Direct and Indirect Multi-Touch Interaction on a Wall Display

Jérémie Gilliot, Géry Casiez, Nicolas Roussel

Direct and Indirect Multi-Touch Interaction on a Wall Display

Jérémie Gilliot[1], **Géry Casiez**[2] **& Nicolas Roussel**[1]
[1]Inria Lille, [2]Université Lille 1, France
{jeremie.gilliot, nicolas.roussel}@inria.fr, gery.casiez@lifl.fr

ABSTRACT
Multi-touch wall displays allow to take advantage of co-located interaction (direct interaction) on very large surfaces. However interacting with content beyond arms' reach requires body movements, introducing fatigue and impacting performance. Interacting with distant content using a pointer can alleviate these problems but introduces legibility issues and loses the benefits of multi-touch interaction. We introduce WallPad, a widget designed to quickly access remote content on wall displays while addressing legibility issues and supporting direct multi-touch interaction. After briefly describing how we supported multi-touch interaction on a wall display, we present the WallPad widget and explain how it supports direct, indirect and de-localized direct interaction.

Key Words
Multi-touch; large display; direct interaction; indirect interaction; de-localized interaction.

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation (e.g. HCI): User interfaces.

INTRODUCTION

Multi-touch wall displays make it possible for one or more people to interact with computing systems on an unprecedented scale. Co-localized touch-based interaction with the displayed content contributes to a high feeling of directness, but interaction with remote content can quickly become tiring and inefficient due to the required body movements. Objects of interest might be hard to perceive and reach if at the opposite end of the display, for example. Moving closer might help perceive them, but the objects might still remain out of arm's reach. Other users might also be standing on the way, or one might not want to move for some reason. These problems are even more acute when using legacy applications originally designed for smaller screen sizes: desktop applications heavily rely on widgets that can not be moved easily (e.g. menus, toolbars and scrollbars) and that are automatically placed in locations far from ideal on large displays, for example.

Specific techniques are direly needed to properly support multi-touch interactions with legacy and modern applications on large displays. Previous work has investigated the combination of direct and indirect interaction [7], techniques to bring remote objects within arm's reach [2, 3] and interaction at a distance through an on-screen portal [10] or a mobile device [5]. However each of these solutions provides only partial answers to the above issues and multi-touch interaction at a distance in a multi-user context introduces additional challenges. In this paper, we present WallPad, a widget designed to address all of the above issues and the limitations of the current state of the art. WallPad supports elegant creation through a simple gesture, easy access to remote content, and precise direct, indirect and de-localized multi-touch interaction with it.

After a description of the context and our original motivation for this work, we present WallPad and its different features before presenting and discussing previous works related to multi-touch interaction on large displays.

CONTEXT AND MOTIVATION

Most Virtual Reality rooms built in the 1990s and 2000s were based on a large stereoscopic display. They typically used a 3D tracking system to support interaction with the virtual world, and a mouse and a keyboard to interact with the operating system and auxiliary 2D applications. At the time of their design, these VR rooms were clearly technologically advanced and considerable amounts of money were spent to realize them. But as interactive surfaces have become more and more popular in other contexts, the lack of support of their wall display for touch-based interaction certainly contributes to a diminished interest in them today. Our motivation for this work was the desire to upgrade an existing VR room to support multi-touch interactions.

Despite the many hardware and software technologies available for multi-touch interaction, supporting it in a VR room actually remains quite a challenge. The room configuration is often the result of different trade-offs and so cannot necessarily be easily changed. The screen is seldom flat and the image often produced by a carefully calibrated multi-projector system, for example. Ideally, one would like the hardware and software additions for multi-touch sensing to be cheap, easy to set up and with

http://dx.doi.org/10.1145/2670444.2670445

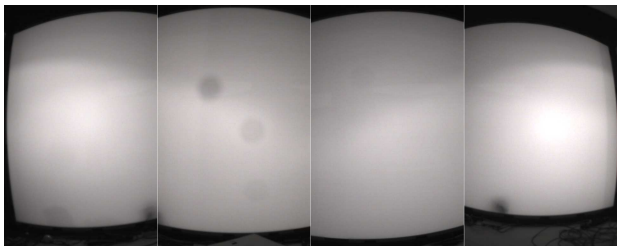

Figure 1. Raw images from our four cameras. The dark spots are caused by camera lens or filter stains.

a minimal impact on the exiting uses of the room for VR applications, including the use of legacy 2D applications.

