On-World Computing

Enabling Interaction on Everyday Surfaces

CMU-HCII-18-101
August 2018

Robert Xiao

Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

brx@cs.cmu.edu
https://www.robertxiao.ca

Thesis Committee:
Scott E. Hudson (Co-Chair)
Chris Harrison (Co-Chair)
Jodi Forlizzi
Hrvoje Benko, Oculus Research

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Abstract

Computers are now ubiquitous. However, computers and digital content have remained largely separate from the physical world – users explicitly interact with computers through small screens and input devices, and the "virtual world" of digital content has had very little overlap with the practical, physical world. My thesis work is concerned with helping computing escape the confines of screens and devices, to spill digital content out into the physical world around us. In this way, I aim to help bridge the gap between the information-rich digital world and the familiar environment of the physical world and allow users to interact with digital content as they would ordinary physical content. I approach this problem from many facets: from the low-level work of providing high-fidelity touch interaction on everyday surfaces, easily transforming these surfaces into enormous touchscreens; to the high-level questions surrounding the interaction design between physical and virtual realms. To achieve this end, building on my prior work, I develop two physical embodiments of this new mixed-reality design: a tiny, miniaturized projector and camera system providing the hardware basis for a projected on-world interface, and a head-mounted augmented-reality head-mounted display modified to support touch interaction on arbitrary surfaces.

Dedication

To my parents, for their support and guidance, and to Aiyang, for her inspiration and love.

Acknowledgements

I would not be here without the many excellent mentors, collaborators, friends and family throughout my graduate career, who have helped shape me into the person I am.

Foremost, I want to deeply thank both of my advisors. Chris Harrison is not only an academic advisor but a dear friend, and I have had the utmost privilege of working alongside him for the past seven years. Through him, I have learned a great deal about life, work, and academia, and I have been deeply grateful for his enthusiasm, support and guidance throughout my graduate life. Scott Hudson is a continual source of wisdom and advice, and he has always given me the freedom to explore my own directions while inspiring me to do my best.

Hrvoje Benko, my unofficial third advisor and committee member, deserves special mention for his patience and mentorship through my multiple stints at Microsoft, and for introducing me to so many new and interesting research ideas and problems. Julia Schwarz has also had a profound effect on my research path, through our many shared adventures from CMU to Qeexo to Microsoft, and I am grateful to her for all of those opportunities. I would also like to thank all of my collaborators at Qeexo – Sang Won Lee, Leandro Zungri, Greg Lew, Jim Baur and Taihei Munemoto, to name a few, for the unforgettable startup experience, their teamwork and professionality.

To my fellow students, Gierad Laput, Yang Zhang, Karan Ahuja and Yasha Iravantchi – big thanks for being there with me in the lab at all hours, and for all the crazy research experiences we've had. I am also indebted to the many co-authors and colleagues from my graduate career, including John Antanitis, Jeffrey Bigham, Ishan Chatterjee, Anthony Chen, Jodi Forlizzi, Anhong Guo, Aaron Hoff, Walter Lasecki, Ivan Poupyrev, Asta Roseway, Alanson Sample, Mike Sinclair, John Tang, Gina Venolia, Karl D.D. Willis, Jason Wiese, and Andy Wilson.

My educational experience has been shaped in innumerable ways by the educators I met along the path. As my undergraduate research advisors, Carl Gutwin and Regan Mandryk irrevocably instilled a passion for human-computer interaction research in me and guided me towards my current research path. I must also thank Oliver Schneider, without whom I might never have

ended up in HCI in the first place. I am also deeply grateful to the many teachers and educators from my childhood, especially Janet Christ, Ms. Kargut and Karen Kowalenko-Evjen, who have each transformed the way I look at the world.

I am forever grateful to my parents, Wei and Chun, and sister Cora, who have always supported and loved me through everything, and from whom I learned every life lesson I know. I have been blessed in life to have been raised in such good company, and for all of the opportunities that they gave to me. I must also extend great thanks to my friends from the very grand old country of Saskatoon for always being there for me.

Finally, Aiyang, the love of my life, is the rock of my life who keeps me grounded. She complements me in every way, and I cherish the fact that we are always learning from each other. She continually challenges me to be the very best person I can be, and for that I will always be grateful.

Table of Contents

List of Figures

List of Tables

CHAPTER 1. INTRODUCTION

Computing is finally ubiquitous, after decades of work in computer science and engineering. Today, this ubiquity comes in the form of highly sophisticated mobile computing devices like smartphones and laptops. However, interactions with these devices are confined to their small screens, limiting the size and sophistication of possible interactions. More critically, the computational power is strictly limited to the digital realm, and cannot be applied to the world even immediately around them.

In contrast, the physical environment offers an expansive canvas for interactions, enabling highly expressive, comfortable, contextual and natural means to interact with content and other people. Augmenting the physical environment with computational capabilities offers a wide range of possibilities. The goal, then, is to allow computing to escape the narrow confines of the present-day small screens and devices, and spill interaction out onto the surfaces and spaces around us, so that it may augment our everyday activities.

One approach to achieve this goal is to replace every surface in the environment with an interactive computer screen. Although this would certainly bring ubiquitous computing into the environment, it would also be prohibitively expensive to install and intrusive, both socially and aesthetically.

In this thesis, I describe my efforts to enable "world-scale" computation by projecting interactive content onto everyday surfaces in the environment, allowing users to interact with relevant, contextual information situated directly in their world, without the need to physically replace parts of the environment. I will focus on two primary challenges that need to be addressed to achieve this vision: 1) *input sensing* and 2) *interaction techniques for on-world interfaces*.

1.1 Input Sensing

First, the world-scale computer must be able to sense user input. Classically, special-purpose controller devices such as mice and keyboards sensed user input using dedicated circuitry. However, these devices can be inappropriate for ad-hoc on-world input: when any surface can be interactive, it is not ideal to ask the user to find their physical on-world keyboard whenever they need to enter text. More recently, the advent of *natural user interaction* has promoted the use of hand gestures, body movements and speech as controller-free input techniques. Using a fixed sensor observing the user, these input techniques allow a user to provide input without needing to hold or use a separate device. However, these techniques suffer from a lack of haptic (physical) feedback, as well as limited precision and high fatigue, all of which preclude their prolonged use for on-world interaction.

Touch interfaces have become ubiquitous for small screens due to the popularity of touchscreen-based smartphones and tablets, and because touch is a natural, expressive modality for computer input. Most touch interfaces use specially-designed panels to sense the electrical or physical characteristics of a touch contact, but augmenting surfaces with touch panels remains expensive and can be intrusive to install in some environments.

The introduction of digital projectors and low-cost depth camera technologies raises the possibility of transforming these everyday surfaces into large, touch-sensitive computing experiences. While free-space hand and finger tracking research spans several decades of research, starting with seminal work by Krueger in the Video Place System [Krueger 1985], comparatively little research has examined finger and touch tracking on ordinary, unmodified surfaces. This can be attributed to the difficult challenge of first segmenting a finger from the background to extract its spatial position and then undertaking the even more challenging task of sensing when a finger has physically contacted a surface (vs. merely hovering close to it).

The advent of inexpensive depth cameras offered a promising potential solution for addressing this challenge. Early work by Wilson et al. [Wilson 2010a] demonstrated the potential of this approach for detecting touches on arbitrary surfaces. While prior approaches have demonstrated

that depth-based touch tracking should be viable, full exploration of the design space requires input sensing that exhibits high stability and positional accuracy, as well as reliable touch segmentation/detection with both low false positives and low false negatives.

Unfortunately, attaining this very high accuracy strains the capabilities of even the latest generation of depth cameras. The depth resolution and noise characteristics of current generation sensors means that fingertips simply merge into the surface at distances needed to cover reasonably sized work surfaces, making precise touch tracking extremely difficult. This has made it challenging to move beyond the first proof-of-concept research systems to practical use in real deployments. In Chapter 4, I describe a first proof-of-concept system which implements depth-based touch tracking, while in Chapter 6, I refine this earlier system into a practical, accurate touch tracking system for on-world computation.

Providing solid, accurate touch tracking on everyday surfaces makes it possible to transform any surface into a large touchscreen, marking the first step towards a true on-world computational system. Now, the second major challenge is for users to interact with computational content on these surfaces, and for the system to interact with physical objects in the environment.

1.2 Interaction Techniques for On-World Interfaces

Interaction with on-world content is markedly different from interaction with a typical desktop computer. One difference is that there can be many different surfaces in the environment, each of which might host interactive content. The system must be responsible for sensing these surfaces and determining which surfaces may be suitable for content (e.g. large enough, flat enough, oriented correctly). An appropriate surface then needs to be selected for an interactive element (e.g. application interface), either manually by the user using a summoning, launching or instantiation interaction technique, or automatically by the system using a layout algorithm which may account for prior positioning, user preference, surface characteristics, interface size and shape needs, and so on. In Chapter 4, I explore manual selection and instantiation techniques in the

context of a projected on-world interactive system, while in Chapter 5 I explore optimization-based techniques for automatically positioning interfaces in the environment.

Next, the system must respond to and coexist with artifacts in the physical environment. Instead of the clean, perfect world of the rectangular display, an on-world computational system must contend with a wide variety of surfaces and objects, and with surfaces that are often cluttered and messy. This is often overlooked in other work – in the future interfaces envisioned in many prior systems, desks, tables and work spaces are clear of physical artifacts like keyboards, mice, mugs, papers, knickknacks, and other contemporary and commonplace items. Furthermore, these spaces are constantly in flux, with items moving, stacking, appearing and disappearing as users interact with them.

Omitting these items often simplifies the implementation, but makes them less practically applicable to real-world situations. Failing to account for these objects makes applications and systems brittle – for example, a mug placed atop a virtual, projected interface might inject spurious touch input, or a book placed over an interface might occlude and disable the system. Further, because physical objects cannot move or change size on their own, the burden of responsiveness falls to the digital elements. Thus, digital applications must employ a variety of strategies to successfully cohabit a work surface with physical artifacts. In Chapter 5, I contemplate strategies that systems can use to responsively handle changes in the environment, including techniques for coexisting with physical desk objects.

1.3 Document Structure

In the following chapters, I will introduce my research into solving each of these areas, showing that it is indeed possible to develop a system that supports touch interaction with projected content on surfaces in the environment. Chapter 2 provides an overview of the literature and prior work domains that intersect with my thesis work. Chapter 3 charts my initial explorations into on-world interfaces, starting with my explorations into environment sensing and interaction (3.1), and large-display interaction (3.2), and finishing with more relevant explorations into on-world

projection (3.3) and touch sensing (3.4). Chapter 4 describes my first complete on-world interactive system, WorldKit, designed to explore the design and implementation of projected sensors and interfaces. WorldKit examines both input sensing and interaction techniques, and serves as a starting point for further exploration. In Chapter 5, I explore on-world interaction techniques in greater depth, seeking to build a system that could naturally interact with and respond to the physical environment in a more nuanced way. Chapter 6 addresses the input sensing problem with the DIRECT touch tracking system, which provides touch tracking on everyday surfaces that is on par with physical touchscreens.

Subsequently, I turn my attention to building two physical embodiments of my work. The first embodiment, detailed in Chapter 7, is a miniaturized projection and depth sensing system compact enough to be housed in a smartwatch form factor. This tiny system is capable of projecting a fully realized touch interface onto a wearer's arm, extending the smartwatch's interface across the skin surface. This system represents the smallest known projection and touch sensing system, providing the hardware basis for ubiquitous on-world projected interfaces. In Chapter 8, I detail the second embodiment, which is based on an augmented-reality headset in which physical surfaces are virtually augmented in the head-mounted display. Users, wearing a head-mounted display, see virtual content overlaid on physical surfaces, and can reach out and interact with the content using the DIRECT input sensing pipeline. This project explores touch sensing from the perspective of a dynamic, moving head-mounted display, and interaction techniques that cross the boundary between on-surface and in-air interactions, and between 2D and 3D content. Finally, in Chapter 9, I conclude by summarizing key contributions and defining future avenues of exploration into this space.

