' comments. In general, about half an hour was needed to become acquainted with the mappings and the spatial layout of the keyboard (for the Soufflant) and the touchpad (for the Tapant). Note that our participants were not guitarists. In the first sessions they mostly relied on visual feedback for pitch selection with the Soufflant. The trumpeter appreciated the freedom of movement but sometimes had trouble to align the microphone correctly. We therefore designed and 3D-printed wide and narrow mouthpieces to clip to the side of the laptop, which were tested in subsequent sessions. While they corrected the alignment problem and provided better feedback on air flow, they did so at the cost of a slightly restricted range of movement. With the Tapant, the unconventional mapping of contact area size to hit intensity (instead of force) was initially disturbing for the drummer. With all participants, the lack of feedback to delimit the different drum areas led to frequent errors, which we remedied by drawing their boundaries

on the touchpad with a pencil. More generally, despite the limitations of the "office music" approach compared to their respective acoustic instruments, participants were enthusiastic about our instruments and could play simple melodies and drum patterns after a few hours of training.

We continued the sessions during two months with the pianist, who participated with Ludovic Potier and I in a public event[24] in May 2014. This event provided us with the opportunity to assess our instruments in a public performance setting, and to test their adequacy to *incrementation* and *exploratory design* activities (Blackwell and Green, 2003) by using them to compose a four-minute song[25] we would play at this event after a short address. The lack of a polyphonic instrument compelled us to design an early version of the Pianotant, that we used to compose the song and record a backing track but did not use during the live performance. Because we were only three musicians, only three parts (one Tapant and two Soufflants with soprano saxophone and trombone sounds) of the five-voice song we composed were performed live, an electric piano, a bass guitar, and claves being recorded on the backing track. As the pianist did not have access to a laptop or mobile phone compatible with our software, we developed a web-based version of the Soufflant so that he could rehearse the performance by himself. In addition, we scheduled regular rehearsal sessions for one month before the event. We shared a musical score, but despite our familiarity with traditional staff notation we did not use it due to the short amount of training time; instead we noted pitch with the corresponding keyboard letters and symbols.

In the summer of 2014, we started to tackle the problem of switching between performance and other uses based on our previous observations in transcription and composition activities, and designed the shortcuts for sequencing control. In subsequent public performances, we mainly performed improvised music and used these shortcuts for live looping, which allowed rich musical content despite being only two performers on stage. We also participated in several public demonstrations at conferences (e.g., IHM'14, Aceituno and Potier, 2014a) and cultural events to meet other novice users than friends, co-workers, and lab visitors. One of the main impediments to the adoption of our instruments is the fact that their implementation tie them to the OS X platform, that they still require some knowledge to set up sound synthesis on first use, and that they had been works-in-progress for a long time. Thus, we had trouble finding local users who both possessed the right type of laptop and were willing to commit to integrating the instruments in their daily practice and enduring the frequent design changes and bug fixes. In the future, we would also like to focus on allowing more advanced interactions with sequencer and live-coding software during performance, and extending the macro-diversity of office music to other musical genres.

8.4.2. PHYSICAL INTERACTION WITH LAPTOPS BEYOND MUSICAL PERFORMANCE

Our positive experience with office music led us to believe that physical actions on the laptop can inspire the design of novel interaction techniques in the more traditional context of graphical user

---

[24]4th edition of the *Forum sur l'Interaction Tactile et Gestuelle*, Tourcoing, France. `http://fitg.lille.inria.fr`. Last accessed 06/16/2015.

[25]A crude recording of the song, *Fidji*, can be heard at `http://poietic.oin.name`.
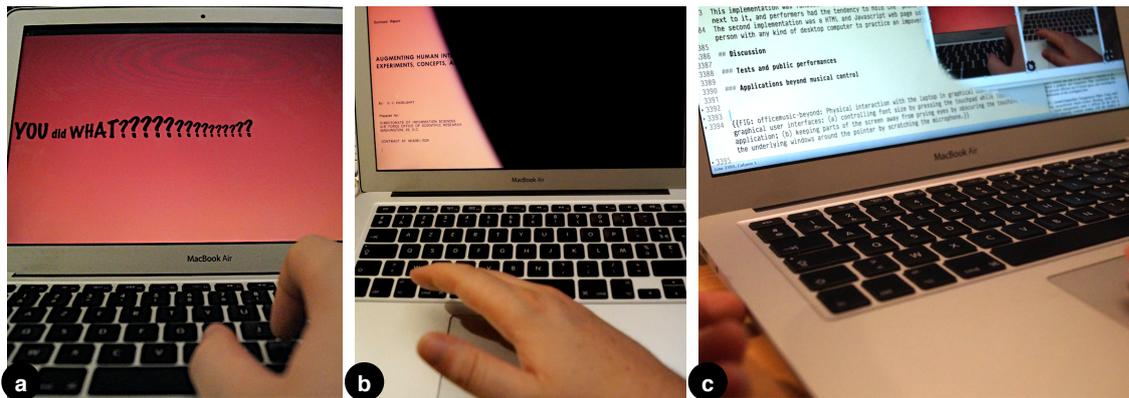
FIGURE 8.19: Physical interaction with the laptop in graphical user interfaces: (a) controlling font size by pressing the touchpad while typing in an "emotional instant messaging" application; (b) keeping parts of the screen away from prying eyes by obscuring the touchpad with the edge of the right hand; (c) revealing parts of the underlying windows around the pointer (on the right) by scratching the microphone (on the left side of the laptop).

interfaces. To illustrate this potential, we will describe three simple examples (Figure 8.19) we have implemented as proof-of-concept prototypes on OS X, based on the software we developed for our instruments.

The first example (Figure 8.19a), inspired from an application of the pressure-sensitive keyboard proposed by Dietz et al. (2009), concerns conversation in an instant messaging application. Such applications commonly provide control over the typographic attributes of a message, giving the user various ways to express an emotion or to emphasize a statement. However, as Dietz et al. (2009) remarked, users seldom have time to use these options in the middle of a fast-paced conversation. Using an interaction similar to that of the Pianotant, a user can control the font size of the characters as she types by varying the pressure of the base of the hand on the surface of the touchpad.

With the second example (Figure 8.19b), a user can immediately obscure the contents of all the connected displays by putting his open hand on the touchpad (while closing the lid of the laptop would sometimes not affect external displays). This is useful for privacy (e.g., to momentarily hide sensitive information from onlookers), but the fact that the obscured parts of the screen reflect the posture of the hand on the touchpad makes this example suitable to other uses. For example, a presenter can progressively reveal information in multiple ways depending on how the hands are positioned on the touchpad and subsequently moved away (e.g., from left to right, or by curtain opening).

The third example (Figure 8.19c) concerns the management of overlapping windows that prevent the user from reading and interacting with underlying contents. To simultaneously display two overlapping windows in the same area, windows can be displayed using blending techniques, such as alpha-blending or multiblending (Baudisch and Gutwin, 2004). To interact with the underlying contents, one can use content-aware techniques that leverage free space in the overlapping window

(Ishak and Feiner, 2004), but the content underneath occupied regions is still unaccessible. Fold-and-drop (Dragicevic, 2004) can be used to momentarily expose underlying windows, but the technique is mainly designed for drag-and-drop, and provides no option for going back to the initial state apart from completing or canceling a dragging action. Our example allows to uncover parts of the underlying windows around the pointer by scratching the microphone with the left hand (on a MacBook), creating transparent "holes" whose size is proportional to the energy expended in scratching (scratching more vigorously produces a louder sound). The holes stay in place when the user moves the overlapping window, leaving the underlying content visible. When the holes are no longer needed, the user simply taps several times on the microphone until windows return to their initial state. The current prototype only displays screenshots of each window in the user's workspace as they were arranged when the prototype was launched, thus preventing any interaction with the original windows apart from moving them, but a working version could be made using a compositing window system such as Metisse (Chapuis and Roussel, 2005). Pushing the example further, a user could thus dig through the layers of overlapping windows by scratching multiple times, revealing information relevant to her current task or interacting with the revealed contents (e.g., clicking on an important button, or dropping a file in the hole).

8.5. Conclusion

In this chapter, we presented "office music", a vision of musical control where the laptop is played somewhat similarly to traditional acoustic instruments instead of being used solely as an enabler of graphical interaction with audio software. We first examined how our approach extends the existing work in laptop-based musical instrument design. Then, we devised guidelines for the support of musical expression with high dimensionality using native laptop input, with a specific focus on the match between input devices and gestures, physical constraints and support. We also briefly discussed some tradeoffs involved in the design of computer keyboard layouts for the selection of tones. Based on this work, we proposed three office music designs: the Pianotant, a polyphonic piano-like instrument, the Soufflant, a monophonic wind instrument, and the Tapant, an unpitched percussion instrument to play modern drum parts. We also described a set of keyboard shortcuts allowing loop recording and control, as well as rapidly alternating between "performance mode" and normal interaction with graphical interfaces with less *turn-taking* than the usual switch between external physical controllers and the laptop. These instruments have been used successfully, by us and others, in various musical situations including studio recording and live performance. Finally, we discussed three example prototypes showing how physically-inspired interactions on the laptop can be applied outside a musical context.

Conclusion

To be successful as general-purpose amplifiers of thought, desktops and laptops must support direct and expressive interaction. Support for expressiveness is important because, to be useful, the system must allow the user to express various, and sometimes complex, intentions. Support for directness is also important because it determines the extent to which the user is willing to perform ambitious actions using the system, harnessing the power of externalization in exploration and creative thought. Support for both is a complex issue that requires non-trivial solutions. In this work, we argued that leveraging the unexploited capabilities of standard input devices, and the familiar actions users can perform on them, could help attain this overall objective. We investigated this approach in three contexts: increasing the granularity of basic pointing and dragging actions, increasing the extent of dragging actions, and increasing the dimensionality of musical expressions.

9.1. Summary

We first studied the notion of directness (Chapter 2) by reviewing the literature on direct manipulation and on related concepts, with a specific focus on works based on activity theory. This study clarified this loaded concept by separating its spatiotemporal, cognitive, and mediational aspects, as well as identifying properties for the support of each of them. Spatiotemporal directness relates to immediacy in space and time between action and effect, cognitive directness relates to a low amount of mental work between intention and action, and mediational directness relates to the absence of intervening operations or agency in action. These three dimensions of directness can be supported or inhibited by an interactive system in various ways.