Initial room configuration

Inaugurated in 2006, the VR room in our research facility included about 400 k€ worth of equipment including a large display wall. The screen of this display is made of a 5.96 m × 2.43 m × 7 mm acrylic layer constrained by a frame to follow a curved cylinder shape (76 cm sagitta) and coated on the front with a diffuser. Two specifically calibrated stereo projectors placed behind it make it possible to seamlessly display a composite image of 2392×1050 pixels.

To support multi-touch on such a large surface, optical sensing seemed the most affordable and practical solution. As the room had been literally built around the screen, changing the acrylic layer for an Endlighten[1] one or taking apart its frame to put LEDs on its side was not an option. Diffused surface illumination (DSI) and frustrated total internal reflection (FTIR) were thus quickly ruled out. The curved shape of the screen also ruled out laser and LED light planes (LLP and LED-LP). Rear diffuse illumination (DI) had been successfully used by the University of Groningen for their *Reality touchscreen*[2], with 1000 LEDs. We tried a similar approach by placing two illuminators taken from an Immersion iliGHT table behind our screen, but this did not work well. We had great difficulties achieving homogeneous rear IR illumination, the shininess of the acrylic layer causing numerous reflections and saturated areas.

During our tests, however, we observed that the room's lighting system emitted enough light in the IR spectrum to provide homogeneous front illumination. We thus decided to take advantage of this, acquired the appropriate hardware and started developing the necessary software to detect and analyze the IR shadows on the screen.

Hardware additions

The room's lighting system consists of 15 lamps attached to the ceiling plus 4 omni-directional ones on the walls and 4 others on the ground. All lamps are regular halogen bulbs which we set to their maximum intensity for maximum IR illumination.

[1] `http://openmaterials.org/materials-101-light-diffusing-acrylic/`

[2] `http://www.rug.nl/science-and-society/centre-for-information-technology/hpcv/nieuws/touchscreen1`

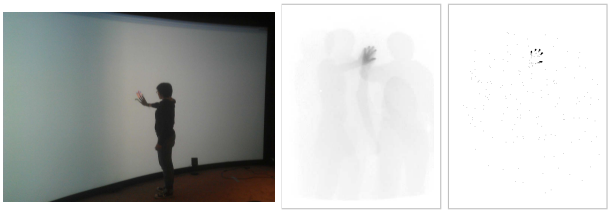

Figure 2. Background subtraction (middle) and adaptive thresholding (right) when touching the screen (left).

We use four IDS UI-1220LE cameras equipped with Tamron 13VM2811ASIR lenses and visible light filters (750 nm high-pass filters) to capture the entire screen[3]. The cameras are positioned in portrait mode behind the screen, outside of the projection volume. The overlapped region between two cameras is about 30 cm large. The four cameras are connected to a PC (2.66 GHz Intel Xeon, 4 GB RAM, Windows 7) running a custom software that processes the four 752×480 images at 60 Hz and sends contact information to the machine in charge of the display over UDP.

Software additions

Starting from raw images like those of Figure 1, our software detects projected shadows and analyzes them to generate touch events. Like in a stadium where players have several shadows in different directions, the plurality of IR sources allows to discriminate contact points (where shadows converge) from the penumbra created by objects and people further away from the screen.

Our software uses the IDS SDK to adjust the settings (e.g. the exposure time) and retrieve images from the four cameras. The images are then processed using OpenCV, the cameras having been calibrated using a classical black-white chessboard. Each image is first corrected for barrel distortion. A background image taken while the room was empty is subtracted from it, which removes everything but the shadows (Figure 2, middle image). A Gaussian adaptive thresholding function with a 13×13 neighborhood is used to preserve only the darkest areas (Figure 2, right). Blobs are then extracted from these areas, filtered using size (minimum and maximum) and aspect-ratio thresholds, and geometrically corrected for on-screen projection. The blobs found in the four images are finally merged based on a distance threshold, labeled, and serialized as a TUIO bundle sent over the network.