CHAPTER 2. BACKGROUND

My thesis work intersects with many diverse areas of human-computer interaction research, which I now review. First, I will review work related to my interaction style and approach, specifically prior on-world computing literature and work on augmented desktops. Next, I cover work that intersects with my technical approach, specifically covering projection mapping and on-world touch contact tracking.

2.1 Computing on the World

The notion of having computing everywhere has been a grand challenge for the HCI community for decades [Underkoffler 1999, Weiser 1999]. Today, users have achieved ubiquitous computing not through ubiquity of computing infrastructure, but rather by carrying sophisticated mobile devices everywhere we go (e.g., laptops, smartphones) [Harrison 2010a]. This strategy is both cost effective and guarantees a minimum level of computing quality. However, by virtue of being portable, these devices are also small, which immediately precludes a wide range of applications.

In contrast, the environment around us is expansive, allowing for large and comfortable interactions. Moreover, applications can be multifaceted, supporting complex tasks, and allowing for multiple users. And perhaps most importantly, the environment is already present – we do not need to carry it around. These significant benefits have spawned numerous research systems for interacting beyond the confines of a device and on the actual surfaces of the world.

A broad range of technical approaches has been considered for appropriating the environment for interactive use, including acoustic [Harrison 2008] and electromagnetic sensing [Cohn 2011]. A popular alternative has been camera/projector systems. By using light, both for input (sensing) and output (projection), system components can be placed out of the way, yet provide distributed functionality. This is both minimally invasive and potentially reduces the cost of installation (i.e. not requiring substantial wiring for sensors).

2.1.1 Projected interfaces in and on the World

Seminal work on such systems was initiated in the late 1990s. An early project, The Intelligent Room [Brooks 1997], eloquently described their objective as: “Rather than pull people into the virtual world of the computer, we are trying to pull the computer out into the real world of people.” The system used cameras to track users, from which a room’s geometry can be estimated. A pair of projectors allows one wall to be illuminated with interactive applications. Additional cameras were installed on this wall at oblique angles, allowing for finger touches to be digitized.

Of note, The Intelligent Room required all interactive surfaces be pre-selected and calibrated. The goal of the Office of the Future [Raskar 1998] was to enable users to designate any surface as a “spatially immersive display.” This was achieved by capturing the 3D geometry of surfaces through structured light. With this data, interfaces could be rectified appropriately, and potentially updated if the environment was dynamic (as could [Jones 2010]). The authors also experimented with head tracking (via a separate magnetically-driven system) to provide interfaces that appeared correct from the user’s viewpoint (*i.e.* egocentrically correct), even when projecting on irregular surfaces. Although calibration to a surface would be automatic, the work does not describe any user mechanisms for defining surfaces. Once applications were running on surfaces, the system relied on conventional means of input (e.g., keyboard and mouse).

The Luminous Room [Underkoffler 1999] was another early exploration of projector/camera-driven interaction. It was unique in that it enabled simple input on everyday projected surfaces. Through computer vision, objects could be recognized through fiducial markers. It also suggested that the silhouette of hands could be extracted, and incorporated into interactive applications. The system used a single conventional camera, so presumably hover/occlusion could not be easily disambiguated from touch. Like The Intelligent Room, this system also required pre-calibration and configuration of surfaces before they could be used.

The Everywhere Displays project [Pinhanez 2001] used a steerable mirror in conjunction with a projector to output dynamic graphics on a variety of office surfaces. To correct for distortion, a camera was used in concert with a known projected pattern. Using this method, a 3D scene could

be constructed for desired projection surfaces. The authors speculate that touch sensing could be added by stereo camera sensing or examining shadows.

More recently, Bonfire [Kane 2009] – a laptop mounted camera/projection system – enabled interactive areas on either side of the laptop. Because the geometry of the setup is known a priori, the system can be calibrated once, allowing graphics to be rendered without distortion despite oblique projection. Touch interaction was achieved by segmenting the fingers based on color information, and performing a contour analysis. The desk-bound nature of laptops means Bonfire also intersects with "smart desk" systems (see *e.g.,* [Koike 2001, Wellner 1993]), which tend to be static, controlled infrastructure (*i.e.,* a special desk). Although both setups provide opportunities for interactive customization, the context is significantly different.

The advent of low-cost depth sensing led to a resurgence of interactive environment projects. A single depth camera can view a large area and be used to detect touch events on everyday surfaces [Wilson 2007, Wilson 2010a]. LightSpace [Wilson 2010b] used an array of calibrated depth cameras and projectors to create a live 3D model of the environment. This can be used to track users and enable interaction with objects and menus in 3D space. LightSpace can also create virtual orthographic cameras in a specified volume, which can be used for example, to create thin planar volumes that can be used as multitouch sensors. A similar project called OASIS [Intel 2012] describes similar capabilities and goals, although many details have not been made public.

OmniTouch [Harrison 2011] is a worn depth camera and projection system that enables multitouch finger interaction on ad hoc surfaces, including fixed infrastructure (e.g., walls), handheld objects (e.g., books), and even users' bodies. Applicable surfaces within the system's field of view (~2m) are tracked and an approximate real-world size is calculated, allowing interfaces to be automatically scaled to fit. Orientation is estimated by calculating an object's average surface normal and second moment, allowing for rectified graphics. Users can "click and drag" on surfaces with a finger, which sets an interface's location and physical dimensions – a very simple example of user-defined interfaces, as we discuss in the next section. Finally, [Jones 2010] allowed for interactive projections onto physical setups constructed from passive blocks; user interaction is achieved with an IR stylus.

2.1.2 User-Defined Interfaces

An important commonality of the aforementioned systems is a lack of end-user mechanisms for defining interactive areas or functionality. There is, however, a large literature regarding user defined interactions, going back as far as command line interfaces [Good 1984] and extending to the present, with *e.g.*, unistroke characters [Wobbrock 2005] and touch screen gestures [Nielsen 2004, Wobbrock 2009]. More closely related to our work is user-driven interface layout and composition. For example, end-users can author interfaces by sketching widgets and applications [Landay 2001], which is far more approachable than traditional GUI design tools. Research has also examined run-time generation of user interfaces based on available I/O devices, the task at hand, and user preferences and skill [Gajos 2010].

When moving out into the physical world, easily defining interfaces is only half the problem. Equally challenging is providing easy-to-use end-user tools that allow instrumentation of the physical world. Sensors, microprocessors and similar require a degree of skill (and patience) beyond that of the typical user. Hardware toolkits [Arduino, Greenberg 2001, Hartmann 2006, Lee 2004] were born out of the desire to lower the barrier to entry for building sensor driven applications. There have also been efforts to enable end users to easily create custom, physical, interactive objects with off the shelf materials, for example, Styrofoam and cardboard [Akaoka 2010, Avrahami 2002, Hudson 2006].

Most closely related to our technical approach are virtual sensing techniques – specifically, approaches that can sense the environment, but need not instrument it. This largely implies the use of cameras (though not exclusively; see *e.g.*, [Cohn 2011, Harrison 2008]). By remote sensing on *e.g.*, a video feed, these systems can sidestep many of the complexities of building interfaces physically onto the environment.

One such project is Eyepatch [Maynes-Aminzade 2007], which provides a suite of computer vision based sensors, including the ability to recognize and track objects, as well as respond to user gestures. These complex events can be streamed to user-written applications, greatly simplifying

development. Slit-Tear Visualizations [Tang 2008], although not used as inputs per se, are conceptually related. The interface allows users to draw sensing regions onto a video steam, which, through a simple visualization, allow users to readily distinguish environmental events, such as cars passing. Similarly, Light Widgets [Fails 2002] allows users to select regions on everyday surfaces using a live video feed for sensing. Regions can instantiate one of three different widget types: buttons, linear sliders and radial dials. Sensing is achieved with a pair of cameras set apart (to disambiguate occlusion from touch on surfaces), with user fingers detected by finding skin-colored blobs.

2.2 Augmented Desktops

Much of the work relating to on-world interaction can be found in the augmented desktop literature. The concept of an augmented desktop – either projected [Wellner 1993] or through AR/VR technologies [Mulder 2003] – goes back to at least the 1970's with Knowlton's optically superimposed button array [Knowlton 1977]. However, the full vision of the augmented desk did not appear until the early 90s, with seminal systems such as Xerox PARC's digital desk [Newman 1992, Wellner 1993] and Ishii's TeamWorkStation [Ishii 1990]. The concept was rapidly expanded upon in the 90's with key systems including interactiveDESK [Arai 1995], EnhancedDesk [Koike 2001], I/O Bulb [Underkoffler 1999], Illuminated Light [Underkoffler 1998], Office of the Future [Raskar 1998] and the Everywhere Displays Projector [Pinhanez 2001].

Recent advances in electronics and networking have created opportunities to refine the augmented desk experience. Magic Desk [Bi 2011] prototyped an augmented desk interface using a physical touchscreen as the desk surface. IllumiShare [Junuzovic 2012] offers a sophisticated, end-to-end desktop remote collaboration experience, while Bonfire [Kane 2009] takes the concept mobile with cameras and projectors operating behind the lid of a laptop. MirageTable [Benko 2012] merges physical and virtual objects on a tabletop, with a simple physics-based interaction approach. Newer projects, such as LuminAR [Linder 2010] and AR Lamp [Kim 2014], put forward light-bulb-like implementations in attempts to achieve the technical vision proposed in the I/O Bulb [Underkoffler 1999].

Systems that superimpose rectified information onto physical artifacts (*e.g.*, [Newman 1992, Underkoffler 1998, Wellner 1993]) might be described as using "following" or "snapping". However, there is an important difference between projected content that merely tracks with physical objects, and an interface that attaches to an object 3D geometry and follows its movements. The closest related work is the "binding" behavior described in Live Paper [Robertson 1999]. Other efforts, such as WorldKit [Xiao 2013], aim to bootstrap on-world application development by offering an SDK to abstract away many of the complexities of operating on everyday surfaces (e.g., touch tracking, rectified projected output). There have also been recent design-oriented efforts to study e.g., augmented desk usage in the wild [Hardy 2012] as well as superior desk form factors [Wimmer 2010].

Finally, ObjecTop [Khalilbeigi 2013] proposes a set of interactive behaviours that can be used to interact with occluded interfaces underneath physical objects, including highlighting, repositioning and grouping occluded objects. Although ObjecTop used an optical multitouch table and fiducially-tracked planar objects, the interactions are still applicable to projected interactions. In this work, we describe the technical approach needed for virtual-physical cohabitation in a true 3D setting without any instrumentation of the objects or surfaces, and further extend ObjecTop with interactions for summoning interactive applications and automatically evading physical obstacles.

Behaviors surrounding physical desks and workplaces have long intrigued researchers from fields including management sciences, cultural anthropology and ergonomics (e.g., [Malone 1983, Sellen 2003, Vyas 2012]). More recently, HCI researchers have studied desk practice to better understand how to support and integrate digital workflows (e.g., [Bondarenko 2005, Gebhardt 2014, Hardy 2012, Malone 1983, Steimle 2010]). While this prior work provides great insight into the culture of desk practice, it tends to overlook the minutiae of small-scale, desk-level interactions.

2.3 On-World Display

To provide a complete interactive experience, some form of display is needed to overlay virtual content onto ordinary surfaces in the environment. This is generally known as *augmented reality* (AR) – the augmentation of the physical reality with virtual imagery. There are several possible approaches, although the most commonly seen methods are spatial AR, mobile AR and immersive (head-mounted display) AR. A fourth category of augmented reality, on-body interaction, seeks to place interfaces directly on user's own bodies.

2.3.1 Spatial Augmented Reality

A common path to on-world display is to project content onto the surfaces using data projectors, placing the virtual content directly on the physical objects. This is also commonly known as projection mapping within the visual effects, art and advertising domains, enabling complex visual shows and interactive exhibits to be implemented using simple materials and projection units. Spatial augmented reality has a long history, and the interested reader is directed to [Bimber 2005] for a more complete discussion of this topic.