We then moved on to the notion of expressiveness (Chapter 3). We explored its human side from a communication perspective, borrowing tools from semiotics and semiotic engineering, and discussed the notion of shared code; that is, the set of mappings between content (what is conveyed through interaction) and expression (how it is conveyed) that, when taken into account by the system, make up its expressive power. We also determined that a system's expressive power was dependent on the input devices used and the interaction languages which organize the user's actions on them, and discussed five strategies to change it. The first two are concerned with changes in established mappings of the code: rephrasing (providing new ways of expressing a content with previously unrelated expressions) and enrichment (introducing new contents in existing expressions). In contrast, the following three are less general and are specifically concerned with increasing expressive power: increasing the granularity of actions (the range of accessible levels of detail), increasing their extent (the range of extreme states that can be reached), and increasing their dimensionality (the number of dimensions of a content that an

action can control). We then identified the shortcomings of existing works following these strategies, using the previous definition of directness, and motivated our contributions in these three areas.

Concerning granularity, we focused on basic pointing and dragging actions (Part II), and questioned the ubiquitous practice of using integer positions for pointer control, which artificially constrained fine motor skills in all previous attempts to increase granularity.

We showed that, using simple pointing movements, a user was able to attain a level of detail sufficient for many tasks that currently require more indirect techniques, provided that the system supports subpixel interaction (Chapter 4); that is, the possibility of pointing "in-between pixels", using a secondary source of visual feedback. We detailed three critical factors that must be taken into account by pointing transfer functions in order to enable subpixel interaction: the characteristics of the devices, the capabilities of their users, and the density of the data model underlying the manipulated objects. In particular, we showed how the subpixel part of the transfer function could be dynamically adapted to the data model, allowing arbitrary levels of granularity without compromising normal pointer behavior, up to a certain limit.

This limit is dependent on the useful resolution (Chapter 5), the smallest displacement a user can produce using a device. Due to the lack of empirical knowledge on this aspect of fine motor skills in the literature, we proposed an operational definition of the useful resolution and an experimental protocol to measure it based on a series of increasingly small device displacements. From the results of a controlled experiment, we suggested that the useful resolution was dependent on movement direction, and that it was between 200 and 400 CPI with the 6400 CPI mouse we used. We also hypothesized the role of stick-slip phenomena in a discussion of the motor strategies employed by our participants, and highlighted the importance of transfer function adaptation and calibration to user skills in subpixel interaction.

Concerning extent, we focused on edge-scrolling (Part III), a widely implemented technique that extends dragging actions by scrolling a viewport once the pointer crosses its edge, and we provided a theoretical and empirical basis for its study.

We proposed a design space of edge-scrolling techniques (Chapter 6) with three main dimensions (scrolling area, transfer function, activation conditions) and two task-related factors (task type and edge configuration). Using a reverse-engineering method, we examined 33 existing implementations across two operating systems, and showed that existing designs occupied various positions in the design space, with few areas of conformity. We also studied the use and perception of edge-scrolling by running an interactive online survey with 214 participants, demonstrating that edge-scrolling is a widely used technique, and that perceived usability issues of control, heterogeneity, and performance, could be linked to the three dimensions of the design space, as well as to a fourth one, visual guidance.

Based on the shortcomings of existing designs, we proposed push-edge and slide-edge scrolling (Chapter 7), two edge-scrolling techniques that guarantee constant expressive power in all

viewport geometries and are based on position control. In a first controlled experiment, we showed that push-edge and slide-edge outperform the standard edge-scrolling implementation of OS X in a text selection task using a mouse and a touchpad, and suggested that push-edge scrolling is best used with the former, while slide-edge is more adapted to the latter. In a second controlled experiment using only a mouse, we compared several designs, including push-edge and existing ones, and highlighted suboptimal design features in rate control and position control edge-scrolling transfer functions.

Concerning dimensionality, we focused on the multiparametric control of sound synthesis in real time (Part IV), and demonstrated that the untapped opportunities of standard laptop input made this specific activity possible with unmodified laptops.

We proposed "office music" (Chapter 8), a vision of live musical performance where the laptop is primarily used in a way similar to traditional musical instruments. We outlined several important design considerations, including the relevance of standard inputs for sensing various musical gestures, the influence of physical constraints on the user's possible postures and interaction possibilities, and the tradeoffs involved in the design of computer keyboard layouts for musical tone selection. Based on these considerations, we detailed the design and implementation of three office music prototypes: the Pianotant, a polyphonic piano-like instrument, the Soufflant, a monophonic wind instrument controlled by the user's breath, and the Tapant, an unpitched percussion instrument inspired by modern drum kits. We discussed the progressive refinements of the prototypes as well as our successful uses of these instruments for recording, composition, and live performance, and presented examples of applications of some of the gestures investigated in other contexts than musical performance.

9.2. Progress and applicability

Our overall approach was centered around the capabilities of standard input devices and the basic actions users can perform on them, and led us to three ways of increasing the expressive power of desktops and laptops in different areas. They have been designed in order to inherently promote directness.

Subpixel interaction increases the granularity of basic pointing and dragging actions by leveraging fine motor skills. Provided adequate visual feedback, it is designed to be as spatiotemporally direct and cognitively direct as normal pointing, and does not introduce any intermediate interaction step (mediational directness). In addition, it reuses and encourages the accommodation of a well-rehearsed utilization schema—controlling a pointer using an input device—and balances the gains in expressive power according to the user's skills.

Push-edge and slide-edge scrolling increase the extent of dragging actions, and can be perceived as direct extensions of these actions. Because they rely on position control, they offer lower referential distance than rate control edge-scrolling techniques, while retaining their other benefits

in spatiotemporal and mediational directness. Moreover, the gains in expressive power are not limited by the viewport configuration anymore.

Finally, the three office music instruments increase the dimensionality of musical expressions: for example, the Tapant shows that a single finger on a touchpad can simultaneously control six parameters of a synthesized drum hit sound, compared to a maximum of three in examples from the literature (Fiebrink et al., 2007). The mappings of these instruments were designed so as to promote low referential distance, with most gestures being linked to musical output in a non-arbitrary way (e.g., blowing mapped to air pressure and overblowing, hitting mapped to drum hit force), and high skill reuse (e.g., the keyboard layout retains the structure of guitar fingering). Besides, the fact that the same physical object can be used either as a musical instrument or as a means for graphical interaction increases mediational directness in recording, composition, or creative exploration activities compared to the use of external controllers.

Nevertheless, the tradeoff between directness and expressiveness still exists. Let us put some of our contributions into perspective with the following example from @guiard-99navigation: imagine that in a virtual, "purely kinematic" version of our world, you are currently in Paris and decide to fly about 5000 km away to pick a particular flower in Central Park, New York City, by its 1 mm stem. In a multiscale map interface, by explicit control of panning and zooming, this task, although difficult, is certainly achievable: first by zooming out so as to see both sides of the Atlantic, then by repeatedly zooming in and panning around progressively smaller locations, until being able to see and select the stem of the flower. Suppose a user wants to perform a similar task without explicitly controlling scale (assuming adequate secondary feedback for the subpixel level), and only using subpixel interaction and edge-scrolling: what is a reasonable distance she would be able to fly in order to pick a flower in one pointing movement? The most extreme conditions our work tested were 6500 px scrolling (edge-scrolling experiments, Section 7.3.1, 500 lines of text), and 146 subdivisions per pixel (subpixel video example, Section 4.3.2). Assuming that each subpixel represents 1 mm, then one pixel represents 14.6 cm on the map, and a movement combining these extreme conditions only spans 949 m! Techniques more adapted to the scale of the original example are still needed to cover the 4999 initial kilometers, but if the rest of the task can be facilitated by this work, then we can consider that it has advanced our main goal.

9.3. Limitations and future work

The current work has contributed novel interactive behaviors and provided preliminary results to understand some of their aspects, but it has also left many important questions open. Some of these limitations have been evoked along this dissertation, and we will conclude by outlining a few of them, together with ideas for future iterative refinement, and a more general outlook on directness and expressiveness.

Our work mainly focused on input, and overlooked crucial aspects related to feedback and visual design. The secondary sources of visual feedback used during subpixel interaction are likely to

influence pointing and dragging performance, but how to best design and orchestrate them? Another interesting direction would be to exploit existing focus + context lens designs and to propose and experimentally evaluate implicit activation methods. In our map example (Section 4.3.5), a speed-coupled lens appeared when the user performs slow sub-pixel-speed movements, and disappeared after a larger movement. From our experience, it seems that actions remaining in the pixel territory and subpixel ones have characteristic movement profiles that could be exploited to implicitly activate a similar lens with low risks of unintended activation, following the *kinematic triggering* concept of Mott and Wobbrock (2014). Concerning edge-scrolling, we suggested that visual guidance could help solve reported control and performance issues and proposed examples to support awareness, feedback, and anticipation, but these remain to be explored and tested. Note that we also did not investigate using the screen in office music instruments, but displaying the keyboard or drum layout could facilitate learning, for example.

The findings of this thesis hold for the constrained situations we studied, but their extension to other contexts can be interesting both for future research and practical applications. Subpixel interaction and edge-scrolling on touchscreen interfaces are certainly possible, but their adaptation to collocated input would likely require a special treatment. For instance, the usual issues encountered in collocated touch input (e.g., fat fingers, visual parallax, hand occlusion, landing and take-off imprecision, Chapuis and Dragicevic, 2011) are likely to influence the design of subpixel transfer functions and applications, and the useful resolution of the finger will probably differ due to the different limb usage and movement possibilities (e.g., rolling gestures, Roudaut et al., 2009). Concerning touchscreen edge-scrolling techniques, they are more similar to those allowing inter-viewport tasks than the intra-viewport ones we studied. The necessarily small scrolling area in these techniques represents an important challenge for expressive power, and our push-edge and slide-edge designs cannot be used due to their reliance on the possibility to decouple user movements in motor space to the movement of the pointer in visual space. Concerning our office music prototypes, we designed them to control the synthesis of realistic acoustic instrument sounds, and we harnessed the referential similarity between the gestures performed on these instruments and the sound they produce. How would one control the synthesis of sounds that do not evoke physically-produced ones, such as Shepard-Risset glissandi (Arfib et al., 2002) ?

More generally, the approach followed in this thesis, as well as the study of the direct-expressive tradeoff we tried to address, remain to be systematized. Directness and expressiveness are still imprecise notions that lack empirical criteria, and while we tried to clarify some of their aspects and explicitly consider them in the designs we proposed, we believe there is opportunity for more fundamental and thorough research on these issues. Enabling more complex and ambitious actions with minimal cognitive overhead is only a small step towards fulfilling our personal computers' potential as an intellectual and creative tool.

"You're probably waiting for something impressive. What I'm trying to prime you for, though, is the realization that the impressive new tricks all are based upon lots of changes in the little things you do. This computerized system is used over and over and over again to help me do little things—where my methods and ways of handling little things are changed until, lo, they've added up and suddenly I can do impressive new things."