The geometric correction for projection compensates the strong distortion caused by the curved shape of the screen and the positioning of the cameras near the floor. The curved screen is approximated by 10×6 flat rectangles, a classical homography being applied on each one to define the projection from the camera image to the screen.

Our system can detect 50+ simultaneous contacts with a precision between 3 and 5 mm (at the bottom and top of the screen) and an average overall latency of 140 ms (including 30 ms for image processing). With the configured thresholds, fingers are detected from 1 cm away from the

[3] The total cost for each camera is about 340 €

Figure 3. WallPad in a drawing application. The widget comprises a rectangular touch sensitive area (a) with a button below (b). Finger drags in the touch sensitive control the relative position of the pointer (c). The region around the pointer is displayed in the touch sensitive area. Each WallPad button and pointer have a unique color to help disambiguating them.

screen, which is hardly perceivable when touching an object but can be annoying when tapping as it requires larger finger movements. The calibrations of the cameras and projection are good enough so that users do not notice any offset between their finger(s) and the detected contact(s). The overall result is actually surprisingly good considering the small additions we made to the room equipment. Yet multi-touch sensing is not enough. Specific interaction techniques are also required to support usable multi-touch interactions with legacy (often 2D) and custom applications.

WALLPAD

WallPad[4] is a new widget we designed to ease multi-touch interaction on wall displays. It supports the creation of proxy views in an easy and elegant way to bring distant interface regions within arm's reach. It is intended to help improving distant objects legibility through these views, ease switching between direct multi-touch interaction within them and indirect cursor-based interaction through them, and support fast and precise remote cursor manipulation through non-linear transfer functions. In the following, we describe the WallPad widget from its use as a regular touchpad to more sophisticated interactions, detailing its creation, moving, resizing and dismissal.

Regular Touchpad Use

To ease learning for novice users, WallPad was designed to operate by default as a conventional touchpad. Its visual representation is similar to one, comprising a rectangular touch sensitive area with a button below (Figure 3). As WallPad appears on top of existing applications, its representation is semi-transparent to reduce occlusion and visual clutter. Any finger drag in the rectangular area allows to relatively position a distant arrow-shaped pointer through a non-linear transfer function designed to easily cover large distances while preserving precise positioning (pointer movements are scaled down when movements are slow and scaled up when they are fast). We used a transfer

[4]the source code and video are available at http://ns.inria.fr/mint/WallPad

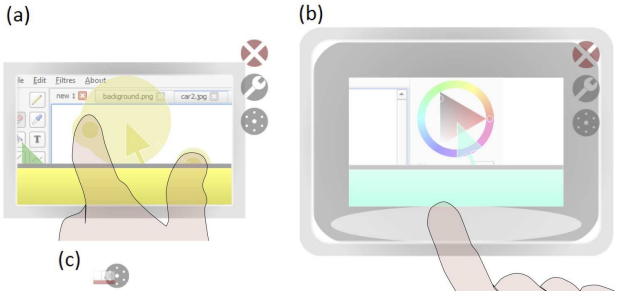

Figure 4. (a) WallPad screen capture in the de-localized direct interaction mode: WallPad is opaque to the background. (b) WallPad extended border after pressing the wrench button: the widget can be moved and resized. (c) Minimized view after pressing the minimized button.

function similar to the one used by OS X for real touchpads [4]. To reduce clutching after a drag has been initiated, finger tracking is not limited to the WallPad widget area during this operation. Taps on the button produce click events. As distant objects can be hard to perceive, the rectangular area displays a 1:1 view of the region underneath the pointer (clipped to the widget area, see Figure 3a) which allows to precisely position it, to select small objects or read small texts for example.