A few approaches for projecting onto the complex and often irregular geometry of everyday environments are especially relevant to my thesis work. For example, The Office of the Future [Raskar 1998] proposed using 3D head tracking and office-wide depth sensing to project imagery onto irregular surfaces, such that it would appear perspectively correct from the user's view. iLamps [Raskar 2003] used structured light to sense the 3D geometry of a projection surface (e.g. multiple walls or curved surfaces) and uses this data to minimize visual distortion of projection output. Finally, depth cameras have made it easier to sense the geometry of an environment and perform projection mapping. This enables real-time rectification onto moving targets, as shown in *e.g.*, OmniTouch [Harrison 2011].

2.3.2 Mobile Augmented Reality

Another approach for augmented reality is the use of handheld devices to display virtual content overlaid on live camera imagery, via video see-through. Users may interact with the virtual content through the handheld's touchscreen or by physically moving the handheld. This approach has recently gained prominence due to the popularity and ubiquity of mobile phones.

An early implementation of mobile augmented reality was explored in [Wagner 2003], in which a PDA handheld was modified with a camera addon to support augmented video. Common application areas for such handheld augmented reality systems include *e.g.*, supporting physical navigation [Mulloni 2011], mobile gaming (*e.g.* augmented tabletop games), or providing low-cost shared virtual experiences [Wagner 2005]. However, the need to constantly hold the handheld up to employ the camera precludes convenient use of this approach in everyday contexts, in which interactions with both hands are desirable. In my present work, I mainly focus on interactions with the surfaces directly, without the indirection introduced by mobile augmented reality systems.

2.3.3 Head-Mounted Augmented Reality

Recent technological developments have made head-mounted augmented reality devices, or head-mounted displays (HMDs) possible. Such devices typically take the shape of glasses or helmet-like devices, with translucent displays overlaying virtual content on top of the physical world. While these systems were previously available as heads-up displays for specialized applications (*e.g.* military pilot interfaces), technological improvements have brought such devices in range of ordinary consumers. Augmented reality glasses (*e.g.* the Epson Moverio) and headsets (*e.g.* the Microsoft HoloLens) have recently become available to consumers.

HMD-based augmented reality makes it possible to place virtual content anywhere within a physical space, creating immersive experiences. For this reason, HMD-based augmented reality sys-

tems are often said to provide *immersive augmented reality*, or IAR. While the field of IAR interactions is relatively new, explorations such as 3D collaborative video chat [Chen 2015] suggest powerful capabilities and promise for this platform in the future.

A vast range of input techniques have been proposed for use in mixed-reality systems [Zhou 2016], such as gaze input [Bâce 2016, Park 2008], hand gestures [Dorfmuller-Ulhaas 2001, Leap Motion, Lee 2007], voice input [Bolt 1980], physical controllers (using physical buttons [Microsoft Hololens], motion sensing, or external tracking [HTC]), tangible interfaces [Billinghurst 2004, Ishii 1997], or multimodal combinations of these approaches (*e.g.,* [Bolt 1980, Chatterjee 2015]). Of these, gaze, gesture and voice provide no haptic feedback, while controllers and tangible interfaces require additional external hardware to function.

2.3.4 On-Body Interfaces

The primary goal of on-body interfaces is to provide "always-available input", where a user does not need to carry or pick up a device [Saponas 2009, Tan 2010]. To support this class of interactions, numerous approaches have been considered. The simplest is to take conventional physical computing elements and place them on the body; iconic examples include a one-handed keyboard [Lyons 2004] and a wrist-bound touchpad [Thomas 2002]. Integrating input capabilities into clothing has also been the subject of considerable research [Mann 1997, Post 1997].

A more practical approach is to employ projection to project interfaces directly onto the body. Perhaps unsurprisingly, the art community was among the first to embrace the fusion of projected media and the human form. Examples include the opening sequence to Guy Hamilton's "Goldfinger" (1964) and Peter Greenway's "The Pillow Book" (1996), both of which projected imagery onto actors' bodies for dramatic effect. More recently, an interactive installation by Sugrue [Sugrue 2007] allowed visitors to touch a screen containing virtual "bugs" that would move out onto people's hand and arms via overhead projection. [Barnett 2009] provides a survey of many of these artistic efforts.

Owing to the size, weight and power consumption of typical computers and projectors, it is most common to find on-body projection systems installed in the environment. For example, TenoriPop [NTT 2010] rendered interactive graphics onto the hands of shoppers using a ceiling mounted projector/camera rig. Similar overhead setups have also been considered for medical uses, where *e.g.*, anatomy can be overlaid onto bodies for surgical assistance [Gavaghan 2011] and education [Donnelly 2009, Patten 2007]. In the HCI literature, LightSpace [Wilson 2010b], LightGuide [Sodhi 2012] and Armura [Harrison 2012] all used overhead setups to enable a variety of on-body projected interactions.

Rarest are *worn* systems that attempt both input and graphical output on the body. This is a nascent, but growing literature, often referred to as "on-body interfaces". Early systems include [Sakata 2009], which describes a "palm top" projection system using fiducial markers worn on the wrist to provide 6DOF position tracking of a wearer's hand. Similarly, SixthSense [Mistry 2009] tracked color markers worn on the fingers with a neck-worn camera to detect finger input; an integrated pico-projector could render interfaces onto the body or environment.

Other work has aimed to avoid instrumenting users with markers. For example, Skinput [Harrison 2010c] – worn on the upper arm – relied on bioacoustic signals resulting from touches to the skin. Continuous finger tracking was not possible, so projected interfaces were built around pre-learned touch locations. Using computer vision, PALMbit [Yamamoto 2007] could track finger-to-finger touches without markers, enabling a projected interface on the palm. Finally, OmniTouch [Harrison 2011] used a shoulder-worn computer vision system to track free-form multitouch finger inputs on the body and environment.

2.4 Touch Tracking on Large Surfaces

There are many different approaches for touch tracking on large surfaces. The simplest approach is to create a special purpose surface, using *e.g.* cameras [Han 2005, Matsushita 1997] or capacitive sensors [Lee 1985, Wei 2010]. Alternatively, existing surfaces can be retrofitted with sensors, such as acoustic sensors to detect the sound of a tap [Paradiso 2002, Xiao 2014], or infrared

emitters and receivers to detect occlusion from a finger. There are also methods that can operate on ad hoc, uninstrumented surfaces. These systems most often use optical sensors, detecting fingers using finger template matching [Koike 2001], skin-color segmentation [Letessier 2004], contour segmentation [Chang 2005], thermal imprint tracking [Saba 2012], or LIDAR [Paradiso 2000].

Detecting whether a finger has contacted a surface is challenging with conventional RGB or infrared cameras, which has inspired several approaches. PlayAnywhere [Wilson 2005] demonstrated a touch tracking approach based on analyzing the shadows cast by a finger near the surface. Sugita *et al.* [Sugita 2008] detect touches by tracking the visual change in the fingernail when it is pressed against a surface. TouchLight [Wilson 2004] uses a stereo pair of cameras, detecting touches when the finger images pass beyond a virtual plane. Many other systems use finger dwell time or an external sensor (accelerometer [Kane 2009], microphone, or acoustic sensor [Paradiso 2002, Xiao 2014]) to detect touch events.

Most related to the techniques I present in this thesis are depth camera-based touch-tracking systems. Depth cameras sense the physical distance from the sensor to each point in the field of view, making it possible (in concept) to innately sense whether a finger has contacted a surface or not. Broadly, these systems can be placed into two categories:

Background modeling approaches compute and store a model or snapshot of the depth background. Touches are detected where the live depth data differs from the background depth map in specific ways. Wilson [Wilson 2010a] uses a background snapshot computed from the maximum depth point observed at each pixel over a small window of time. KinectFusion [Izadi 2011] uses a background model developed by analyzing the 3D structure of the scene from multiple angles (SLAM), effectively producing a statistically derived background map. WorldKit [Xiao 2013] also uses a statistical approach, computing the mean and standard deviation for each depth pixel, modeling both the background and noise. Finally, MirageTable [Benko 2012] captures a background mesh and renders foreground hands as particles.

Finger modeling approaches attempt to segment fingers based on their physical characteristics, and generally do not require background data. OmniTouch [Harrison 2011] used a template-finding approach to label finger-like cylindrical slices within depth images. The slices are then merged into fingers, and finally touch contacts. FlexPad [Steimle 2013] detected and removed hands by analyzing the subsurface scattering of the Kinect's structured infrared light pattern, allowing the background to be uniquely segmented.

Surprisingly few systems attempt to fuse depth sensing with other sensing modalities for touch tracking. Of the existing literature on sensor fusion depth-sensing systems, only the Dante Vision project [Saba 2012] uses a multisensory approach for touch tracking, combining depth sensing with thermal imaging infrared camera. This is used to improve touch contact detection accuracy (a thermal imprint is left on the surface upon physical touch), though at the expense of continuous tracking and high contact latency (~200 ms).

CHAPTER 3. INITIAL EXPLORATIONS

My interest in on-world interaction was informed and shaped by my initial forays into human-computer interfaces. In this chapter, I describe a few of the key projects along this research trajectory. I start by exploring the general topic of interactions with objects *in* the real world with EMSense, then narrow my focus to interactions *across displays* in the world with CapCam and UbiCursor. To complement interactions with objects, I also consider interactions with surfaces in Toffee, setting the stage for true on-world interactions in the remainder of this proposal.

3.1 Interacting with Infrastructure in the World: Deus EM Machina

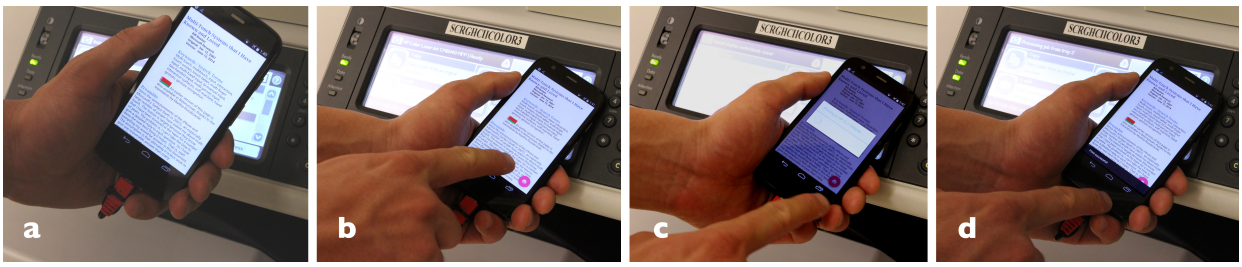

Figure 3.1. EM-Sense Phone "Print Document" contextual charm.
While the user is reading a document (a), they can tap the phone on a printer to bring up a "print" charm (b). Activating the charm spools the document to the printer (c), which prints it immediately (d).

One approach to developing on-world interaction is to interact with intelligent devices already embedded in the environment. We are surrounded by an ever-growing ecosystem of connected and computationally-enhanced appliances, from smart thermostats and light bulbs, to coffee makers and refrigerators. The much-lauded Internet of Things (IoT) revolution predicts billions of such devices in use by the close of the decade [Gartner 2015]. Despite offering sophisticated functionality, most IoT devices provide only rudimentary on-device controls. This is because 1) it is expensive to include *e.g.*, large touchscreen displays on low-cost, mass-market hardware, and 2) it is challenging to provide a full-featured user experience in a small form factor. Instead, most IoT appliances rely on users to launch a special-purpose application on their smartphones or

browse to a specific webpage in the cloud or on their local area network. Quintessential examples include "smart" light bulbs (*e.g.*, Philips Hue), media devices (*e.g.*, Chromecast), Wi-Fi cameras (*e.g.*, Dropcam) and internet routers.

Clearly, this manual launching approach will not scale as the number of IoT devices grows. If we are to have scores of these devices in our future homes and offices—as many prognosticate—will we have to search through scores of applications to dim the lights in our living room or find something to watch on TV? What is needed is an instant and effortless way to automatically summon rich user interface controls, as well as expose appliance-specific functionality within existing smartphone applications in a contextually relevant manner.