 — Douglas C. Engelbart, *Augmenting Human Intellect: A Conceptual Framework.*

Bibliography

Johnny Accot and Shumin Zhai. 1997. Beyond Fitts' Law: Models for Trajectory-based HCI Tasks. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '97)*. ACM, New York, NY, USA, 295–302. 71

Johnny Accot and Shumin Zhai. 2002. More than dotting the i's—foundations for crossing-based interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 73–80. 70, 173

Jonathan Aceituno and Ludovic Potier. 2014a. L'ordinateur portable comme instrument de musique. In *Adjunct proceedings of the 26th french-speaking conference on Human-Computer Interaction (IHM'14)*. ACM, 3–4. 210

Jonathan Aceituno and Ludovic Potier. 2014b. The Secret Life of Computers. In *CHI'14 Extended Abstracts on Human Factors in Computing Systems*. ACM, 151–152. 209

Alexander Travis Adams, Berto Gonzalez, and Celine Latulipe. 2014. SonicExplorer: Fluid Exploration of Audio Parameters. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 237–246. 181, 189

Christopher Ahlberg and Ben Shneiderman. 1994a. The Alphaslider: A Compact and Rapid Selector. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '94)*. ACM, New York, NY, USA, 365–371. 73

Christopher Ahlberg and Ben Shneiderman. 1994b. Visual information seeking: tight coupling of dynamic query filters with starfield displays. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 313–317. 154

Marko Aho. 2013. Gypsy Swing as a Medium for Virtuosic Gestures. *Ethnomusicology Review* 18 (2013). 185

Dzimitry Aliakseyeu, Pourang Irani, Andrés Lucero, and Sriram Subramanian. 2008. Multi-flick: An Evaluation of Flick-based Scrolling Techniques for Pen Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 1689–1698. 124, 158

Rodrigo Almeida and Pierre Cubaud. 2006. Supporting 3D Window Manipulation with a Yawing Mouse. In *Proceedings of the 4th Nordic Conference on Human-computer Interaction: Changing Roles (NordiCHI '06)*. ACM, New York, NY, USA, 477–480. 76

Tue Haste Andersen. 2005. A simple movement time model for scrolling. In *CHI'05 Extended Abstracts on Human Factors in Computing Systems*. ACM, 1180–1183. 123

Georg Apitz and François Guimbretière. 2004. CrossY: A Crossing-based Drawing Application. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM, New York, NY, USA, 3–12. 71

Caroline Appert, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2005. Context matters: Evaluating Interaction Techniques with the CIS Model. In *People and Computers XVIII — Design for Life*. Springer London, 279–295. 55, 66

Caroline Appert, Olivier Chapuis, and Emmanuel Pietriga. 2010. High-precision Magnification Lenses. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 273–282. 74, 159

Caroline Appert, Olivier Chapuis, and Emmanuel Pietriga. 2012. Dwell-and-spring: Undo for Direct Manipulation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 1957–1966. 72

Caroline Appert and Jean-Daniel Fekete. 2006. OrthoZoom Scroller: 1D Multi-scale Navigation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. ACM, New York, NY, USA, 21–30. 27, 67, 73

Daniel Arfib, Jean-Michel Couturier, Loïc Kessous, and Vincent Verfaille. 2002. Strategies of mapping between gesture data and synthesis model parameters using perceptual spaces. *Organised sound* 7, 02 (2002), 127–144. 185, 217

Kevin Wayne Arthur, Nada Matic, and Paul Ausbeck. 2008. Evaluating touch gestures for scrolling on notebook computers. In *CHI'08 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2943–2948. 124, 157

Yuji Ayatsuka, Jun Rekimoto, and Satoshi Matsuoka. 1998. Popup Vernier: A Tool for Sub-pixel-pitch Dragging with Smooth Mode Transition. In *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology (UIST '98)*. ACM, New York, NY, USA, 39–48. 73

Mathias Baglioni, Sylvain Malacria, Eric Lecolinet, and Yves Guiard. 2011. Flick-and-brake: Finger Control over Inertial/Sustained Scroll Motion. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems (CHI EA '11)*. ACM, New York, NY, USA, 2281–2286. DOI: http://dx.doi.org/10.1145/1979742.1979853 158

Gilles Bailly, Laurence Nigay, and David Auber. 2005. 2M: Un Espace De Conception Pour l'Interaction Bi-manuelle. In *Proceedings of the 2nd French-speaking Conference on Mobility and Ubiquity Computing (UbiMob '05)*. ACM, New York, NY, USA, 177–184. 55

Gilles Bailly, Thomas Pietrzak, Jonathan Deber, and Daniel J. Wigdor. 2013. Métamorphe: Augmenting Hotkey Usage with Actuated Keys. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 563–572. 71

Ravin Balakrishnan, Thomas Baudel, Gordon Kurtenbach, and George Fitzmaurice. 1997. The Rockin'Mouse: Integral 3D Manipulation on a Plane. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '97)*. ACM, New York, NY, USA, 311–318. 27, 76

Ravin Balakrishnan and Ken Hinckley. 1999. The Role of Kinesthetic Reference Frames in Two-handed Input Performance. In *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology (UIST '99)*. ACM, New York, NY, USA, 171–178. 51

Rafael Ballagas and Jan Borchers. 2006. *Selexels: A conceptual framework for pointing devices with low expressiveness*. Technical Report AIB-2006-16. RWTH, Department of Computer Science. 85, 87

Didier D. Bardon, Richard E. Berry, Shirley L. Martin, S. A. Morgan, John M. Mullay, and Craig A. Swearingen. 1997. Method for Auto-Scroll with Greater User Control. *IBM Technical Disclosure Bulletin* 40, 6 (June 1997), 181–182. 126

Jakob E. Bardram and Olav W. Bertelsen. 1995. Supporting the development of transparent interaction. In *Human-Computer Interaction*. Springer Science and Business Media, 79–90. 26, 45, 46, 52, 53, 54

Deborah Barreau and Bonnie A. Nardi. 1995. Finding and Reminding: File Organization from the Desktop. *SIGCHI Bulletin* 27, 3 (July 1995), 39–43. 62

Patrick Baudisch. 1998. Don't Click, Paint! Using Toggle Maps to Manipulate Sets of Toggle Switches. In *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology (UIST '98)*. ACM, New York, NY, USA, 65–66. 71

Patrick Baudisch and Carl Gutwin. 2004. Multiblending: Displaying Overlapping Windows Simultaneously Without the Drawbacks of Alpha Blending. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 367–374. 211

Michel Beaudouin-Lafon. 1997. Interaction instrumentale: de la manipulation directe à la réalité augmentée. *Actes des neuvièmes journées sur l'Interaction Homme-Machine* (1997), 97–104. 47

Michel Beaudouin-Lafon. 2000. Instrumental interaction: an interaction model for designing post-WIMP user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM, 446–453. 26, 47, 48, 157

Michel Beaudouin-Lafon. 2001. Novel Interaction Techniques for Overlapping Windows. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology (UIST '01)*. ACM, New York, NY, USA, 153–154. 71

Michel Beaudouin-Lafon. 2004. Designing Interaction, Not Interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '04)*. ACM, New York, NY, USA, 15–22. 27, 47, 57, 174

Michel Beaudouin-Lafon. 2005. Enjeux et perspectives en interaction homme-machine. In *Paradigmes et enjeux de l'informatique*. Hermès, 197–213. 24

Michel Beaudouin-Lafon, Stéphane Huot, Halla Olafsdottir, and Pierre Dragicevic. 2014. Glide-Cursor: Pointing with an Inertial Cursor. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces (AVI '14)*. ACM, New York, NY, USA, 49–56. 86, 161

Michel Beaudouin-Lafon and Wendy E. Mackay. 2000. Reification, polymorphism and reuse. In *Proceedings of the working conference on Advanced visual interfaces - AVI '00*. Association for Computing Machinery (ACM). 24

Benjamin B. Bederson. 2004. Interfaces for Staying in the Flow. *Ubiquity* 5, 27 (Sept. 2004). 43

Joseph D. Belfiore, Christopher J. Guzak, Christopher E. Graham, Stepehn M. Madigan, Tandy W. Trower, II, Randall L. Kerr, and Adrian M. Wyard. 1998. Auto-scrolling during a drag and drop operation. (1998). http://www.google.com/patents/US5726687 126

Ross Bencina. 2005. The metasurface: applying natural neighbour interpolation to two-to-many mapping. In *Proceedings of the 2005 conference on New interfaces for musical expression*. National University of Singapore, 101–104. 77

François Bérard. 1999. The Perceptual Window: Head Motion as a new Input Stream. In *Proceedings of INTERACT'99*. 238–244. 71

François Bérard and Amélie Rochet-Capellan. 2012. Measuring the linear and rotational user precision in touch pointing. In *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces*. ACM, 183–192. 89, 106

François Bérard, Guangyu Wang, and Jeremy R. Cooperstock. 2011. On the Limits of the Human Motor Control Precision: The Search for a Device's Human Resolution. In *Proceedings of the 13th IFIP TC 13 International Conference on Human-computer Interaction - Volume Part II (INTERACT'11)*. Springer-Verlag, Berlin, Heidelberg, 107–122. 77, 86, 89, 103, 105, 106, 190

Richard E. Berry, Stephen S. Fleming, and A. C. Temple. 1991. Auto-Scroll During Direct Manipulation. *IBM Technical Disclosure Bulletin* 33, 11 (April 1991), 312. 75, 126, 174

Olav W. Bertelsen, Morten Breinbjerg, and Søren Pold. 2007. Instrumentness for creativity mediation, materiality and metonymy. In *Proceedings of the 6th ACM SIGCHI conference on Creativity and cognition*. Association for Computing Machinery (ACM). 183

Florent Berthaut, Myriam Desainte-Catherine, and Martin Hachet. 2010. DRILE : An Immersive Environment for Hierarchical Live-Looping. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. Sydney, Australia, 192–197. 182

Florent Berthaut, Haruhiro Katayose, Hironori Wakama, Naoyuki Totani, and Yuichi Sato. 2011. First Person Shooters as Collaborative Multiprocess Instruments. In *Proceedings of the Internation Conference on New Interfaces for Musical Expression*. 44–47. 70, 76, 188

Dominik Bial, Florian Block, and Hans Gellersen. 2010. A study of two-handed scrolling and selection on standard notebook computers. In *Proceedings of the 24th BCS Interaction Specialist Group Conference*. British Computer Society, 355–364. 124, 156

Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. 1993. Toolglass and Magic Lenses: The See-through Interface. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '93)*. ACM, New York, NY, USA, 73–80. 71