De-localized Direct Interaction

WallPad also allows to directly interact with the objects presented in the local rectangular view of the distant area (*de-localized direct interaction*). Single and double taps on the objects displayed in the rectangular region are passed to their remote location. Thus, users can directly select menu items or icons shown in the proxy view instead of having to first bring the pointer on the objects of interest and second pressing the button to select them (Figure 3a). Items are selected on touch release events to prevent selecting objects when the rectangular area is used as a regular touchpad.

Multi-touch object manipulation is enabled after holding a finger steady in the widget for 0.5s or touching it with several fingers. The view is turned from semi-transparent to opaque when multi-touch interaction is engaged to provide a visual feedback of the quasi-mode change and suggest objects can be further manipulated (Figure 4a). Contact areas are displayed as little colored disks around the pointer at the remote location to provide awareness to other users. WallPad remains in this mode for 0.5 s after the last contact is lost to support clutching while dragging objects.

WallPad LIFE CYCLE

The soft-aspect [11] of WallPad allows to create as many devices as needed, with different sizes and positions.

Creation gesture

WallPads can be dynamically created, moved, resized and dismissed. Creating one requires the use of an unambiguous gesture to prevent unexpected apparition. After considering different alternatives, we came up with the idea of bringing out a pointer from a finger. The corresponding gesture consists in first touching the display with a finger and, while keeping this first finger steady on the surface,

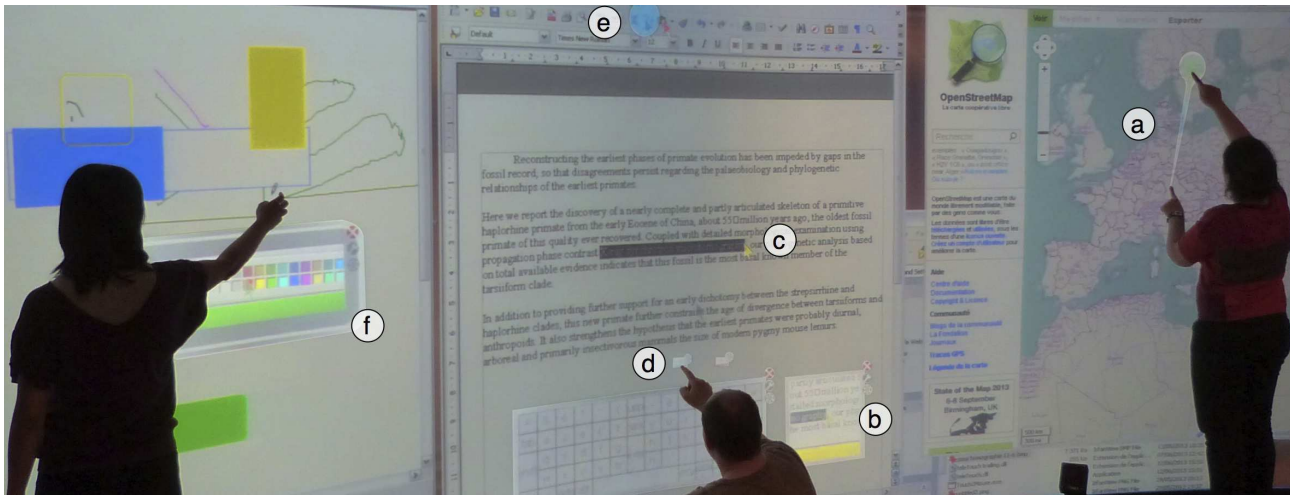

Figure 5. A user creating a WallPad (a). Another using two WallPads to select (b) some text (c) and click (d) on Copy/Paste buttons (e) beyond reach. A third user accessing a color palette located at the bottom of the screen through a WallPad (f).