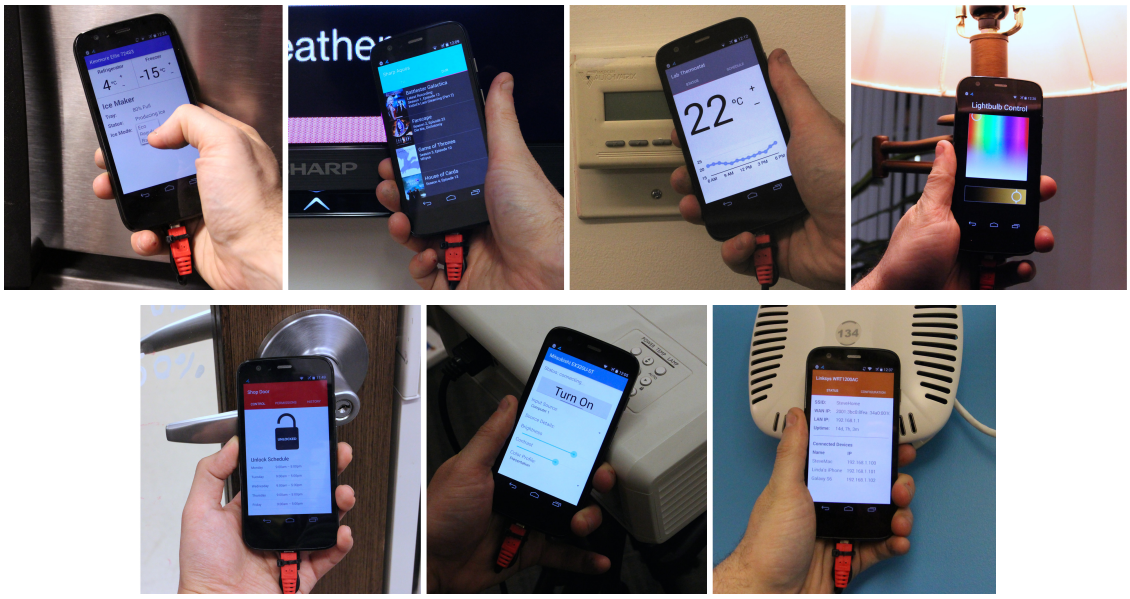

Figure 3.2. Example Deus EM Machina applications.
Left to right: top row: refrigerator, television, thermostat, lightbulb. Bottom row: door lock, projector, and wireless router.

To explore these capabilities, we built Deus EM Machina, a hardware augmentation for smartphones which senses electromagnetic interference. Deus EM Machina is capable of sensing the electromagnetic noise emitted by all electronic devices, enabling the phone to detect nearby appliances. We used Deus EM Machina to explore two approaches to mitigate the IoT interaction

bottleneck. The most straightforward option is to *automatically* launch manufacturers' applications instantly upon contact with the associated appliance. For example, touching a smartphone to a thermostat launches the thermostat's configuration app (Figure 3.2). In this case, the currently running app on the smartphone is swapped out for a new full screen app. Alternatively, the phone can expose what we call *contextual charms*—small widgets that allow the running smartphone application to perform actions on the touched appliance. For example, when reading a PDF, touching the phone to a printer will reveal an on-screen print button (Figure 3.1).

This general vision of rapid and seamless interaction with connected appliances has been explored many times in prior work (*e.g.*, [Hodes 1997, Olsen 2008, Schmidt 2012]), and in this work we set out to practically achieve it. We created full-stack implementations for several of our example applications to show that the interactions are realizable today. In cases where appliances have proprietary APIs, we can automatically launch the manufactures' smartphone app. For IoT devices with open APIs (fortunately the trend), contextual charms can expose appliance-specific functionality across the smartphone experience.

To recognize appliances on-touch, we had to significantly extend the technical approach proposed by Laput *et al.* in EM-Sense [Laput 2015]—a smartwatch that detected electromagnetic emissions of grasped electrical and electromechanical objects. Critically, our technical approach requires no modification or instrumentation of appliances, as it senses noise already produced by devices during normal operation, and can therefore work "out of the box" with already-deployed devices.

In summary, in Deus EM Machina, I explored a novel system that allows instant recognition of uninstrumented electronic appliances, which in turn allows us to expose contextual functionality via a simple tap-on-device interaction. We demonstrated substantially better accuracy than prior work (98.8% recognition of 17 unmodified appliances in a user study), while running entirely on an augmented smartphone. In addition to conventional full-screen control applications, we contributed "contextual charms", a new cross-device interaction technique. Finally, in contrast to most prior work, we created *truly* functional implementations for many of our example demos using existing IoT capabilities.

3.2 Facilitating Across-Device Interaction with Large Displays in the World: CapCam

Although small smartphones are powerful, especially when augmented with techniques like Deus Ex Machina, they still cannot do everything. For one, they cannot enable the kinds of expansive, free-form creativity and ideation offered by large canvases, such as whiteboards and TV-sized touchscreens. Indeed, one possible computational future is the integration of smart whiteboards and wall-sized touchscreens into everyday contexts.

Large touchscreen displays, such as public kiosks, digital whiteboards and interactive tabletops have become increasingly popular as prices have fallen. Similarly, mobile devices, such as smartphones and tablets, have achieved ubiquity. While such devices are reasonably smart on their own, cross-device interactions hold much promise for making interactive experiences even more powerful [Hinckley 2004].

However, interacting *across* devices is rarely straightforward. Although many mobile devices support *e.g.*, Bluetooth pairing, such pairing options are generally time-consuming and cumbersome (*i.e.*, on the order of 5 seconds, see Table 1). Often, users must confirm or manually enter connection parameters (e.g., device, network identifier) and security parameters (*e.g.*, PIN) [Rekimoto 1997]. Short-range NFC, an emerging technology, aims to mitigate many of these issues. However, it requires specific hardware on both devices, and more importantly, only indicates device presence, not position (unless receivers are tiled into a matrix [Seewoonauth 2009] or combined with another method, like optical fiducial tags [Bazo 2014]). Thus, it only allows for coarse device-to-device pairing, precluding metadata such as spatial position and rotation of devices, as well as rich multi-device experiences. Moreover, NFC is not commonly available on larger devices, such as laptops, tablets, and interactive surfaces – the class of surfaces we chiefly target.

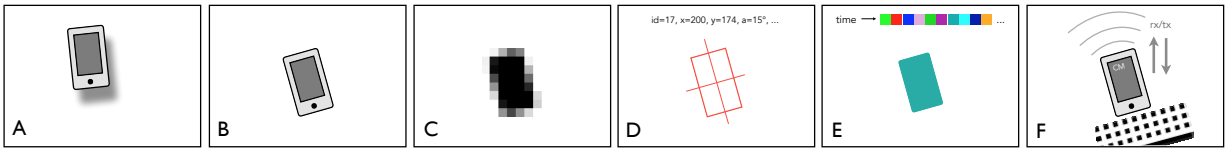

Figure 3.3. CapCam pairing and interaction process.
A: CapCam is used to pair two devices, a "cap" device (background is a large touchscreen display) and a "cam" device (here, a smartphone). B: The phone is pressed to the display. C: The phone body creates a characteristic signal on the touchscreen's capacitive sensor. D: CapCam extracts the shape, position and orientation of the phone from this capacitive image. E: CapCam encodes pairing data (e.g., IP, port and password) as a flashing color pattern, rendered beneath the phone body. F: The phone's rear camera captures the pattern, and uses it to establish a conventional two-way wireless link (e.g., WiFi). With both devices paired and communicating, interactive applications can be launched, such as this virtual keyboard.

In response, we developed *CapCam*, a new technique that provides rapid, ad-hoc connections between two devices. CapCam pairs a "cap" device with a capacitive touchscreen to a "cam" device with a camera sensor (Figure 3.3A). For example, typical smartphones and tablets can be paired with each other, and these devices can be paired to even larger touchscreens, such as smart whiteboards and touchscreen monitors. CapCam uses the cap device's touchscreen to detect and track the cam device (Figure 3.3, C and D), and renders color-modulated pairing data that is captured by the cam device's rear camera (Figure 3.3E).

This pairing data contains configuration information necessary to establish a bidirectional link (e.g., IP address, port and password). In this way, CapCam provides a unidirectional communication mechanism from the touchscreen to the camera, which is then used to bootstrap a full bidirectional, high-speed link (Figure 3.3F). Because CapCam also provides precise, continuous spatial tracking, we can enable rich synergistic applications utilizing both (or many) devices at once.

Overall, we believe CapCam exhibits six desirable properties – it enables *zero-configuration* pairing via automatically transmitted pairing codes; it is *rapid*, capable of establishing links in roughly one second; *anonymous*, in that it requires no identifying information to be exchanged; pairing is explicitly initiated by users through a *purposeful* pressing of a device to a host screen; it enables *targeted* interactions on said screen via position tracking; and, it allows for *multiple devices* to be paired and used on the same cap device simultaneously.

Although many prior systems have independently addressed pairing or spatial interaction, few have *combined* these into a single system. CapCam provides both pairing and spatial interaction as phases of a single interactive transaction, enabling rapid, ad hoc interactions, e.g. walking up to a public display and initiating rich, spatial interactions nearly instantaneously.

We also developed several applications and interactions enabled by CapCam (such as a two-player air hockey game, seen in Figure 3.4), and performed an evaluation of the technical aspects of our approach, including pairing latency, pairing code bandwidth and bit error rate across three exemplary devices.

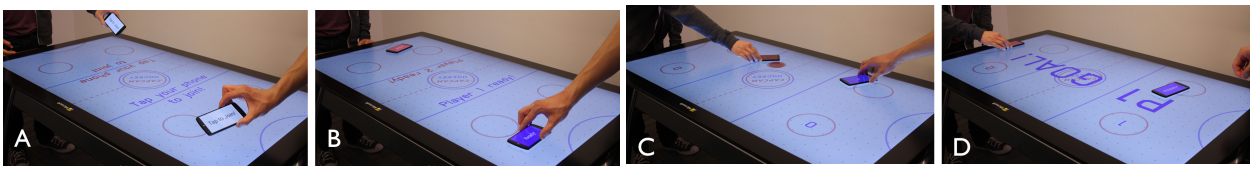

Figure 3.4. CapCam air hockey.
(A) Players are invited to join the game. (B) When players press their phones to the table, CapCam rapidly and anonymously pairs the phones to the display. When two phones are paired, the game begins. (C) Players use their physical phones to deflect the virtual puck. CapCam tracks the phones and their orientations on the screen. (D) Sounds, vibrations and player-specific information appear on the phone, directed by the game through the paired connection.

3.3 Facilitating Across-World, Large-Display Interaction: UbiCursor

If the entire environment consists of a constellation of small and large displays and devices, how can you interact across multiple such devices? Multi-display environments (MDEs) are systems in which several display surfaces create a single digital workspace, even though the physical displays themselves are not contiguous. There are many different types of MDE: dual-monitor computers are a simple (and now ubiquitous) example, but more complex environments are also now becoming feasible such as control rooms with multiple monitors in multiple locations, meeting rooms with wall and table displays, or ad-hoc workspaces made from laptops and mobile devices.

One main problem in MDEs is that of moving the cursor from one display to another [Nacenta 2009]. This is essentially a targeting task, but one that differs from standard targeting in that the

visual feedback is fragmented based on the locations and sizes of the physical displays. In some situations, displays may be far apart, or may be at different angles to one another or to the user. The composition of the MDE and the arrangement of physical displays can have large effects on people's ability to move between visible surfaces.

There are two common ways in which MDE workspaces can be organized: 'warping' and perspective-ether approaches. Warping means transporting the cursor directly from one display to another, without moving through the physical space between monitors. Several techniques for warping have been developed, such as *stitching* (which warps the cursor as it moves across specific edges of different displays) [Benko 2007], *wormholes* (which warp the cursor when it moves into a specific screen region), *warp buttons* (in which pressing a software or hardware button moves the cursor to each display) [Benko 2005a], or *named displays* (in which the user selects the destination display from a list).