Louis Bigo, Jérémie Garcia, Antoine Spicher, and Wendy Mackay. 2012. Papertonnetz: Music Composition With Interactive Paper. In *Proceedings of the Sound and Music Computing Conference*. 219–225. 199

Alan F. Blackwell and Thomas R. G. Green. 2003. *Notational Systems – the Cognitive Dimensions of Notations framework*. Morgan Kaufmann, 103–134. 209, 210

Tina Blaine and Sidney Fels. 2003. Contexts of Collaborative Musical Experiences. In *Proceedings of the 2003 Conference on New Interfaces for Musical Expression (NIME '03)*. National University of Singapore, Singapore, Singapore, 129–134. 184, 200

Renaud Blanch, Yves Guiard, and Michel Beaudouin-Lafon. 2004. Semantic Pointing: Improving Target Acquisition with Control-display Ratio Adaptation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 519–526. 117

Susanne Bødker. 1987. *Through the Interface—A Human Activity Approach to User Interface Design*. Ph.D. Dissertation. University of Aarhus, Aarhus, Denmark. 43, 44, 55

Rainer Böhme and Stefan Köpsell. 2010. Trained to Accept?: A Field Experiment on Consent Dialogs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 2403–2406. 117

Susan Bovair, David E. Kieras, and Peter G. Polson. 1990. The Acquisition and Performance of Text-editing Skill: A Cognitive Complexity Analysis. *Human-Computer Interaction* 5, 1 (March 1990), 1–48. 54

Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. 147

Susan E. Brennan. 1998. The grounding problem in conversations with and through computers. *Social and cognitive approaches to interpersonal communication* (1998), 201–225. 40

Vannevar Bush. 1945. As We May Think. *The Atlantic* (jul 1945). 24

William Buxton. 1990. A Three-state Model of Graphical Input. In *Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction (INTERACT '90)*. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 449–456. 66

William A. S. Buxton. 1983. Lexical and pragmatic considerations of input structures. *ACM SIGGRAPH Computer Graphics* 17, 1 (jan 1983), 31–37. 184

William A. S. Buxton. 1986. Chunking and Phrasing and the Design of Human-computer Dialogues. In *Proceedings of the IFIP World Computer Congress*. Dublin, Ireland, 475–480. 42, 55

William A. S. Buxton. 1987. There's more to interaction than meets the eye: Some issues in manual input. In *User Centered System Design: New Perspectives on Human-Computer Interaction*, D. A. Norman and S. W. Draper (Eds.). Lawrence Erlbaum and Associates, 319–337. 48, 125

William A. S. Buxton. 1993. HCI and the Inadequacies of Direct Manipulation Systems. *SIGCHI Bulletin* (1993). 39

William A. S. Buxton and Brad Myers. 1986. A Study in Two-handed Input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '86)*. ACM, New York, NY, USA, 321–326. 69, 125, 156, 158

Pascal Béguin and Pierre Rabardel. 2000. Designing for instrument-mediated activity. *Scandinavian Journal of Information Systems* 12, 1 (2000). 45

Claude Cadoz. 1988. Instrumental Gesture and Musical Composition. In *Proceedings of the International Computer Music Conference (ICMC 1988)*. Cologne, Germany, 12. 58, 184

Claude Cadoz. 1994. Le geste canal de communication homme/machine: la communication "instrumentale". *Technique et Science Informatiques* 13, 1 (1994), 31–61. 58

Claude Cadoz and Marcelo M. Wanderley. 2000. Gesture - Music. *Trends in gestural control of music* (2000). 184, 194, 195, 196

Stuart K. Card, William K. English, and Betty J. Burr. 1978. Evaluation of Mouse, Rate-Controlled Isometric Joystick, Step Keys, and Text Keys for text Selection on a CRT. *Ergonomics* 21, 8 (1978), 601–613. 23

Stuart K. Card, Jock D. Mackinlay, and George G. Robertson. 1991. A morphological analysis of the design space of input devices. *TOIS* 9, 2 (apr 1991), 99–122. 184

Stuart K. Card, Thomas P. Moran, and Allen Newell. 1983. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum and Associates. 66, 85

John M. Carroll and Mary Beth Rosson. 1987. Paradox of the active user. In *Interfacing thought: cognitive aspects of human-computer interaction*. MIT Press, 80–111. 46

Géry Casiez and Nicolas Roussel. 2011. No more bricolage! Methods and tools to characterize, replicate and compare pointing transfer functions. In *Proceedings of UIST '11*. ACM, 603–614. 24, 85, 88, 89, 90, 93, 99, 103, 108, 111, 115

Géry Casiez, Daniel Vogel, Ravin Balakrishnan, and Andy Cockburn. 2008. The Impact of Control-Display Gain on User Performance in Pointing Tasks. *Human–Computer Interaction* 23, 3 (2008), 215–250. 85, 98

Jared Cechanowicz and Carl Gutwin. 2009. Augmented Interactions: A Framework for Adding Expressive Power to GUI Widgets. In *Human-Computer Interaction–INTERACT 2009*. Springer, 878–891. 66, 67

Jared Cechanowicz, Pourang Irani, and Sriram Subramanian. 2007. Augmenting the Mouse with Pressure Sensitive Input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. ACM, New York, NY, USA, 1385–1394. 72

Mark Cerqueira, Spencer Salazar, and Ge Wang. 2013. SoundCraft: Transducing StarCraft 2. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. Graduate School of Culture Technology, KAIST, Daejeon, Republic of Korea, 243–247. 188

Daniel Chandler. 1994. The transmission model of communication. (1994). `http://visual-memory.co.uk/daniel/Documents/short/trans.html` 59

Dennis Chao. 2001. Doom As an Interface for Process Management. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '01)*. ACM, New York, NY, USA, 152–157. 70

Olivier Chapuis and Pierre Dragicevic. 2011. Effects of motor scale, visual scale, and quantization on small target acquisition difficulty. *ACM Transactions on Computer-Human Interaction (TOCHI)* 18, 3 (2011), 13. 86, 103, 105, 115, 217

Olivier Chapuis and Nicolas Roussel. 2005. Metisse is Not a 3D Desktop!. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST '05)*. ACM, New York, NY, USA, 13–22. 212

Olivier Chapuis and Nicolas Roussel. 2007. Copy-and-paste Between Overlapping Windows. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. ACM, New York, NY, USA, 201–210. 71

Steve Charles and Roy Williams. 1989. Measurement of hand dynamics in a microsurgery environment: preliminary data in the design of a bimanual telemicro-operation test bed. In *Proceedings of the NASA Conference on Space Telerobotics*, G. Rodriguez and H. Seraji (Eds.), Vol. 1. 109–118. 104

Fanny Chevalier, Pierre Dragicevic, and Steven Franconeri. 2014. The Not-so-Staggering Effect of Staggered Animations on Visual Tracking. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2241–2250. 24

Herbert H. Clark and Susan E. Brennan. 1991. Grounding in Communication. In *Perspectives on socially shared cognition*, Lauren B. Resnick, John M. Levine, and Stephanie D. Teasley (Eds.). American Psychological Association, Washington, 127–149. 39

Herbert H. Clark and Deanna Wilkes-Gibbs. 1986. Referring as a collaborative process. *Cognition* 22, 1 (1986), 1–39. 39

Andy Cockburn and Carl Gutwin. 2009. A predictive model of human performance with scrolling and hierarchical lists. *Human-Computer Interaction* 24, 3 (2009), 273–314. 123, 172

Andy Cockburn, Philip Quinn, Carl Gutwin, and Stephen Fitchett. 2012. Improving scrolling devices with document length dependent gain. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 267–276. 130, 157, 159, 173

Philip R. Cohen, Mary Dalrymple, Douglas B. Moran, Fernando C.N. Pereira, Joseph W. Sullivan, Robert A. Gargan Jr, Jon L. Schlossberg, and Sherman W. Tyler. 1989. Synergistic use of direct manipulation and natural language. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. Austin, Texas. 39

Nick Collins, Alex McLean, Julian Rohrhuber, and Adrian Ward. 2003. Live coding in laptop performance. *Organised Sound* 8 (12 2003), 321–330. 76, 182

Joëlle Coutaz and Laurence Nigay. 1994. Les propriétés CARE dans les interfaces multimodales. In *Actes de la conférence IHM'94*. Lille, France, 7–14. 64

Jean-Michel Couturier. 2006. A model for graphical interaction applied to gestural control of sound. In *Proceedings of the Sound and Music Computing Conference*, Vol. 6. 77, 186, 187

E. R. F. W. Crossman and P. J. Goodeve. 1983. Feedback control of hand-movement and Fitts' law. *The Quarterly Journal of Experimental Psychology A* 35 (1983). 105

Mihaly Csikszentmihalyi. 1975. *Beyond boredom and anxiety: the experience of play in work and games*. Jossey-Bass. 42

Mihaly Csikszentmihalyi. 1990. *Flow: the psychology of optimal experience*. New York: Harper. 42

Luke Dahl. 2012. Wicked Problems and Design Considerations in Composing for Laptop Orchestra. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. University of Michigan, Ann Arbor, Michigan. 198

Clarisse Sieckenius de Souza. 1993. The Semiotic Engineering of User Interface Languages. *International Journal of Man-Machine Studies* 39, 5 (Nov. 1993), 753–773. 53, 60

Clarisse Sieckenius de Souza. 2005. *The Semiotic Engineering of Human-Computer Interaction*. MIT Press. 61, 62, 65, 70

Marc Destefano, John K. Lindstedt, and Wayne D. Gray. 2011. Use of complementary actions decreases with expertise. In *Proceedings of the 33rd Annual Conference of the Cognitive Science Society. Austin, TX: Cognitive Science Society*. 2709–2014. 25

Paul H. Dietz, Benjamin Eidelson, Jonathan Westhues, and Steven Bathiche. 2009. A Practical Pressure Sensitive Computer Keyboard. In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology (UIST '09)*. ACM, New York, NY, USA, 55–58. 72, 211

Regina Dittrich, Brian Francis, Reinhold Hatzinger, and Walter Katzenbeisser. 2007. A paired comparison approach for the analysis of sets of Likert-scale responses. *Statistical Modelling* 7, 1 (2007), 3–28. 144, 145

Christopher Dobrian and Daniel Koppelman. 2006. The 'E' in NIME: Musical Expression with New Computer Interfaces. In *Proceedings of the 2006 Conference on New Interfaces for Musical Expression (NIME '06)*. IRCAM, Paris, France, 277–282. 184, 201

Sarah A. Douglas and Anant Kartik Mithal. 1994. The effect of reducing homing time on the speed of a finger-controlled isometric pointing device. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 411–416. 49, 68