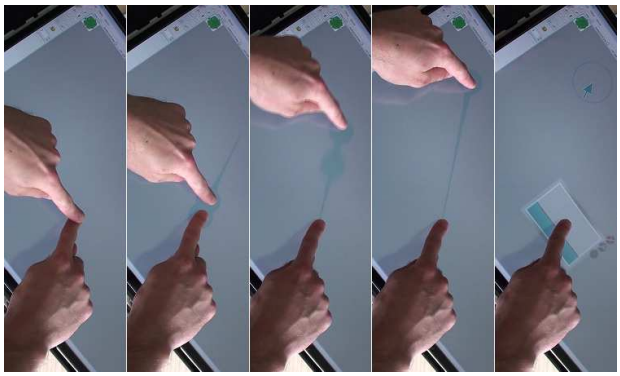

Figure 6. Creating a WallPad (shown here on a 3M multi-touch screen for clarity). From left to right: the user first slides a finger along a finger that remains steady on the surface and continues sliding on the display up to a distance threshold after which the WallPad is created. Feedback and feed-forward help knowing the gesture has been correctly recognized and predict when the WallPad will be created and where the pointer will appear.

using a finger from the other hand to slide on the first one up to the display and continue sliding for a given distance threshold (Figure 6). Once the sliding finger touches the display, specific feedback (a line and a disk) is used to indicate the gesture has been successfully recognized (Figure 5a). It is then used as feed-forward to show where the pointer will be created (Figure 6). The creation can be canceled by either removing the first finger or by not crossing the distance threshold with the second finger.

During our tests, we never faced ambiguous detection with other gestures such as pinch or rotate. This type of gesture can be easily extended to create other widgets. A virtual keyboard could be created by sliding two fingers instead of one, for example. The WallPad creation gesture could have interfered with the two finger scale gesture, but this one is usually performed by moving the fingers in opposite directions while our gesture expects the first finger to remain steady. The chosen gesture specifies both the desired widget location and the initial location of the cursor. It does not allow to create cursors in out-of-reach regions, but the relative non-linear mapping allows to quickly move newly created cursors to these regions. A unique color is associated to each WallPad (button and pointer) to help disambiguate them.

Dynamic and reconfigurable aspects

A WallPad can be moved or resized after pressing the wrench icon to highlight a border around it (Figure 4b). Touching the border with one finger allows to move the WallPad while using two fingers allows to resize it. Resizing is especially useful when one wants the WallPad to fit the shape of a remote toolbar, for example (Figure 5f). It is also useful for adjusting the size of the working area when using the de-localized direct interaction (Figure 5b). The ability to create multiple local proxies for distant rectangular regions is somewhat similar to what *WinCuts* [17] and the *User Interface Façades* [16] support on desktop systems.

WallPads can be minimized by pressing a specific icon located below the wrench one (Figure 4c). Minimized WallPad appear as a small square buttons with a 20 mm side (Figure 5d). Once minimized, the corresponding pointer can no longer be moved (a drag gesture on a minimized widget moves it, not the corresponding pointer). However single and double tap events are still forwarded to it. Minimized WallPad are useful to reduce visual clutter and occasionally press a button which is far away such as undo/redo or copy/paste buttons (Figure 5d,e). Finally, a WallPad can be dismissed by hitting the cross button (Figure 4c).

Remote WallPad pointers are treated as ordinary graphical objects and can be directly manipulated. This is particularly useful when a pointer gets in the way of a remote user, by occluding some information for example. That user could simply move the pointer aside. This feature can also ease collaboration, a remote user having the possibility to position another user's pointer where he wants her to do something.

RELATED WORK AND DISCUSSION

Multi-touch walls afford co-localized interaction with screen content (*direct interaction*), a convenient way to

interact with objects within arm's reach. Yet interacting with farther objects quickly becomes tiring and inefficient because of the required body movements. To alleviate this problem a common solution is to interact indirectly with these remote objects using a pointer controlled with a relative mapping (*indirect interaction*). For example *HybridPointing* [7] was designed to switch from direct to indirect pen interaction on large displays by catching and releasing a trailing widget. Alternatively, objects can be thrown to distant locations but it can make precise positioning difficult [9]. The *SurfaceMouse* [1] emulates a desktop mouse on multi-touch surfaces to control a single cursor with a relative mapping. The Surface mouse is activated when the wrist and five fingers are in contact with the surface. Tangible tools have also been used to switch between direct and indirect interaction but they are not very practical with vertical surfaces [19]. However even if these techniques allow to switch between direct and indirect interaction using different methods, they only support single point interaction.The *Rizzo* virtual mouse allows to control a virtual pointer using a relative mapping while a magnified view of the remote objects is displayed near fingertips [18]. This improves the problem of legibility with distant objects but Rizzo was not designed to be used together with co-located interaction on the objects of interest nor does it support the manipulation of remote objects with multi-touch gestures.