Warping techniques can be fast and effective for cross-display movement. However, they suffer from a number of problems. Warping requires that the user remember an additional mapping (edges, holes, buttons, or names), which might take time to learn; in some techniques (such as stitching), the mappings may become incorrect when the user moves to a new location in the environment. Warp techniques are also distinctly less natural than regular mouse movement: they introduce an extra step into standard targeting actions, and make it more difficult for the user to plan and predict the result of ballistic movements [Nacenta 2008]. Finally, the instantaneous jumps of warping techniques cause major tracking and interpretation problems for other people who are trying to follow the action in the MDE.

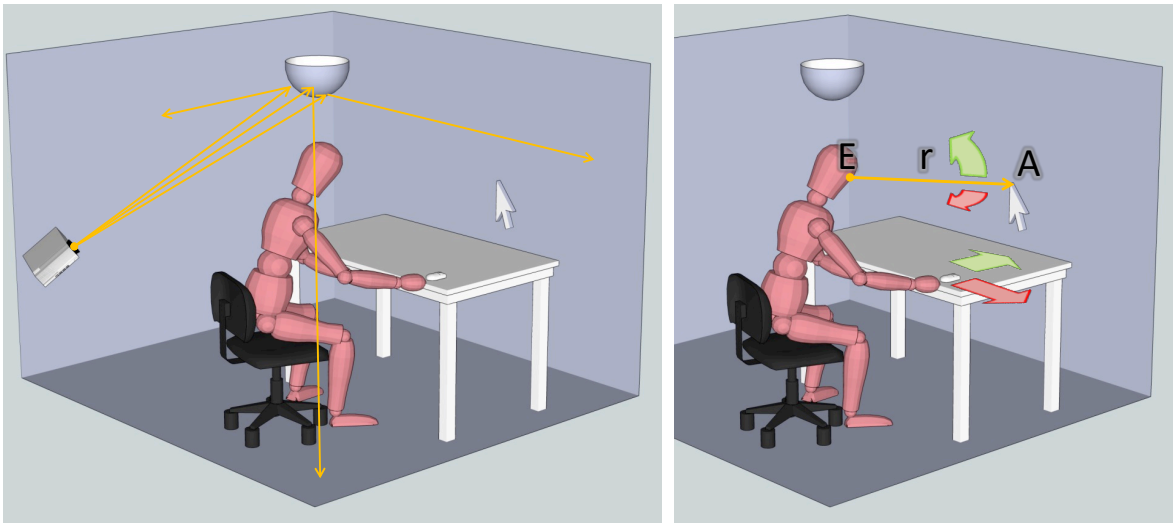

Figure 3.5. UbiCursor low-resolution full-coverage display (LRFC).
Left: schematic of the LRFC display. By reflecting onto the spherical mirror, the projector can project onto almost any surface. Right: the movements of the mouse cause a change in the orientation of perspective cursor's defining ray.

Perspective-ether techniques for cross-display movement are a different approach that addresses these problems. In this approach, the entire environment is considered to be part of the workspace, including the space between the displays (i.e., 'mouse ether' [Baudisch 2004]). The visible parts of the workspace, corresponding to the physical displays, are then arranged based on what the user can see from their current location and perspective. Perspective-ether MDE views provide a workspace in which cursor movement behaves as the user expects, and in which the arrangement of displays corresponds exactly to what the user sees in front of them.

The natural mapping of a perspective-ether view, however, comes at the cost of having to include the 'ether' (i.e., the real-world space between monitors) in the digital workspace. This implies that in order to get from one display surface to another, users must move through a displayless region where there is no direct feedback about the location of a cursor. This is not a major problem with ray-casting solutions (e.g., 'laser pointing'), but does affect indirect pointing devices such as mice or trackpads. One current solution is to use the available display surfaces to provide indirect feedback about the location of the cursor – that is, each display provides feedback (such as an arrow or halo) to indicate the location of the cursor in displayless space.

Although indirect feedback for non-displayed targets can be effective (e.g., [Gustafson 2008]), it does require that the user perform (sometimes complex) estimation and inference to determine the cursor's actual location, making cross-display movement more difficult than movement within a display. To address the problems of indirect cursor feedback, we designed a simple solution: provide direct visual feedback about the location of the cursor in 'displayless' space.

We have built a novel display system, called a Low-Resolution Full-Coverage (LRFC) display that can accomplish this solution in any multi-display environment. The LRFC system uses a data projector pointed at a hemispherical mirror to blanket the entire room with addressable (although low resolution) pixels. Using an LRFC display to provide feedback about the cursor in the empty space between monitors results in a technique called *Ubiquitous Cursor* (or *UbiCursor*). The projector only draws the cursor in the space between physical monitors, and uses simple room measurements to ensure that the cursor is shown in the correct location for the user. The result is fast, accurate, and direct feedback about the location of the cursor in 'displayless' space. Our goal is not to turn the entire room into a surface for showing data [Welch 2000] – only to provide information about objects that are between physical displays.

To test the new technique, we ran a study in which participants carried out cross-display movement tasks with three types of MDE: stitched, perspective-ether with indirect feedback, and perspective-ether with direct feedback (i.e., Ubiquitous Cursor). Our study showed that movement times were significantly lower with UbiCursor than with either stitching or indirect feedback. This work is the first to demonstrate the feasibility of low-resolution full-coverage displays, and shows the value of providing direct cursor feedback in multi-display environments.

Multi-display environments present the problem of how to support movement of objects from one display to another. We developed the Ubiquitous Cursor system as a way to provide direct between-display feedback for perspective-based targeting. In a study that compared Ubiquitous Cursor with indirect-feedback Halos and cursor-warping Stitching, we showed that Ubiquitous Cursor was significantly faster than both other approaches. Our work shows the feasibility and the value of providing direct feedback for cross-display movement, and adds to our understanding of the principles underlying targeting performance in MDEs.

3.4 Ad Hoc Touch Sensing on the World: Toffee

UbiCursor solves the problem of *projecting* onto the world, but naturally we want to ask if it is possible to reach out and *touch* the projections. Touch sensing on the world is an entirely separate problem, and so Toffee was born out of a desire to experiment with on-world touch sensing.

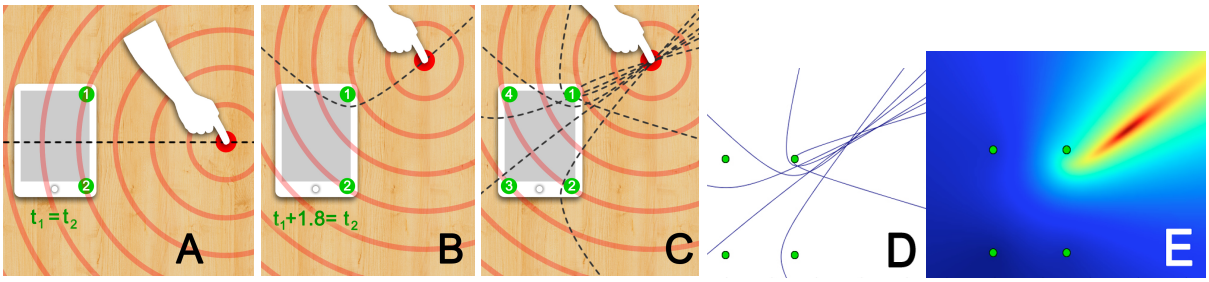

Figure 3.6. Toffee tap tracking process.
A: When a finger taps equidistant to two sensors, such that arrival time t_1=t_2, it is only possible to infer that the finger tapped somewhere along a line (dashed). B: If the finger lies closer to one sensor, this function becomes hyperbolic. C: In a four-sensor setup, six hyperbolas can be computed; intersections are solutions for the originating touch location. D: Real world data of the scenario in C plotted in matplotlib. E: A visualization of total squared error.

In order for mobile devices to fit into our pockets and bags, they are generally designed with small screens and physical controls. Simultaneously, human fingers are relatively large (and are unlikely to shrink anytime soon). This has led to the recurring problem of limited surface area for touch-based interactive tasks. Thus, there is a pressing need to develop novel sensing approaches and interaction techniques that aim to mitigate this fundamental constraint.

One option is to transiently appropriate surface area from the environment around us [Harrison 2010b]. This allows devices to remain small, but opportunistically provide large areas for accurate and comfortable input (and potentially graphical output if e.g., projectors are used). Tables, in particular, have presented an attractive target for researchers. Foremost, mobile devices often reside on tables, enabling several ad hoc sensing approaches (e.g., vibro-acoustic [Harrison 2008, Kane 2009] and optical [Butler 2008, Kratz 2009]). Moreover, unlike e.g., painted walls, tables are accepted areas for work, have durable surfaces, and typically have surface area available for interactive use.

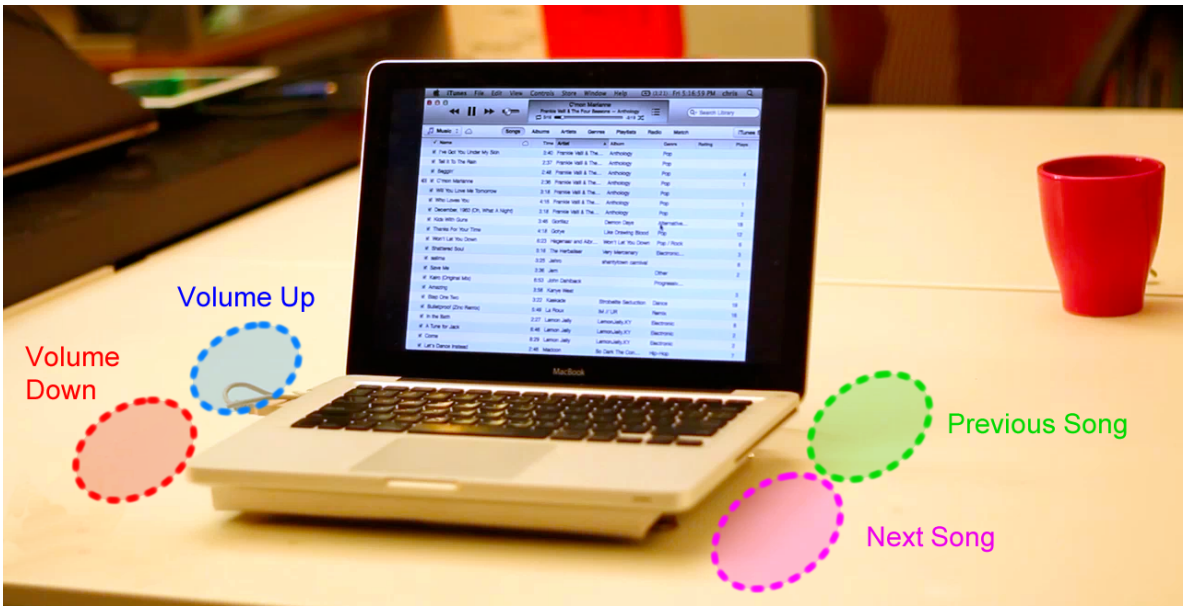

Figure 3.7. Toffee-enabled music player.
Music player controls can be bound to regions around the laptop, e.g., volume up and down.

To explore this approach, we built *Toffee*, a system that allows devices to appropriate tables and other hard, flat surfaces they are placed on for ad hoc radial tap input. This is achieved using a novel application of acoustic time differences of arrival (TDOA) analysis [Bancroft 1985, Caffery 1998, Carter 1981, Ishii 1999, Leo 2002, Paradiso 2005, Paradiso 2002] – bringing the technique, for the first time, to small devices in a way that is compatible with their inherent mobility. At a high level, Toffee allows users to define virtual, ad hoc buttons on a table's surface, which can then be used to trigger a variety of interactive functions, including application launching, desktop switching, music player control, and gaming. Leveraging the large surface area of the table, users can potentially trigger simple functionality eyes free.

Our study results suggest that resolving a true, 2D position (angle and distance) is not sufficiently robust for accurate interactive use. However, angle estimation is robust, with an average error of 4.3° on a laptop sized setup. Thus, we suggest that interactions should be built around radial interaction, similar to bezel interactions [Ashbrook 2008b] and peripheral free-space gesturing [Harrison 2009]. Our example applications, which use radial interactions, allow users to make use of the expanded envelope of interactive space surrounding laptops, tablets, and smartphones.