Paul Dourish. 2001. *Where the Action Is: The Foundations of Embodied Interaction*. MIT Press. 46

Pierre Dragicevic. 2004. Combining Crossing-based and Paper-based Interaction Paradigms for Dragging and Dropping Between Overlapping Windows. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM, New York, NY, USA, 193–196. 27, 72, 212

USB-IF DWG. 2001. *Device Class Definition for Human Interface Devices (Version 1.11)*. Technical Report. USB Implementer's Forum. 88, 191

Umberto Eco. 1976. *A Theory of Semiotics*. Indiana University Press. 53, 59, 60, 65

Digby Elliott, Steve Hansen, Lawrence E. M. Grierson, James Lyons, Simon J. Bennett, and Spencer J. Hayes. 2010. Goal-Directed Aiming: Two Components but Multiple Processes. *Psychological Bulletin* 136, 6 (2010), 1023–1044. 105

Douglas C. Engelbart. 1962. *Augmenting Human Intellect: A Conceptual Framework*. Technical Report AFOSR-3223. Stanford Research Institute, Menlo Park, California. 24, 26

William K. English, Douglas C. Engelbart, and Melvyn L. Berman. 1967. Display-selection techniques for text manipulation. *IEEE Transactions on Human Factors in Electronics* 8, 1 (1967), 5–15. 23

Guillaume Faure, Olivier Chapuis, and Nicolas Roussel. 2009. Power Tools for Copying and Moving: Useful Stuff for Your Desktop. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1675–1678. 27, 72

Rebecca Fiebrink, Ge Wang, and Perry R. Cook. 2007. Don't Forget the Laptop : Using Native Input Capabilities for Expressive Musical Control. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. New York City, NY, United States, 164–167. 77, 181, 189, 196, 216

Stephen Fitchett and Andy Cockburn. 2010. MultiScroll: using multitouch input to disambiguate relative and absolute mobile scroll modes. In *Proceedings of the 24th BCS Interaction Specialist Group Conference*. British Computer Society, 393–402. 156, 158

Paul M. Fitts. 1954. The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology* 47, 6 (1954), 381–391. 104

George W. Fitzmaurice. 1996. *Graspable User Interfaces*. Ph.D. Dissertation. University of
Toronto. 48

George W. Fitzmaurice, Azam Khan, Robert Pieké, William A. S. Buxton, and Gordon Kurten-
bach. 2003. Tracking Menus. In *Proceedings of the 16th Annual ACM Symposium on User
Interface Software and Technology (UIST '03)*. ACM, New York, NY, USA, 71–79. 159

Clifton Forlines, Chia Shen, and Bill Buxton. 2005. Glimpse: A Novel Input Model for Multi-
level Devices. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems (CHI EA
'05)*. ACM, New York, NY, USA, 1375–1378. 72

Clifton Forlines, Daniel Wigdor, Chia Shen, and Ravin Balakrishnan. 2007. Direct-touch vs.
Mouse Input for Tabletop Displays. In *Proceedings of the SIGCHI Conference on Human Factors
in Computing Systems (CHI '07)*. ACM, New York, NY, USA, 647–656. 51

David M. Frohlich. 1993. The history and future of direct manipulation. *Behaviour and
Information Technology* 12, 6 (nov 1993), 315–329. 26, 35, 38, 39, 40, 55

David M. Frohlich. 1997. Direct manipulation and other lessons. *Handbook of human-computer
interaction* (1997), 463–488. 26, 39

George W. Furnas and Xiaolong Zhang. 1998. MuSE: A Multiscale Editor. In *Proceedings of the
11th Annual ACM Symposium on User Interface Software and Technology (UIST '98)*. ACM, New
York, NY, USA, 107–116. 75

William W. Gaver. 1991. Technology affordances. In *Proceedings of the SIGCHI conference on
Human factors in computing systems Reaching through technology - CHI '91*. Association for
Computing Machinery. 53

William W. Gaver, Gerda Smets, and Kees Overbeeke. 1995. A Virtual Window on Media Space.
In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '95)*.
ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 257–264. 71

Barney G Glaser and Anselm L Strauss. 1967. *The discovery of grounded theory: Strategies for
qualitative research*. Aldine de Gruyter, Chicago. 147

Berto Gonzalez and Celine Latulipe. 2011. BiCEP: Bimanual Color Exploration Plugin. In *CHI
'11 Extended Abstracts on Human Factors in Computing Systems (CHI EA '11)*. ACM, New York,
NY, USA, 1483–1488. 189

Michael A. Grasso, David S. Ebert, and Timothy W. Finn. 1998. The Integrality of Speech
in Multimodal Interfaces. *ACM Transactions on Computer-Human Interaction (TOCHI)* 5, 4
(1998), 303–325. 39

Wayne D. Gray and Wai-Tat Fu. 2001. Ignoring perfect knowledge in-the-world for imperfect
knowledge in-the-head. In *Proceedings of the SIGCHI conference on Human factors in computing
systems - CHI '01*. Association for Computing Machinery. 25

Benny Greb and Enno Dugnus. 2009. Hotdog. In *Brass Band [CD]*. Grundsound, Neustadt,
Germany. 209

Herbert Paul Grice. 1975. Logic and Conversation. *Syntax and Semantics* 3 (1975), 41–58. 39

Yves Guiard. 1987. Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model. *Journal of Motor Behavior* 19, 4 (1987), 486–517. 55, 197

Yves Guiard. 2009. The Problem of Consistency in the Design of Fitts' Law Experiments: Consider Either Target Distance and Width or Movement Form and Scale. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1809–1818. 104

Yves Guiard, Michel Beaudouin-Lafon, Julien Bastin, Dennis Pasveer, and Shumin Zhai. 2004. View size and pointing difficulty in multi-scale navigation. In *Proceedings of the working conference on Advanced visual interfaces*. ACM, 117–124. 122, 123, 125, 172

Yves Guiard, Michel Beaudouin-Lafon, and Denis Mottet. 1999. Navigation As Multiscale Pointing: Extending Fitts' Model to Very High Precision Tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '99)*. ACM, New York, NY, USA, 450–457. 77, 86, 104, 105

Yves Guiard, Yangzhou Du, and Olivier Chapuis. 2007. Quantifying degree of goal directedness in document navigation: Application to the evaluation of the perspective-drag technique. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 327–336. 123

Yves Guiard and Halla B Olafsdottir. 2011. On the measurement of movement difficulty in the standard approach to Fitts' law. *PloS one* 6, 10 (2011), e24389. 104

François Guimbretière and Terry Winograd. 2000. FlowMenu: Combining Command, Text, and Data Entry. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST '00)*. ACM, New York, NY, USA, 213–216. 67

Robert Hamilton. 2008. Q3OSC or: how I learned to stop worrying and love the bomb… game. In *Proceedings of the International Computer Music Conference*. 188

Robert Hamilton. 2011. UDKOSC: an immersive musical environment. In *Proceedings of the International Computer Music Conference*. 717–720. 188

H. Rex Hartson, Antonio C. Siochi, and Deborah Hix. 1990. The UAN: A User-oriented Representation for Direct Manipulation Interface Designs. *ACM Trans. Inf. Syst.* 8, 3 (July 1990), 181–203. 66

Reinhold Hatzinger and Regina Dittrich. 2012. prefmod: An R Package for Modeling Preferences Based on Paired Comparisons, Rankings, or Ratings. *Journal of Statistical Software* 48, 10 (2012), 1–31. 144

Ken Hinckley, Edward Cutrell, Steve Bathiche, and Tim Muss. 2002. Quantitative analysis of scrolling techniques. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 65–72. 122, 123, 155, 157, 158, 161, 172, 173

Ken Hinckley and Mike Sinclair. 1999. Touch-sensing input devices. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press. DOI:http://dx.doi.org/10.1145/302979.303045 124

Ken Hinckley, Mike Sinclair, Erik Hanson, Richard Szeliski, and Matt Conway. 1999. The VideoMouse: A Camera-based Multi-degree-of-freedom Input Device. In *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology (UIST '99)*. ACM, New York, NY, USA, 103–112. 76

Louis Hjelmslev. 1961. *Prolegomena to a theory of language*. The University of Wisconsin Press. 59, 60

James Hollan, Edwin Hutchins, and David Kirsh. 2000. Distributed cognition: toward a new foundation for human-computer interaction research. *TOCHI* 7, 2 (jun 2000), 174–196. 25, 62

Christian Holz and Patrick Baudisch. 2010. 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)*. ACM, New York, NY, USA, 581–590. 51

Kasper Hornbæk. 2013. Some whys and hows of experiments in Human-Computer Interaction. *Foundations and Trends in Human-Computer Interaction* 5, 4 (2013), 299–373. 44

Andy Hunt and Ross Kirk. 2000. Mapping strategies for musical performance. *Trends in Gestural Control of Music* 21 (2000), 2000. 27, 76, 183, 185, 190

Dugald Ralph Hutchings and John Stasko. 2004. Revisiting Display Space Management: Understanding Current Practice to Inform Next-generation Design. In *Proceedings of Graphics Interface 2004 (GI '04)*. Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 127–134. 96

Edwin L. Hutchins. 1987. *Metaphors for interface design*. Technical Report ICS 8703. Defense Technical Information Center. 26, 38, 39, 48, 53, 56

Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. 1985. Direct manipulation interfaces. *Hum.-Comput. Interact.* 1, 4 (Dec. 1985), 311–338. 26, 38, 52, 53, 56

Takeo Igarashi and Ken Hinckley. 2000. Speed-dependent automatic zooming for browsing large documents. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*. ACM, 139–148. 158

Don Ihde. 1979. The Experience of Technology: Human-Machine Relations. In *Technics and Praxis*. Boston Studies in the Philosophy of Science, Vol. 24. Springer Netherlands, 3–15. 54

Edward W. Ishak and Steven K. Feiner. 2004. Interacting with Hidden Content Using Content-aware Free-space Transparency. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM, New York, NY, USA, 189–192. 212

Robert K. Jacob, Audrey Girouard, Leanne M. Hirshfield, Michael S. Horn, Orit Shaer, Erin Treacy Solovey, and Jamie Zigelbaum. 2008. Reality-based interaction: a framework for post-WIMP interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 201–210. 53, 201

Robert K. Jacob, Linda E. Sibert, Daniel C. McFarlane, and M. Preston Mullen Jr. 1994. Integrality and separability of input devices. *ACM Transactions on Computer-Human Interaction (TOCHI)* 1, 1 (1994), 3–26. 48, 55, 184

Guillaume Jacquemin, Thierry Coduys, and Matthieu Ranc. 2012. Iannix 0.8. In *Actes des Journées d'Informatique Musicale*. Mons, Belgium, 107–115. 187