Instead of controlling a cursor to interact with remote objects, *Drag&Pop* [2] and the *Vacuum* widget [3] allow to temporarily bring distant targets within arm's reach to interact with them. However, these techniques require knowledge about the objects of interest which makes them unsuitable for interaction in empty spaces, when drawing for example. Instead of providing access to selected remote objects, *WIM* [15] allows to interact with any distant object of a 3D scene by presenting a miniature view of the whole scene. In practice, interaction with small objects is rather difficult, especially in the case of large scenes of course. Back in the 2D world, rather than showing a miniature view of the whole workspace, *Frisbee* [10] brings a delimitated (round) part of it within arm's reach. Users can directly manipulate remote objects through this duplicated view, without the need for a remote pointer. However the absence of a pointer at the remote location can actually be a problem in multi-user situations, if someone else is working in the same area. *Tablecloth* [13], which temporarily moves desktop portions, is also badly suited for multi-user situations as it would alter everyone's view of the display. Personal devices are sometimes used to support multi-user interaction with remote objects [5, 14, 12]. Yet interaction with a large display on a small separate input surface poses a lot of other questions [8].

Like the closest technique in the literature, Frisbee, WallPad allows to create, reposition and resize a local view on a remote space and and to refine one's focus in that space. The main difference between the two lies in the interaction with distant objects. With Frisbee, since interaction can only take place in the local view, one has to make sure that all the objects and areas of interest for the task are locally visible beforehand. WallPad supports this mode of operation, but also pointer-based interaction in the remote space. This in turn supports situated, unprepared interactions, and not just planned ones. The display of a pointer for action at the remote location instead of a disk shape corresponding to the local view's focus also has a profound impact on multi-user interaction. It reduces visual clutter and allows users to know what others are doing, and not just where they are generally looking. The use of multiple Frisbees is mentioned in [10], but only as a design enhancement "to support large-scale workspace interactions". Other advantages of WallPad over Frisbee include support for direct multi-touch interaction with remote objects, pointer-based indirect interaction using a non-linear mapping to reduce clutching (a 1:1 mapping is used when moving a Frisbee's focus in the remote space) and a rectangular shape better suited to common regions of interest (e.g. widgets and windows).

Using WallPad, users can dynamically create cursors whenever deemed necessary, each of them being independently controlled. The widgets are persistent - until users dismiss them - and are freely movable. WallPads can be used to repeatedly access remote content (e.g. widgets), to drag objects over large distances, or to interact with out-of-reach ones. By switching between different WallPads, users can quickly change the locus of interaction. Beyond the speed and accuracy advantages of using a pointer controlled through a non-linear transfert function, WallPad makes it possible to instantaneously move to different exact locations to support recurrent interactions, in a way similar to UIMarks [6].

CONCLUSION

WallPad was motivated by the lack of interaction techniques to properly support direct and indirect interaction on multi-touch wall displays. Throughout its design, it has been refined several times through informal user testing to obtain a tool easy to use in varied situations. The current version requires little learning. Once users know how to create a WallPad and how to transition to the de-localized direct interaction mode, the other operations are pretty straight-forward. WallPad is intended to reduce user fatigue when interacting with remote content and doing so without sacrificing precision or legibility. We expect WallPad to be a simple and powerful tool allowing users who master it to interact at a distance with applications that were not necessarily designed for large multi-touch displays.

ACKNOWLEDGEMENTS

This work was supported by the ANR project ANR-09-CORD-013 InSTInCT - `http://anr-instinct.cap-sciences.net` - and the Conseil Régional Nord - Pas de Calais.