CHAPTER 4. ON-WORLD PROJECTION AND TOUCH SENSING

4.1 Introduction

Creating interfaces *in the world*, where and when we need them, has been a persistent goal of research areas such as ubiquitous computing, augmented reality, and mobile computing. In this chapter I discuss the *WorldKit* system, which supports very rapid creation of touch-based interfaces on everyday surfaces. Further, it supports experimentation with other interaction techniques based on depth- and vision-based sensing, such as reacting to the placement of an object in a region, or sensing the ambient light in one area of a room.

Figure 4.1. Sample interaction in WorldKit.
Using a projector and depth camera, the WorldKit system allows interfaces to operate on everyday surfaces, such as a living room table and couch (A). Applications can be created rapidly and easily, simply by "painting" controls onto a desired location with one's hand - a home entertainment system in the example above (B, C, and D). Touch-driven interfaces then appear on the environment, which can be immediately accessed by the user (E).

WorldKit draws together ideas and approaches from many systems. For example, it draws on aspects of Everywhere Displays [Pinhanez 2001] conceptually, LightWidgets [Fails 2002] experientially, and LightSpace [Wilson 2010b] technically. Based on very simple specifications (in the default case, just a simple list of interactor types and action callbacks) interfaces can be created which allow users to quite literally *paint* interface components and/or whole applications wherever they are needed (Figure 4.1) and then immediately start using them. Interfaces are easy enough to establish that the user could, if desired, produce an interface on the fly each time they

entered a space. This flexibility is important because unlike an LCD screen, the world around is ever-changing and configured in many different ways (e.g., our lab is different from your living room). Fortunately, we can bring technology to bear to overcome this issue and make best use of our environments.

Like LightSpace [Wilson 2010b], our system makes use of a projector and inexpensive depth camera to track the user, sense the environment, and provide visual feedback. However, our system does not require advance calibration of the spaces it operates in – it can simply be *pointed at* nearly any indoor space. Further, with a projector slightly smaller than the one used in our prototype, it could be deployed in a volume similar to a modern laptop. Indeed, as we show later in Chapter 7, the hardware has advanced considerably, making it possible to implement this approach in a truly mobile form. In addition, our system provides an extensible set of abstractions which make it easy and convenient to program simple interfaces while still supporting exploration of new interaction techniques in this domain.

In the next section, we will consider how users might make use of these created interfaces. We then turn to implementation details, discussing the hardware used, sensing techniques and other low level details. We will then consider how these basic capabilities can be brought together to provide convenient abstractions for *paint anywhere* interactive objects, which make them very similar to programming of conventional GUI interfaces. We then consider aspects of the software abstractions that are unique to this domain and describe an initial library of interactor objects provided with our system. Several example applications we built atop this library are also described. While previous systems have considered many of the individual technical capabilities built into our system in one form or another, the WorldKit system breaks new ground in bringing these together in a highly accessible form. The system enables both easy and familiar programmatic access to advanced capabilities, as well as a new user experience with dynamic instantiation of interfaces *when and where they are needed*.

4.2 Interaction

A core objective of our system is to make it simple for users to define applications quickly and easily, such that they *could* feasibly customize an application each time they used it. The default interaction paradigm provided by our system allows users to "paint" interactive elements onto the environment with their hands (Figure 4.1 and Figure 4.2). Applications built upon this system are composed of one or more *in-the-world interactors*, which can be combined to create interactive applications.

By default, when an interface is to be deployed or redeployed, a list of interactor types and accompanying callback objects is provided – one for each element of the interface. The application *instantiates* each interactor using a specified instantiation method, defaulting to user-driven *painted* instantiations. For these elements, the system indicates to the user what interactor is to be "painted". The user then runs his or her hand over a desired surface (Figure 4.1A and Figure 4.2A,B). Live graphical feedback reflecting the current selection is projected directly on the environment. When satisfied, the user lifts their hand. The system automatically selects an orientation for the interactor, which can optionally be adjusted by dragging a hand around the periphery of the selected area, as shown in Figure 4.2D. This completes the setup for a single interactor. The user then paints the next interface element, and so on.

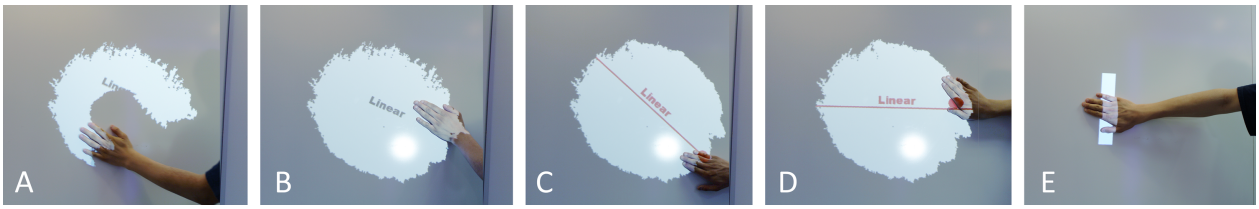

Figure 4.2. A user defines an interactor in WorldKit.
This sequence shows how a user can define the location, size and orientation of an interactor. First, the user starts painting an objects area (A, B), defining its location and size. Objects have a default orientation (C), which can be reoriented by dragging along the periphery (D). Finally, the interactor is instantiated and can be used (E).

Once all elements have been instantiated, the interface starts and can be used immediately. The entire creation process can occur very quickly. For example, the living room application sequence depicted in Figure 4.1 can be comfortably completed within 30 seconds. Importantly, this process need not occur every time – interactor placements can be saved by applications and reused the next time they are launched.

This approach offers an unprecedented level of personalization and responsiveness to different use contexts. For example, a typical living room has multiple seating locations. With our system, a user sitting down could instantiate a custom television interface using surfaces in their immediate vicinity. If certain controls are more likely to be used than others (*e.g.*, channel switching), these can be placed closer to the user and/or made larger. Other functions could be omitted entirely. Moreover, users could layout functionality to match their ergonomic state. For example, if lying on a sofa, the arm rests, skirt or back cushions could be used because they are within reach.

4.2.1 Triggering Interfaces and Interface Design

Triggering the instantiation of an interface, including the design thereof can be achieved several ways. One option is for the system to be speech active. For example, the user could say “activate DVR” to bring up the last designed interface or “design DVR” to custom a new one. Alternatively, a free space gesture could be used, for example, a hand wave. A smartphone could also trigger interfaces and interface design, allowing for fine grain selection of functionality to happen on the touchscreen, and design to happen on the environment. Finally, a special (visible or invisible) environmental “button” could trigger functions.

4.3 System Implementation

4.3.1 Hardware and Software Basics

Our system consists of a computer connected to a Microsoft Kinect depth camera mounted on top of a Mitsubishi EX320U-ST short-throw projector (Figure 4.3). The Kinect provides a 320x240

pixel depth image and a 640x480 RGB image, both at 30 FPS. It can sense depth within a range of 50cm to 500cm with a relative error of approximately 0.5% [Khoshelham 2012]. Our short-throw projector has approximately the same field-of-view as the depth camera, allowing the two units to be placed in the same location without producing significant blind spots. As shown in KinectFusion [Izadi 2011], the depth scene can be refined over successive frames, yielding superior accuracy.

The software controlling the system is programmed in Java using the Processing library [Processing]. It runs on e.g., a MacBook Pro laptop with a 2GHz Intel Core i7 processor and 4 GB of RAM. The system runs at around 30FPS, which is the frame rate of the depth camera.

Figure 4.3. A short throw projector with mounted Kinect.

4.3.2 One-Time Projector / Depth Camera Calibration

We calibrate the joined camera-projector pair using a calibration target consisting of three mutually perpendicular squares of foamcore, 50cm on a side, joined at a common vertex. The seven

non-coplanar corners of this target are more than sufficient to establish the necessary projective transform between the camera and projector, and the extra degrees of freedom they provide are used to improve accuracy via a simple least-squares regression fit.

As long as the depth camera remains rigidly fastened to the projector, the calibration above only needs to be performed once (*i.e.*, at the factory). The setup can then be transported and installed anywhere – the depth sensor is used to automatically learn about new environments without requiring explicit measurement or (re-)calibration in a new space. If the environment changes temporarily or permanently after interfaces have been defined by a user (*e.g.*, a surface being projected on is moved), it may be necessary to re-define affected interfaces. However, our interactive approach to interface instantiation makes this process extremely lightweight for even novice users.

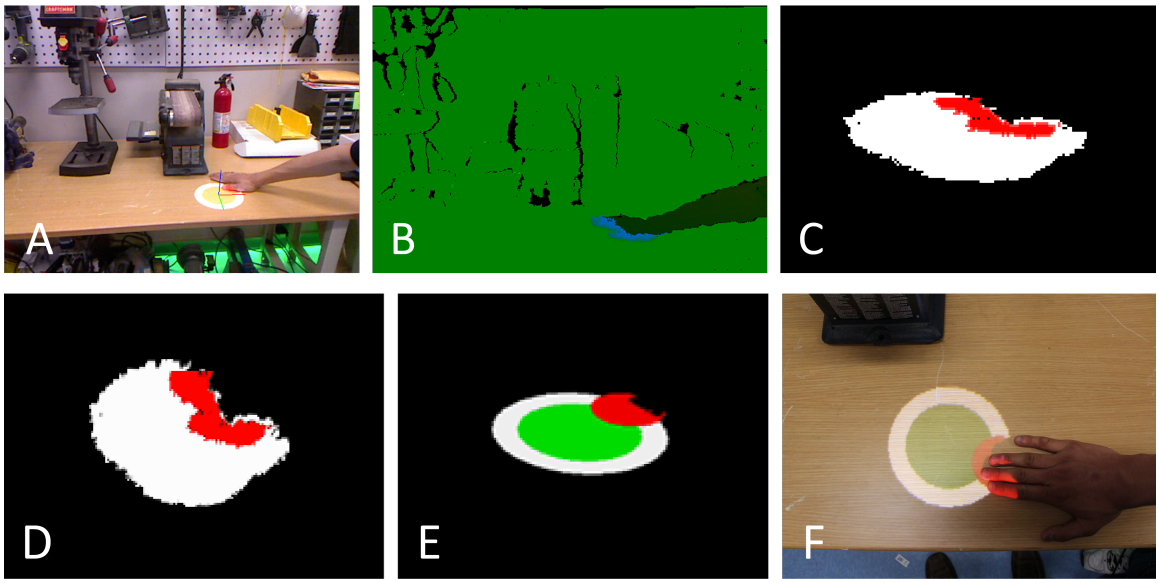

Figure 4.4. WorldKit touch event processing.
User touches an interactor placed on a surface (A - view from Kinect). The depth differences from the background are computed (B - green, no significant difference; dark green, differences over 50mm; blue, candidate touch pixels). The candidate contact pixels (red) are masked by the interactor's depth-image mask (white) (C). The contact pixels are transformed into the interactor's local coordinate system, providing an orthographic view (D). For output, interactor graphics are warped into the projector's image space (E), so that they appear correctly on a surface (F).

4.3.3 Basic Contact Sensing

Our system relies on surface contact sensing for two distinct purposes. First, when creating interfaces, touches are used to define interactor location, scale and orientation on the environment. This requires global touch sensing. Second, many interactor types (e.g., binary contact inputs), are driven by surface contact (i.e., touch or object contact or presence) data. To achieve this, we mask the global scene with each interactor's bounds; data from this region alone is then passed to the interactor for processing. In some cases (e.g., counting interactor), additional computer vision operations are completed internally (e.g., connected components for blob detection).

To achieve the highest quality sensing possible, we employ several strategies to filter the depth image. First, when the system starts up, we capture 50 consecutive depth frames and average them to produce a background profile. Note that this implies that the scene must be stationary and in a "background" configuration when the system is initialized. It is also possible to automatically accumulate a background image over a longer period of time to provide some ability to handle dynamic reconfigurations, e.g., moved furniture. Within the background image, the standard deviation at each pixel location across the frames is used as a noise profile. Subsequently, each observed depth value is divided by the computed baseline deviation at that pixel, and values that are greater than 3 standard deviations from the mean are considered significant (Figure 4.4B). Significant pixels which differ from the background surface by at least 3mm and at most 50mm are considered candidate *contact pixels*. We then perform blob detection across these candidates using a connected-components algorithm to further eliminate erroneous pixels arising from noise in the depth sensor.