Herbert D. Jellinek and Stuart K. Card. 1990. Powermice and user performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 213–220. 157

Jeff A. Johnson. 1995. A comparison of user interfaces for panning on a touch-controlled display. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press/Addison-Wesley Publishing Co., 218–225. 127

Lynette A. Jones. 1997. Dextrous hands: Human, prosthetic, and robotic. *Presence: Teleoperators and Virtual Environments* 6, 1 (1997), 29–57. 104

Lynette A. Jones. 1998. Manual Dexterity. In *The Psychobiology of the Hand*, Kevin J. Connolly (Ed.). Mac Keith Press, 47–62. 104

Lynette A. Jones and Susan J. Lederman. 2006. *Human Hand Function*. Oxford University Press. 104

Sergi Jordà. 2002. FMOL: Toward User-Friendly, Sophisticated New Musical Instruments. *Computer Music Journal* 26, 3 (Sept 2002), 23–39. 187

Sergi Jordà. 2005. *Digital Lutherie: Crafting musical computers for new musics*. Ph.D. Dissertation. Universitat Pompeu Fabra, Barcelona, Spain. 77, 185, 186, 201

Patrik N. Juslin. 2003. Five Facets of Musical Expression: A Psychologist's Perspective on Music Performance. *Psychology of Music* 31, 3 (2003), 273–302. 183

Daniel Kahneman. 2011. *Thinking, Fast and Slow*. Farrar, Straus and Giroux. 183

Victor Kaptelinin. 1995. A Comparison of Four Navigation Techniques in a 2D Browsing Task. In *Conference Companion on Human Factors in Computing Systems (CHI '95)*. ACM, New York, NY, USA, 282–283. 127

Victor Kaptelinin and Bonnie A. Nardi. 2006. *Acting with Technology: Activity Theory and Interaction Design*. MIT Press. 43

Alan Kay and Adele Goldberg. 1977. Personal Dynamic Media. *Computer* 10, 3 (March 1977), 31–41. 23, 24

Alan C. Kay. 1972. A Personal Computer for Children of All Ages. In *Proceedings of the ACM Annual Conference - Volume 1 (ACM '72)*. ACM, New York, NY, USA. 27

Steven W. Keele. 1968. Movement control in skilled motor performance. *Psychological Bulletin* 70, 6, Pt. 1 (1968), 387–403. 105

Douglas Keislar. 1987. History and Principles of Microtonal Keyboards. *Computer Music Journal* 11, 1 (1987), 18–28. 199

Loïc Kessous and Daniel Arfib. 2003. Bimanuality in Alternate Musical Instruments. In *Proceedings of the 2003 Conference on New Interfaces for Musical Expression (NIME '03)*. Montreal, Quebec, Canada, 140–145. 55

David E. Kieras and Peter G. Polson. 1985. An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies* 22, 4 (1985), 365–394. 54

Won Kim, Frank Tendick, Stephen Ellis, and Lawrence Stark. 1987. A comparison of position and rate control for telemanipulations with consideration of manipulator system dynamics. *IEEE Journal of Robotics and Automation* 3, 5 (1987). 159

David Kirsh. 2013. Thinking with External Representations. In *Cognition Beyond the Brain*. Springer Science and Business Media, 171–194. 25

David Kirsh and Paul Maglio. 1994. On Distinguishing Epistemic from Pragmatic Action. *Cognitive Science* 18, 4 (1994), 513–549. 25

Shelly Knotts and Nick Collins. 2014. The Politics of Laptop Ensembles: A Survey of 160 Laptop Ensembles and their Organisational Structures. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. Goldsmiths, University of London, London, United Kingdom, 191–194. 186

Masatomo Kobayashi and Takeo Igarashi. 2006. MoreWheel: Multimode scroll-wheeling depending on the cursor location. In *Adjunct proceedings of UIST*, Vol. 6. 15–18. 158

Masatomo Kobayashi and Takeo Igarashi. 2007. Boomerang: Suspendable Drag-and-drop Interactions Based on a Throw-and-catch Metaphor. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*. ACM, New York, NY, USA, 187–190. 72

Alexander Kulik, Jan Dittrich, and Bernd Froehlich. 2012. The hold-and-move gesture for multi-touch interfaces. In *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services*. ACM, 49–58. 55, 127

Manu Kumar, Terry Winograd, and Andreas Paepcke. 2007. Gaze-enhanced scrolling techniques. In *CHI'07 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2531–2536. 71, 124

Gordon Kurtenbach and William Buxton. 1994. User Learning and Performance with Marking Menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '94)*. ACM, New York, NY, USA, 258–264. 70

Gordon Kurtenbach, George Fitzmaurice, Thomas Baudel, and Bill Buxton. 1997. The design of a GUI paradigm based on tablets, two-hands, and transparency. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*. ACM, 35–42. 71

Tellef Kvifte and Alexander R. Jensenius. 2006. Towards a Coherent Terminology and Model of Instrument Description and Design. In *Proceedings of the 2006 Conference on New Interfaces for Musical Expression (NIME '06)*. IRCAM, Paris, France, 220–225. 190

Andrew Kwatinetz. 1996. Scrolling contents of a window. (1996). `http://www.google.com/patents/US5495566` 126

Bumchul Kwon, Waqas Javed, Niklas Elmqvist, and Ji-Soo Yi. 2011. Direct Manipulation Through Surrogate Objects. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*. 627–636. 39

David M. Lane, H. Albert Napier, S. Camille Peres, and Aniko Sandor. 2005. Hidden Costs of Graphical User Interfaces: Failure to Make the Transition from Menus and Icon Toolbars to Keyboard Shortcuts. *International Journal of Human-Computer Interaction* 18, 2 (2005), 133–144. 70

Gary D. Langolf, Don B. Chaffin, and James A. Foulke. 1976. An Investigation of Fitts' Law Using a Wide Range of Movement Amplitudes. *Journal of Motor Behavior* 8, 2 (1976), 113–128. 84, 103, 104, 105, 106

Celine Latulipe. 2006. *A Symmetric Interaction Model for Bimanual Input.* Ph.D. Dissertation. University of Waterloo, Ontario, Canada. 55

Brenda K. Laurel. 1987. Interface as Mimesis. In *User Centered System Design: New Perspectives on Human-Computer Interaction*, D. A. Norman and S. W. Draper (Eds.). Lawrence Erlbaum and Associates, 67–85. 38

David Lee, KyoungHee Son, Joon Hyub Lee, and Seok-Hyung Bae. 2012. PhantomPen: Virtualization of Pen Head for Digital Drawing Free from Pen Occlusion and Visual Parallax. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 331–340. 51

Andrea Leganchuk, Shumin Zhai, and William A. S. Buxton. 1998. Manual and Cognitive Benefits of Two-handed Input: An Experimental Study. *ACM Transactions on Computer-Human Interaction* 5, 4 (Dec. 1998), 326–359. 55, 125

Andreas C. Lehmann, John A. Sloboda, and Robert H. Woody. 2007. *Psychology for Musicians: Understanding and Acquiring the Skills.* Oxford University Press. 183

Golan Levin. 2000. *Painterly Interfaces for Audiovisual Performance.* Master's thesis. Massachussetts Institute of Technology. 187

Daniel J. Levitin, Stephen McAdams, and Robert L. Adams. 2002. Control Parameters for Musical Instruments: A Foundation for New Mappings of Gesture to Sound. *Organised Sound* 7, 2 (Aug. 2002), 171–189. 193, 194

Shih-Gong Li and Theodore J. L. Shrader. 1998. Scrolling a target window during a drag and drop operation. (1998). `http://www.google.com/patents/US5740389` 126, 130, 159, 174

Joseph C. R. Licklider. 1960. Man-Computer Symbiosis. *IRE Transactions on Human Factors in Electronics* HFE-1, 1 (March 1960), 4–11. 24, 26

Wendy E. Mackay. 2002. Which Interaction Technique Works when?: Floating Palettes, Marking Menus and Toolglasses Support Different Task Strategies. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '02)*. ACM, New York, NY, USA, 203–208. 151

I. Scott MacKenzie. 1992. Fitts' Law as a Research and Design Tool in Human-Computer Interaction. *Human-Computer Interaction* 7, 1 (mar 1992), 91–139. 104

I Scott MacKenzie, Abigail Sellen, and William A. S. Buxton. 1991. A comparison of input devices in element pointing and dragging tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 161–166. 23

I. Scott MacKenzie, R. William Soukoreff, and Chris Pal. 1997. A Two-ball Mouse Affords Three Degrees of Freedom. In *CHI '97 Extended Abstracts on Human Factors in Computing Systems (CHI EA '97)*. ACM, New York, NY, USA, 303–304. 76

I. Scott MacKenzie and Colin Ware. 1993. Lag As a Determinant of Human Performance in Interactive Systems. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)*. ACM, New York, NY, USA, 488–493. 51

Jock Mackinlay, Stuart K. Card, and George G. Robertson. 1990. A semantic analysis of the design space of input devices. *Human-Computer Interaction* 5, 2 (1990), 145–190. 65, 66, 191

Thor Magnusson. 2005. Ixi Software: The Interface As Instrument. In *Proceedings of the 2005 Conference on New Interfaces for Musical Expression (NIME '05)*. National University of Singapore, Singapore, Singapore, 212–215. 188

Thor Magnusson. 2007. The ixiQuarks: merging code and GUI in one creative space. In *Proceedings of the International Computer Music Conference*. 332–339. 188

Sylvain Malacria. 2011. *Conception et évaluation de techniques d'interaction pour surfaces tactiles et papier interactif*. Ph.D. Dissertation. Télécom ParisTech, Paris, France. 27, 75

Sylvain Malacria, Gilles Bailly, Joel Harrison, Andy Cockburn, and Carl Gutwin. 2013. Promoting Hotkey Use Through Rehearsal with ExposeHK. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 573–582. 24

Sylvain Malacria, Eric Lecolinet, and Yves Guiard. 2010. Clutch-free Panning and Integrated Pan-zoom Control on Touch-sensitive Surfaces: The Cyclostar Approach. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 2615–2624. 75, 157

Joseph Malloch, David Birnbaum, Elliot Sinyor, and Marcelo M. Wanderley. 2006. Towards A New Conceptual Framework for Digital Musical Instruments. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-06) (DAFx '06)*. Montreal, Quebec, Canada, 49–52. 76, 182, 184

Gérard Marino, Marie-Hélène Serra, and Jean-Michel Raczinski. 1993. The UPIC System: Origins and Innovations. *Perspectives of New Music* 31, 1 (1993), 258–269. 187