REFERENCES

1. Bartindale, T., Harrison, C., Olivier, P., and Hudson, S. E. SurfaceMouse: supplementing multi-touch interaction with a virtual mouse. In *Proceedings of TEI '11*, ACM (2011), 293–296.

2. Baudisch, P., Cutrell, E., Robbins, D., Czerwinski, M., Tandler, P., Bederson, B., and Zierlinger, A.

Drag-and-pop and drag-and-pick: Techniques for accessing remote screen content on touch- and pen-operated systems. In *Proceedings of INTERACT '03* (2003), 57–64.

3. Bezerianos, A., and Balakrishnan, R. The vacuum: facilitating the manipulation of distant objects. In *Proceedings of CHI '05*, ACM (2005), 361–370.

4. Casiez, G., and Roussel, N. No more bricolage! methods and tools to characterize, replicate and compare pointing transfer functions. In *Proceedings of UIST '11*, ACM (2011), 603–614.

5. Chapuis, O., Bezerianos, A., and Frantzeskakis, S. Smarties: An input system for wall display development. In *Proceedings of CHI '14*, ACM (2014), 2763–2772.

6. Chapuis, O., and Roussel, N. UIMarks: quick graphical interaction with specific targets. In *Proceedings of UIST '10*, ACM (2010), 173–182.

7. Forlines, C., Vogel, D., and Balakrishnan, R. Hybridpointing: fluid switching between absolute and relative pointing with a direct input device. In *Proceedings of UIST '06*, ACM (2006), 211–220.

8. Gilliot, J., Casiez, G., and Roussel, N. Impact of form factors and input conditions on absolute indirect-touch pointing tasks. In *Proceedings of CHI '14*, ACM (2014), 723–732.

9. Hascoët, M. Throwing models for large displays. In *Human Computer Interaction, HCI'2003*, B. H. Group, Ed. (Bath, UK, 2003), 77–108.

10. Khan, A., Fitzmaurice, G., Almeida, D., Burtnyk, N., and Kurtenbach, G. A remote control interface for large displays. In *Proceedings of UIST '04*, ACM (2004), 127–136.

11. Nakatani, L. H., and Rohrlich, J. A. Soft machines: a philosophy of user-computer interface design. In *Proceedings of CHI '83*, ACM (1983), 19–23.

12. Nancel, M., Chapuis, O., Pietriga, E., Yang, X.-D., Irani, P. P., and Beaudouin-Lafon, M. High-precision pointing on large wall displays using small handheld devices. In *Proceedings of CHI '13*, ACM (2013), 831–840.

13. Robertson, G., Czerwinski, M., Baudisch, P., Meyers, B., Robbins, D., Smith, G., and Tan, D. The large-display user experience. *IEEE Computer Graphics and Applications 25*, 4 (July 2005), 44–51.

14. Satyanarayan, A., Weibel, N., and Hollan, J. Using overlays to support collaborative interaction with display walls. In *Proceedings of IUI '12*, ACM (2012), 105–108.

15. Stoakley, R., Conway, M. J., and Pausch, R. Virtual reality on a wim: interactive worlds in miniature. In *Proceedings of CHI '95*, ACM (1995), 265–272.

16. Stuerzlinger, W., Chapuis, O., Phillips, D., and Roussel, N. User interface facades: towards fully adaptable user interfaces. In *Proceedings of UIST '06*, ACM (2006), 309–318.

17. Tan, D. S., Meyers, B., and Czerwinski, M. Wincuts: manipulating arbitrary window regions for more effective use of screen space. In *CHI '04 Extended Abstracts*, ACM (2004), 1525–1528.

18. Vlaming, L., Collins, C., Hancock, M., Nacenta, M., Isenberg, T., and Carpendale, S. Integrating 2D mouse emulation with 3D manipulation for visualizations on a multi-touch table. In *Proceedings of ITS '10*, ACM (2010), 221–230.

19. Vogel, D., and Casiez, G. Conté: multimodal input inspired by an artist's crayon. In *Proceedings of UIST '11*, ACM (2011), 357–366.