This process yields a number of *contact blob images* in the depth camera's coordinate space over each interactor (Figure 4.4C, red). Each image is projectively transformed into the local coordinate system of the corresponding interactor (Figure 4.4D, red). From there, the blobs are passed to the corresponding interactor for type specific interpretation. For instance, the counting interactor from the system library simply uses the number of blobs intersecting that interactor, the multitouch interactor extracts the X-Y locations of each blob, and the area contact interactor

determines the total number of pixels across all blobs within that interactor. Custom types extended from the library classes are free to perform additional processing for special purposes.

4.3.4 Software Structures

The WorldKit system provides a set of programming abstractions that aim both to make it very simple to create simple to moderately complex interfaces and to allow custom interaction techniques to be quickly created and explored in this new domain. Many aspects of the system structure are designed to be as close as possible to the abstractions now provided in nearly all conventional GUI interface toolkits.

For example, interfaces are constructed as trees of objects, which inherit from a base *interactor* class (which has been called a *component*, or *widget* in various other systems). That class establishes the central abstraction for the system and defines an API (and default implementations) for a variety of interface tasks such as: hierarchy (parent/child) management, event-oriented input handling, damage tracking, layout, interface (re)drawing, etc. Since our goal is to stay as close to existing abstractions as we can, we expect that many aspects of the system will already be familiar to developers. See Figure 4.7 for a complete sample application.

To create a new interactor type (primitive), the developer extends the base interactor class (or an existing interactor class with similar functionality), adding new event sources, drawing commands and interaction logic. This is functionally similar to how developers would create new interactors in e.g., Java Swing.

In the following sections we only consider the aspects of the system that are different from typical systems (e.g., interactor instantiation by end users) or require special treatment inside our system to make them appear ordinary (e.g., rectification between 2D drawing and input spaces and surfaces in the 3D world).

4.3.5 Instantiating Interactors

One major difference between WorldKit abstractions and typical GUI toolkits is in how interactors are instantiated. In conventional systems, the details of instantiation are typically determined simply by the parameters to the constructor for an interactor (which may come from a separate specification such as an XML document, and/or are originally determined with a visual layout editor).

In contrast, in WorldKit we provide three options for interactor instantiation: *painted*, *linked*, and *remembered*. By default, interactors use painted instantiation – allowing the user to establish their key properties by "painting" them on the world as described below. For the base interactor class, key properties include size, position, and orientation, but this may be defined differently in specialized subclasses. Alternately, the developer may ask for *linked* instantiation. In that case, a small bit of code is provided to derive the key properties for the interactor from another instantiated interactor. This allows, for example, one key interactor to be painted by the user, and then a related group of components to be automatically placed in relation to it. Finally, *remembered* instantiation can be performed using stored data. This data can come from the program (making it equivalent to conventional interactor instantiation) or from a data structure saved from a previous instantiation of the same interface. This allows, for example, an interface element to be placed "where the user last left it".

For painted instantiations, users define interactor size, location and initial orientation by using a hand painting gesture over the surface where they wish it to appear (Figure 4.2A). During this process an area for the interactor is accumulated. At each step the largest contact blob over the entire depth image is considered. If this blob is larger than a preset threshold, the blob is added to a mask for the interactor. If no blob is larger than the threshold, we determine that the user must have lifted their hand from the surface, and the accumulated mask is saved as the user's painting selection. Note that this mask is defined over the depth image (i.e., in depth image coordinates). We then take the (x,y,depth) points in the depth image indexed by the mask and transform them into a world-space point cloud. Averaging the surface normals over this point

cloud produces the Z-axis of the planar region to be associated with the interactor. The X- and Y-axes lie in this plane, and their direction is controlled by the interactor's orientation.

The initial orientation aligns the Y-axis with the Y-axis of the depth image, which roughly corresponds to the direction of gravity if the depth sensor is mounted horizontally. As mentioned previously, the user can adjust the orientation by touching the interactor (Figure 4.2D).

4.3.6 Geometry Rectification for Input and Output

To provide a convenient API for drawing and input interpretation, the geometry of each interactor in our system is established in terms of a planar region in 3D space, derived as indicated above. Based on the X-, Y- and Z-axes of the interactor in the depth camera/projector coordinate system, we derive a *rectification matrix* mapping depth image coordinates into a local coordinate system for the interactor. This local coordinate system allows the developer to think about interaction drawing and input in simple 2D or surface terms. Full 3D information is available for use by advanced interactor classes if desired.

For input processing, the underlying depth and RGB images are updated 30 times per second. For each update we perform contact blob extraction as outlined earlier. For each interactor, we then intersect both the contact blob point cloud and the full depth image with the interactor's depth-image mask (Figure 4.4C, white). This produces raw depth and RGB images as well as contact areas limited to the region over the interactor. The rectification matrix is then applied to individual interactor depth, RGB, and contact images to produce *rectified* images (Figure 4.4D), i.e. images represented in the interactor's local 2D coordinate system. In a rectified image, one pixel width corresponds to a known unit of distance on the real world. Finally, contact areas are further processed to produce simplified touch events. All of this information is then passed to the interactor(s) concerned.

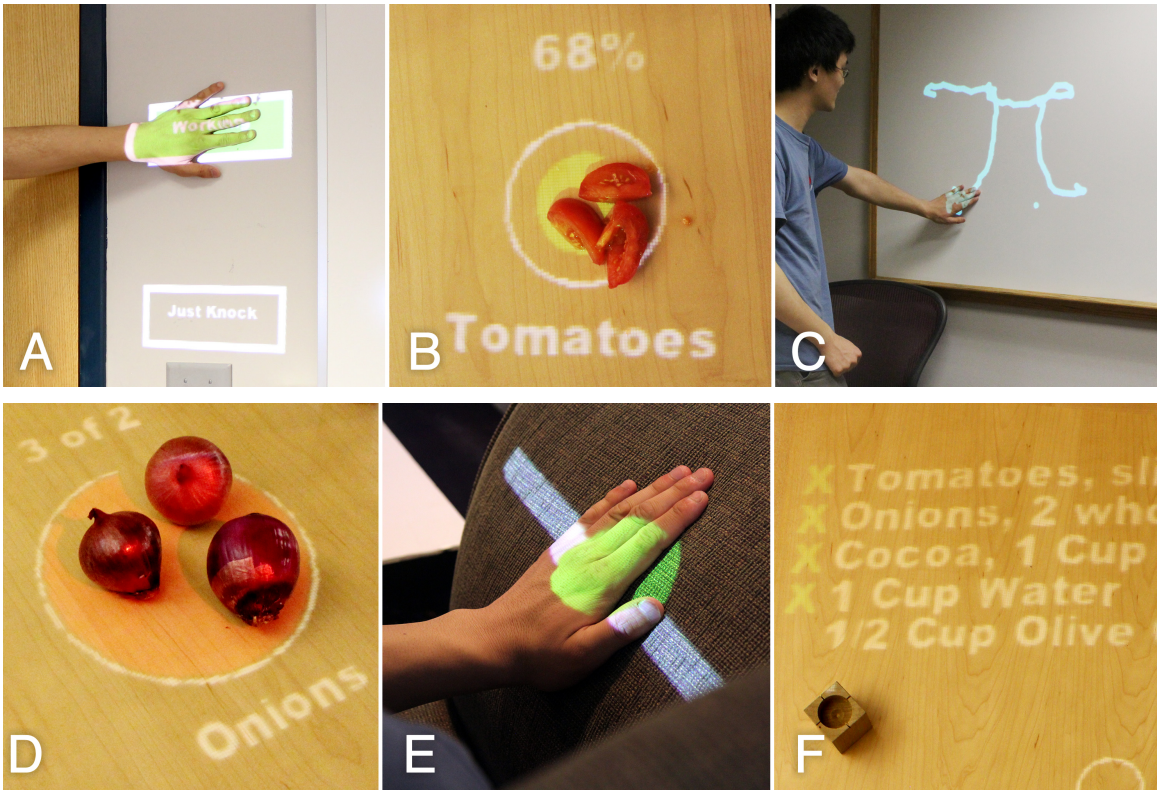

Figure 4.5. Sample interactor classes supported by WorldKit.
Our system provides a library of *interactor* classes which can be extended to perform many tasks. These including a binary contact interactor (A), percentage contact interactor (B), multitouch surface (C), object counting interactor (D), a linear axis interactor (E), as well as a simple output-only interactor (F). See also Table 4.1.

At this point, the interactor may perform additional specialized processing depending on their type. For instance, a brightness interactor from our library will calculate its sensed brightness value based on the rectified RGB image, a contact interactor will update its touch state and fire pressed/released events if applicable, and a multitouch interactor will act based on the contact blobs visible in its rectified image.

Each interactor may also produce output in order to indicate its current state and provide feedback during interaction. To facilitate easy drawing, the system provides a conventional two-dimensional drawing context (a `PGraphics` object within the Processing system) which is passed to an interactor's `draw()` method as needed. This drawing context object is transformed so that interactor drawing is specified in real world units (*e.g.*, millimeters) and oriented to correspond

to the interactor's real-world orientation (*e.g.*, aligned with its derived planar coordinate system as described above). The system takes draw commands on these graphics surfaces and automatically transforms them into the projector's image space for display (Figure 4.4E). Thus, when projected, interfaces render correctly on surfaces regardless of projector perspective, honoring interface layout and dimensions (Figure 4.4F). Finally, because we are projecting imagery onto real-world objects, head tracking is not required.

Interactor Type	Type of Associated Value
Binary contact	True or False
Area contact	Percentage of coverage
Presence	True or False
Contact counting	Number of items (contact blobs)
Linear axis touch	Centroid of touch (1D along axis)
Two axis touch	X/Y centroid of touch
Radial input touch	Angle to centroid of touch
Multitouch input	X/Y centroid of multiple touches
Brightness	Average brightness of surface
Color	Average color of surface

Table 4.1. WorldKit input-oriented interactor types.

4.3.7 Interactor Library

As a part of our system, we created an initial interactor library to support various capabilities of the platform (Figure 4.5). Part of this library is a set of reusable input-oriented base classes (listed in Table 4.1). From these base classes, we derived a set of traditional UI elements featuring both input and output, such as buttons and sliders.

As examples: the binary contact interactor detects events on a surface by examining the set of contact blobs reported to it. If the total pixel count for these exceeds a small threshold (to filter

out noise), the interactor detects a contact/touch. The area contact interactor additionally provides the proportion of depth values that are considered to be in contact range, allowing it to measure the contacted area. The presence interactor detects whether a background object is still present in its original configuration. For example, this can be used to sense if a door has been opened or a screen has been retracted.

A counting interactor counts and reports the number of distinct contact blobs on its surface. Linear axis interactors detect the position of a touch along one axis, which can be used to implement a variety of sliding controls, and multitouch interactors report the X and Y positions of each individual blob. Brightness interactors use the RGB camera feed to detect changes in the brightness of the sensed area. Similarly, color-sensing interactors measure the average color.

As in most systems, interactor types in our system are organized into a class hierarchy and may be extended from classes in the library by overriding methods associated with various interactive tasks. For example, advanced users might override an appropriate class to perform new or more advanced input processing such as hand contour analysis for user identification [Schmidt 2010] or recognition of shoes [Augsten 2010].

4.4 Example Applications

To illustrate the utility and capability of our system, we describe several example applications built using the accompanying library.

4.4.1 Living Room

The television remote control is frequently misplaced or lost in a typical home, leading to much consternation. With WorldKit, any surface can host the controls. This application instantiates a linear interactor to adjust the room's brightness, a radial interactor to adjust the TV volume and a Digital Video Recorder (DVR) interface to select a show of interest (Figure 4.1). Additionally, by adding a presence interactor to the sofa, we can even determine if the user is sitting and show or hide the interface as needed.