Catherine C. Marshall and Frank M. Shipman III. 1995. Spatial Hypertext: Designing for Change. *Commun. ACM* 38, 8 (Aug. 1995), 88–97. 72

Toshiyuki Masui, Kouichi Kashiwagi, and George R. Borden, IV. 1995. Elastic Graphical Interfaces to Precise Data Manipulation. In *Conference Companion on Human Factors in Computing Systems (CHI '95)*. ACM, New York, NY, USA, 143–144. 27, 73

Steven Maupin, David Gerhard, and Brett Park. 2011. Isomorphic tessellations for musical keyboards. In *Proceedings of the Sound and Music Computing Conference*. 471–478. 199, 200

Joanna McGrenere and Wayne Ho. 2000. Affordances: Clarifying and evolving a concept. In *Graphics Interface*, Vol. 2000. 179–186. 53

Hugh McLoone, Ken Hinckley, and Edward Cutrell. 2003. Bimanual Interaction on the Microsoft Office Keyboard. In *Proceedings of INTERACT'03*. 49–56. 55, 69, 124

David McNeill. 1992. *Hand and Mind: What Gestures Reveal about Thought*. University of Chicago Press. 58

Andrew McPherson and Youngmoo Kim. 2011. Design and applications of a multi-touch musical keyboard. In *Proceedings of the Sound and Music Computing Conference*. Padova, Italy. 58

Sumit Mehra, Peter Werkhoven, and Marcel Worring. 2006. Navigating on handheld displays: Dynamic versus static peephole navigation. *ACM Transactions on Computer-Human Interaction (TOCHI)* 13, 4 (2006), 448–457. 124

John R. Meier, John W. Sullivan, and Paul Mercer. 1993. Intelligent scrolling. (1993). `http://www.google.com/patents/US5196838` 126

David E. Meyer, Richard A. Abrams, Sylvan Kornblum, and Charles E. Wright. 1988. Optimality in Human Motor Performance: Ideal Control of Rapid Aimed Movements. *Psychological Review* 95, 3 (1988), 340–370. 85, 105

David E. Meyer, J. E. Keith Smith, Sylvan Kornblum, Richard A. Abrams, and Charles E. Wright. 1990. Speed-Accuracy Tradeoffs in Aimed Movements: Toward a Theory of Rapid Voluntary Action. In *Attention and performance XIII*, Marc Jeannerod (Ed.). Lawrence Erlbaum, 173–226. 85

George A. Miller. 1956. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review* 63, 2 (1956), 81. 40

Robert B. Miller. 1968. Response time in man-computer conversational transactions. In *Proceedings of the Fall Joint Computer Conference*. ACM, 267–277. 55

Andrew J. Milne, William Sethares, and James Plamondon. 2007. Isomorphic controllers and Dynamic Tuning: invariant fingering over a tuning continuum. *Computer Music Journal* 31, 4 (2007), 15–32. 199, 200

Andrew J. Milne, Anna Xamb'o, Robin Laney, David B. Sharp, Anthony Prechtl, and Simon Holland. 2011. Hex Player — A Virtual Musical Controller. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. Oslo, Norway, 244–247. 199

MMA. 1995. *MIDI 1.0 Detailed Specification*. Technical Report. The MIDI Manufacturers Association. 200

Alan Morse and George Reynolds. 1993. Overcoming Current Growth Limits in UI Development. *Commun. ACM* 36, 4 (April 1993), 72–81. 183

Tomer Moscovich. 2009. Contact Area Interaction with Sliding Widgets. In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology (UIST '09)*. ACM, New York, NY, USA, 13–22. 70

Tomer Moscovich and John F. Hughes. 2004. Navigating Documents with the Virtual Scroll Ring. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM, New York, NY, USA, 57–60. 73, 157

Martez E. Mott and Jacob O. Wobbrock. 2014. Beating the Bubble: Using Kinematic Triggering in the Bubble Lens for Acquiring Small, Dense Targets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 733–742. 217

Alex Mulder. 1996. *Hand Gestures for HCI*. Technical Report 96-1. Simon Fraser University. 58

NAMM. 2014. The 2014 NAMM Global Report. (2014). `http://namm.org/files/ihdp-viewer/global-report-2014` 181

Mathieu Nancel. 2012. *Designing and Combining Interaction Techniques in Large Display Environments*. Ph.D. Dissertation. Université Paris-Sud, Paris, France. 85

Mathieu Nancel, Daniel Vogel, and Edward Lank. 2015. Clutching Is Not (Necessarily) the Enemy. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 4199–4202. 86, 157

Theodor H. Nelson. 1980. Interactive systems and the design of virtuality. *Creative Computing* 6, 11 (1980), 56–106. 46

Albert Ng, Julian Lepinski, Daniel Wigdor, Steven Sanders, and Paul Dietz. 2012. Designing for Low-latency Direct-touch Input. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 453–464. 51

Jakob Nielsen. 1993. Noncommand user interfaces. *Commun. ACM* 36, 4 (1993), 83–99. 55

Laurence Nigay. 1994. *Conception et modélisation logicielles des systèmes interactifs : application aux interfaces multimodales*. Ph.D. Dissertation. Université Joseph Fourier, France. 63, 64, 65

Donald A. Norman. 1984. Stages and levels in human-computer interaction. *Int. J. Man-Mach. Stud.* 21 (1984), 365–375. 37

Donald A. Norman. 1987. Cognitive Engineering. In *User Centered System Design: New Perspectives on Human-Computer Interaction*, D. A. Norman and S. W. Draper (Eds.). Lawrence Erlbaum and Associates, 31–61. 37, 64

Jan Noyes. 1983. The QWERTY keyboard: a review. *International Journal of Man-Machine Studies* 18, 3 (1983), 265 – 281. 23

Kunio Ohno, Ken-ichi Fukaya, and Jurg Nievergelt. 1985. A five-key mouse with built-in dialog control. *ACM SigChi Bulletin* 17, 1 (1985), 29–34. 76, 124

Alex Olwal, Steven Feiner, and Susanna Heyman. 2008. Rubbing and tapping for precise and rapid selection on touch-screen displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 295–304. 74

Sile M. O'Modhrain and Chris Chafe. 2000. The Performer-Instrument Interaction: A Sensory Motor Perspective. In *Proceedings of the International Computer Music Conference*. 194

Nicola Orio, Norbert Schnell, and Marcelo M Wanderley. 2001. Input devices for musical expression: borrowing tools from HCI. In *Proceedings of the 2001 conference on New interfaces for musical expression*. National University of Singapore, 1–4. 76

Brett Park and David Gerhard. 2013. Rainboard and Musix: Building dynamic isomorphic interfaces. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. Graduate School of Culture Technology, KAIST, Daejeon, Republic of Korea, 319–324. 199, 200

Mark Pedley. 2013. *Tilt Sensing Using a Three-Axis Accelerometer*. Technical Report AN3461. Freescale Semiconductor. 192

Gary Perelman, Marcos Serrano, Mathieu Raynal, Celia Picard, Mustapha Derras, and Emmanuel Dubois. 2015. The Roly-Poly Mouse: Designing a Rolling Input Device Unifying 2D and 3D Interaction. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 327–336. 27, 76

Charles Perin, Pierre Dragicevic, and Jean-Daniel Fekete. 2015. Crossets: Manipulating Multiple Sliders by Crossing. In *Proceedings of the 2015 Graphics Interface Conference (GI '15)*. Canadian Information Processing Society, Halifax, Canada. 71

Ken Perlin and David Fox. 1993. Pad: An Alternative Approach to the Computer Interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques - SIGGRAPH '93*. Association for Computing Machinery (ACM). 27

Philip Quinn, Andy Cockburn, Géry Casiez, Nicolas Roussel, and Carl Gutwin. 2012. Exposing and understanding scrolling transfer functions. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, 341–350. 124, 127, 156

Philip Quinn, Sylvain Malacria, and Andy Cockburn. 2013. Touch Scrolling Transfer Functions. In *Proc. of UIST '13*. 124, 157, 161

Pierre Rabardel. 1995. Les hommes et les technologies; approche cognitive des instruments contemporains. (1995). 45, 46

Gonzalo Ramos and Ravin Balakrishnan. 2003. Fluid Interaction Techniques for the Control and Annotation of Digital Video. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology (UIST '03)*. ACM, New York, NY, USA, 105–114. 27, 75

Gonzalo Ramos and Ravin Balakrishnan. 2005. Zliding: Fluid Zooming and Sliding for High Precision Parameter Manipulation. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST '05)*. ACM, New York, NY, USA, 143–152. 74

Gonzalo Ramos, Andy Cockburn, Ravin Balakrishnan, and Michel Beaudouin-Lafon. 2007. Pointing Lenses: Facilitating Stylus Input Through Visual-and Motor-space Magnification. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. ACM, New York, NY, USA, 757–766. 74

Jef Raskin. 2000. *The Humane Interface*. ACM Press/Addison-Welsey Publishing Co. 67, 117

Jens Rasmussen. 1983. Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models. *Systems, Man and Cybernetics, IEEE Transactions on* SMC-13, 3 (May 1983), 257–266. 41, 182

François Rastier. 2007. Communication, interprétation, transmission. *Semen, revue de sémio-linguistique des textes et discours* 23 (2007). 59

Jun Rekimoto. 1996. Tilting operations for small screen interfaces. In *Proceedings of the 9th annual ACM symposium on User interface software and technology*. ACM Press. DOI:http://dx.doi.org/10.1145/237091.237115 124

Christopher Richard and Mark Cutkosky. 2000. The effects of real and computer generated friction on human performance in a targeting task. In *Proceedings of the ASME Dynamic Systems and Control Division*, Vol. 69. 2. 105

Ljubiša Ristić. 1994. *Sensor technology and devices*. Artech House. http://books.google.fr/books?id=0AJTAAAAMAAJ 106

Curtis Roads and Max Mathews. 1980. Interview with Max Mathews. *Computer Music Journal* 4, 4 (1980), 15–22. 186

Anne Roudaut, Eric Lecolinet, and Yves Guiard. 2009. MicroRolls: Expanding Touch-screen Input Vocabulary by Distinguishing Rolls vs. Slides of the Thumb. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 927–936. 217

Joseph Butch Rovan, Marcelo M Wanderley, Shlomo Dubnov, and Philippe Depalle. 1997. Instrumental gestural mapping strategies as expressivity determinants in computer music performance. In *Kansei, The Technology of Emotion. Proceedings of the AIMI International Workshop*. Citeseer, 68–73. 185, 194

Joseph D. Rutledge and Ted Selker. 1990. Force-to-motion functions for pointing. In *Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction*. North-Holland Publishing Co., 701–706. 68

Mark S. Sanders and Ernest J. McCormick. 1992. *Human Factors in Engineering and Design* (seventh ed.). McGraw-Hill, Inc. 38, 48

Joey Scarr, Andy Cockburn, Carl Gutwin, and Philip Quinn. 2011. Dips and Ceilings: Understanding and Supporting Transitions to Expertise in User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 2741–2750. 70

Dominik Schmidt, Florian Block, and Hans Gellersen. 2009. A Comparison of Direct and Indirect Multi-touch Input for Large Surfaces. In *Human-Computer Interaction – INTERACT 2009*. Lecture Notes in Computer Science, Vol. 5726. Springer Berlin Heidelberg, 582–594. 51

Andrew Sears and Ben Shneiderman. 1991. High Precision Touchscreens: Design Strategies and Comparisons with a Mouse. *Int. J. Man-Mach. Stud.* 34, 4 (April 1991), 593–613. 51

Zack Settel and Cort Lippe. 2003. Convolution Brother's Instrument Design. In *Proceedings of the 2003 Conference on New Interfaces for Musical Expression (NIME '03)*. National University of Singapore, Singapore, Singapore, 197–200. 184

Claude E. Shannon. 1948. A Mathematical Theory of Communication. *The Bell System Technical Journal* 27 (oct 1948), 379–423. 59

Kang Shi, Sriram Subramanian, and Pourang Irani. 2009. PressureMove: Pressure Input with Mouse Movement. In *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part II (INTERACT '09)*. Springer-Verlag, Berlin, Heidelberg, 25–39. 76

Ben Shneiderman. 1983. Direct Manipulation: A Step Beyond Programming Languages. *Computer* 16, 8 (aug 1983), 57–69. 26, 35, 36, 40, 42

Ben Shneiderman. 1997. Direct manipulation for comprehensible, predictable and controllable user interfaces. In *Proceedings of the 2nd international conference on Intelligent user interfaces*. ACM, 33–39. 35

Ben Shneiderman and Richard Mayer. 1979. Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International Journal of Computer and Information Sciences* 8, 3 (1979), 219–238. 36

Garth Shoemaker and Carl Gutwin. 2007. Supporting multi-point interaction in visual workspaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 999–1008. 27, 96

Amit Singh. 2006. *Mac OS X Internals: A Systems Approach*. Addison-Wesley Professional. 192

Scott Smallwood, Dan Trueman, Perry R. Cook, and Ge Wang. 2008. Composing for Laptop Orchestra. *Computer Music Journal* 32, 1 (mar 2008), 9–25. 186

David Canfield Smith. 1977. *Pygmalion: a computer program to model and stimulate creative thought*. Vol. 40. Birkhauser. 40, 52

G. M. Smith and m. c. schraefel. 2004. The Radial Scroll Tool: Scrolling Support for Stylus- or Touch-based Document Navigation. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM, New York, NY, USA, 53–56. 73, 157

Janet L. Starkes, Irene Payk, Peter Jennen, and David Leclair. 1993. A Stitch in Time: Cognitive Issues in Microsurgery. In *Cognitive Issues in Motor Expertise*. Janet L. Starkes and Fran Allard, 225–239. 84, 104, 116

Pierre Steiner. 2010. Philosophie, technologie et cognition: état des lieux et perspectives. *Intellectica* 53, 54 (2010), 7–40. 25

Stanley S. Stevens. 1946. On the Theory of Scales of Measurement. *Science* 103, 2684 (1946), 677–680. 190

Ivan E. Sutherland. 1963. Sketchpad: A Man-machine Graphical Communication System. In *Proceedings of the May 21-23, 1963, Spring Joint Computer Conference (AFIPS '63 (Spring))*. ACM, New York, NY, USA, 329–346. DOI:http://dx.doi.org/10.1145/1461551.1461591 122

Synaptics. 2001. *Synaptics TouchPad Interfacing Guide*. Technical Report 510-000080 - A. Synaptics Incorporated. 191

Synaptics. 2011. *Synaptics PS/2 TouchPad Interfacing Guide*. Technical Report 511-000275-01 Rev. B. Synaptics Incorporated. 191

Miriam Taverniers. 2008. Hjelmslev's semiotic model of language: An exegesis. *Semiotica* 171 (2008), 367–294. 59

Robert Tubb and Simon Dixon. 2014. A Four Strategy Model of Creative Parameter Space Interaction. In *Proceedings of the Fifth International Conference on Computational Creativity (ICCC 2014)*. Ljubljana, Slovenia. 183

Herre van Oostendorp and Benjamin J. Walbeehm. 1995. Towards modelling exploratory learning in the context of direct manipulation interfaces. *Interacting with computers* 7, 1 (1995), 3–24. 35

Dan Venolia. 1993. Facile 3D Direct Manipulation. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)*. ACM, New York, NY, USA, 31–36. 76

Jo Vermeulen, Kris Luyten, Elise van den Hoven, and Karin Coninx. 2013. Crossing the Bridge over Norman's Gulf of Execution: Revealing Feedforward's True Identity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 1931–1940. 53, 153

Roel Vertegaal, Tamas Ungvary, and Michael Kieslinger. 1996. Towards a musician's cockpit: Transducers, feedback and musical function. In *Proceedings of the International Computer Music Conference*. 308–311. 184

Kim J. Vicente and Jens Rasmussen. 1990. The ecology of human-machine systems II: Mediating direct perception in complex work domains. *Ecological Psychology* 2, 3 (1990), 207–249. 41, 46

Kim J. Vicente and Jens Rasmussen. 1992. Ecological interface design: Theoretical foundations. *Systems, Man and Cybernetics, IEEE Transactions on* 22, 4 (1992), 589–606. 41

Daniel Vogel and Ravin Balakrishnan. 2010. Occlusion-aware Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 263–272. 51

Daniel Wagner and Dieter Schmalstieg. 2007. ARToolKitPlus for Pose Tracking on Mobile Devices. In *Proceedings of the Computer Vision Winter Workshop*. St Lambrecht, Austria, 139–146. 192

Julie Wagner, Stéphane Huot, and Wendy Mackay. 2012. BiTouch and BiPad: Designing Bimanual Interaction for Hand-held Tablets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 2317–2326. 197, 198

Marcelo M. Wanderley. 1999. Non-obvious Performer Gestures in Instrumental Music. In *Gesture-Based Communication in Human-Computer Interaction*. Lecture Notes in Computer Science, Vol. 1739. Springer Berlin Heidelberg, 37–48. 184, 194, 195

Marcelo M. Wanderley and Philippe Depalle. 2004. Gestural Control of Sound Synthesis. *Proc. IEEE* 92, 4 (apr 2004), 632–644. 185

Marcelo M. Wanderley, Bradley W. Vines, Cory McKay, and Welsey Hatch. 2005. The Musical Significance of Clarinetists' Ancillary Gestures: An Exploration of the Field. *Journal of New Music Research* 34, 1 (2005), 97–113. 184

Marcelo M. Wanderley, Jean-Philippe Viollet, Fabrice Isart, and Xavier Rodet. 2000. On the choice of transducer technologies for specific musical functions. In *Proceedings of the 2000 International Computer Music Conference (ICMC '00)*. 244–247. 185

Nayuko Watanabe, Motoi Washida, and Takeo Igarashi. 2007. Bubble Clusters: An Interface for Manipulating Spatial Aggregation of Graphical Objects. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*. ACM, New York, NY, USA, 173–182. 72

A.T. Welford, A.H. Norris, and N.W. Shock. 1969. Speed and accuracy of movement and their changes with age. *Acta Psychologica* 30 (1969). 105

David Wessel and Matthew Wright. 2002. Problems and Prospects for Intimate Musical Control of Computers. *Computer Music Journal* 26, 3 (sep 2002), 11–22. 76, 185, 190

Geraint A. Wiggins. 2006. A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems* 19, 7 (2006), 449 – 458. 183

Terry Winograd and Fernando Flores. 1986. *Understanding computers and cognition: A new foundation for design*. Intellect Books. 44, 46

Jacob O. Wobbrock, Leah Findlater, Darren Gergle, and James J. Higgins. 2011a. The aligned rank transform for nonparametric factorial analyses using only anova procedures. In *Proceedings of CHI'11*. ACM, 143–146. http://doi.acm.org/10.1145/1978942.1978963 110

Jacob O. Wobbrock, Shaun K. Kane, Krzysztof Z. Gajos, Susumu Harada, and Jon Froehlich. 2011b. Ability-based design: Concept, principles and examples. *ACM Transactions on Accessible Computing (TACCESS)* 3, 3 (2011), 9. 54

Shota Yamanaka and Homei Miyashita. 2013. The Nudging Technique: Input Method Without Fine-grained Pointing by Pushing a Segment. In *Proceedings of the Adjunct Publication of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13 Adjunct)*. ACM, New York, NY, USA, 3–4. 71

Anna Zacchi and Frank Shipman. 2007. Personal Environment Management. In *Research and Advanced Technology for Digital Libraries*. Lecture Notes in Computer Science, Vol. 4675. Springer Berlin Heidelberg, 345–356. 62

Mark Zadel and Gary Scavone. 2006. Different Strokes: A Prototype Software System for Laptop Performance and Improvisation. In *Proceedings of the 2006 Conference on New Interfaces for Musical Expression (NIME '06)*. IRCAM, Paris, France, France, 168–171. 187

Robert Zeleznik, Timothy Miller, and Andrew Forsberg. 2001. Pop Through Mouse Button Interactions. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology (UIST '01)*. ACM, New York, NY, USA, 195–196. 72

Shumin Zhai. 1995. *Human performance in six degree of freedom input control*. Ph.D. Dissertation. University of Toronto. 158

Shumin Zhai, Barton A Smith, and Ted Selker. 1997a. Dual stream input for pointing and scrolling. In *CHI'97 Extended Abstracts on Human Factors in Computing Systems*. ACM, 305–306. 124, 125, 157

Shumin Zhai, Barton A Smith, and Ted Selker. 1997b. Improving Browsing Performance: A study of four input devices for scrolling and pointing tasks, paper. In *Proceedings of INTERACT97: The Sixth IFIP Conference on Human-Computer Interaction, Sydney, Australia, Jul*. 14–18. 69, 78, 124, 125, 157, 158

Haimo Zhang and Yang Li. 2014. GestKeyboard: Enabling Gesture-based Interaction on Ordinary Physical Keyboard. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 1675–1684. 70, 191

Jian Zhao, R. William Soukoreff, Xiangshi Ren, and Ravin Balakrishnan. 2014. A model of scrolling on touch-sensitive displays. *International Journal of Human-Computer Studies* 72, 12 (2014), 805 – 821. 123, 157, 172