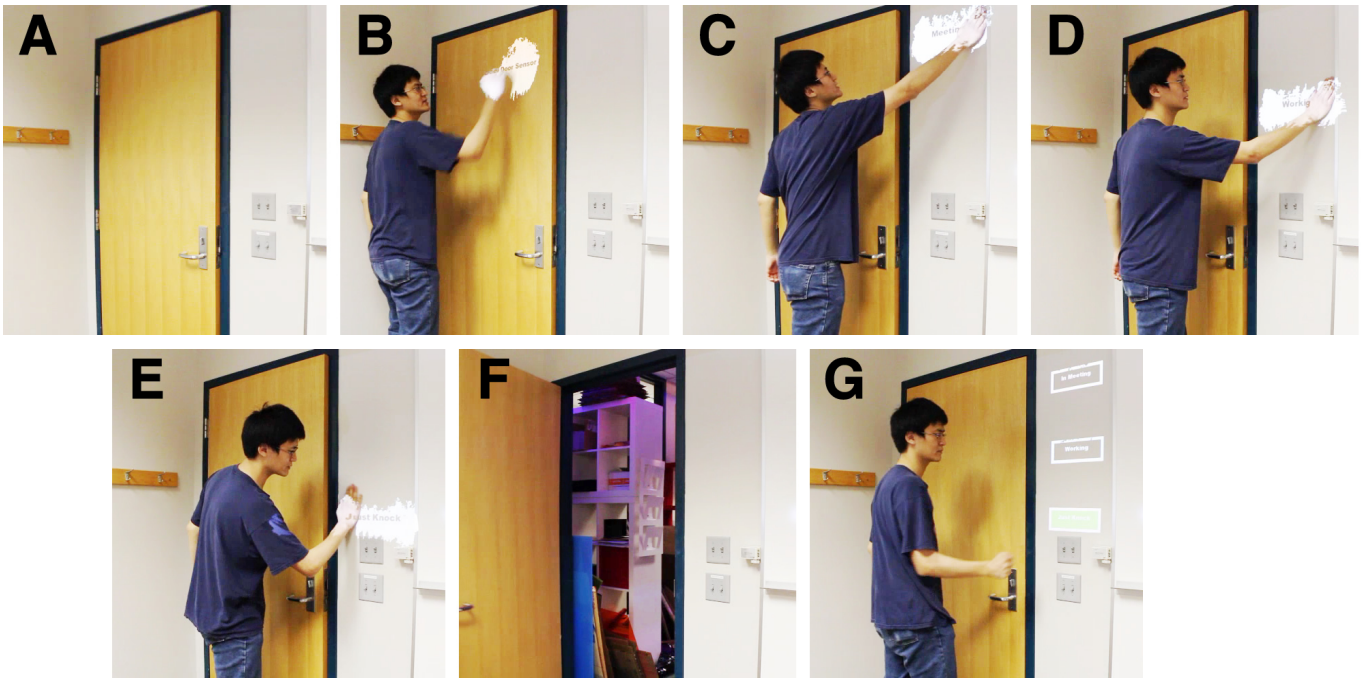

Figure 4.6. A user sets up a simple office status application on his door.
This sequence illustrates a user setting up a simple notification message application on an instrumented office (A). First, a user "paints" a presence interactor on an office door (B), which can detect if the door is open or closed. The user then paints three contact interactors, labeled "in meeting", "working" and "just knock", onto the wall adjacent to the door (C, D and E). When the door is open, the application is invisible (F). When the door is closed, three buttons appear, which are user selectable (G).

4.4.2 Office Door

A closed office door often gives no hints about the interruptible state of the occupant. A simple application of the WorldKit system allows the occupant to convey their status quickly when the door is closed. On the inside of the office, a large presence interactor is drawn on the closed door, and a number of smaller status buttons are drawn to the side (Figure 4.6). When the door is open, the status buttons are hidden, and the exterior indicator shows nothing. With the door closed, the status buttons appear. The exterior indicator reflects the chosen status button; thus, for instance, selecting the "In Meeting" status might cause "I'm in a meeting; please do not disturb" to appear on the outside.

```
import worldkit.Application;
import worldkit.interactors.Button;
import worldkit.interactors.ContactInput.ContactEventArgs;
import worldkit.util.EventListener;

public class OneButtonApp extends Application {
  Button button;

  public void init() {
    button = new Button(this);
    button.contactDownEvent.add(
      new EventListener<ContactEventArgs>() {
        @Override
        public void handleEvent(Object sender,
                                ContactEventArgs args) {
         System.err.println("Got a button event!");
        }
     });
     button.paintedInstantiation("OneButton");
  }

  /* Boilerplate */
  public static void main(String[] args) {
    new OneButtonApp().run();
  }
}
```

Figure 4.7. Example code for a single button WorldKit application.
The application depicted in Figure 4.6 consists of three buttons.

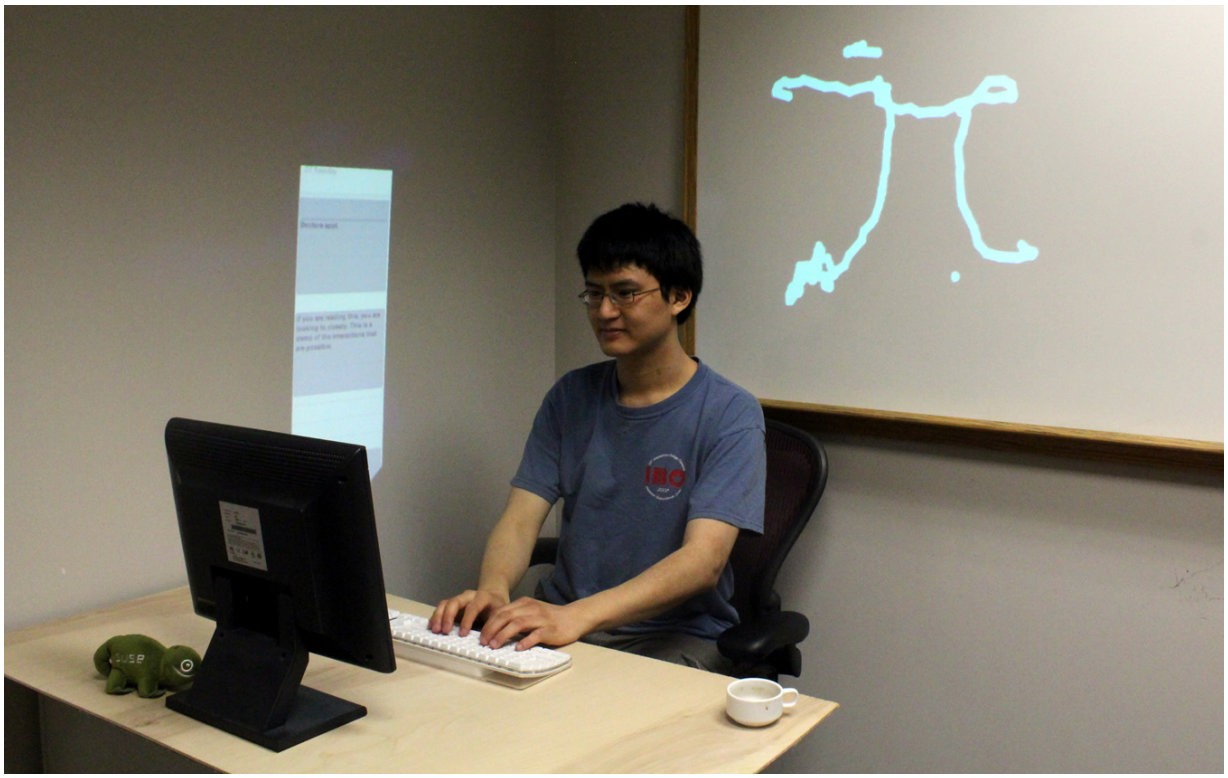

Figure 4.8. Simple WorldKit office application.
The whiteboard behind the user is an interactor. The calendar to the right of the user is visible only when the user's hands are on the keyboard.

4.4.3 Office Desk

This application uses a triggering contact interactor (in this case positioned over a keyboard) to activate a calendar display (which itself uses a linear interactor for scrolling) and a 2D position interactor for a simple whiteboard (Figure 4.8). When the user places his hands on the keyboard, the calendar appears. The user may then scroll the display through the calendar day by simply dragging up and down. Removing the hands from the keyboard causes the calendar to disappear. Users can also draw on the whiteboard, which is always visible regardless of the trigger state.

Figure 4.9. WorldKit kitchen application.
Various interactor types are composed into an ingredient management interface.

4.4.4 Kitchen

In the kitchen, it often becomes a chore to keep track of all the ingredients needed for a complex recipe. To address this, we created a simple recipe helper interface. The application prompts users to select a suitably-sized flat surface (e.g. kitchen counter) to prepare their ingredients. The user selects the desired recipe, and the application automatically lays out a set of interactors within that flat surface to hold each ingredient (Figure 4.9).

Interactors are customized for each ingredient to measure the amount or presence of the requested ingredient. For instance, if the recipe calls for a small number of countable items (e.g., eggs, whole onions), a counting interactor can be used. Ingredients that cannot be measured

easily in the framework can be replaced by contact interactors, which simply record the presence or absence of the ingredient.

The flexibility of our system enables the interface to be quickly reconfigured to suit different sets of ingredients and quantities, without any cumbersome calibration or instrumentation.

4.5 Limitations

As it is currently implemented, WorldKit has two notable drawbacks: the resolution of sensing and graphics can be lower than optimal, and the user may occlude the depth sensor and/or projector in certain configurations.

In the current system, the projector displays imagery at a resolution of 1024x768 over a potentially wide area. In cases where this area is large, this results in a loss of visual detail. We feel that this is not an inherent flaw in the approach, but rather a technological limitation that will improve with time as projectors develop increased resolution. Similarly the Kinect depth camera is also limited in spatial, temporal and depth resolution. However, future depth cameras promise to overcome these limitations. Finally, we note that for the interactions presented in this paper, lack of resolution did not significantly impede the usability of the resulting applications.

Users may occlude the projector and/or depth camera during normal operation; this is fundamentally a limitation of using a single projector and camera setup. In our system, we avoid user confusion by positioning the Kinect on top of the projector (i.e., very closely aligning effective view and display frustums). This ensures that users receive feedback in the form of their own shadow if they occlude the view of the camera.

4.6 Discussion

As indicated above, many of the underlying technical components we bring together in our system have been considered in prior work, and in some cases more than one technical approach has been offered over time. Specifically, we were initially inspired by the visions put forward in

the *Intelligent Room* [Brooks 1997], *Luminous Room* [Underkoffler 1999], and *Everywhere Displays* [Pinhanez 2001] projects. The benefits and goals of having interactivity everywhere were clearly articulated in these early works. However, the instantiation and modification of interactive features was absent or limited. In general, example applications were custom built by the authors, carefully configured and calibrated, and largely inflexible.

Eyepatch, *Slit-Tear Visualizations* and *Light Widgets* put forward elegant approaches to allow end-users to quickly define simple interfaces on a video stream of the environment. We extend this idea to defining interfaces *in situ* – directly on the environment, without the need for a conventional computer. Furthermore, we expand the suite of controls, allowing for richer interactions, including multitouch input and non-human triggers like doors closing.

Moreover, our system projects coordinated graphical feedback onto user-defined areas. This builds on and provides reusable abstractions for the technical approaches presented in *Office of the Future*, *LightSpace*, and *OmniTouch*. Our system takes into account the geometry of user-defined surfaces, providing not only rectified projected output (so graphics appear correctly for all users), but also orthonormal processing of input data (providing higher accuracy and regular coordinate systems).

Only with *all* of these features in place (and with the functionality and low cost of the most recent hardware advances) could we begin to think about mechanisms that are suitable for end users to define interactive functions on everyday surfaces in a highly dynamic fashion, allowing me to produce a robust, extensible, and usable system that makes interfaces accessible wherever and whenever they are needed by an end-user.

CHAPTER 5. ENABLING RESPONSIVE ON-WORLD INTERFACES

5.1 Introduction

Digitally augmented desks are a particularly interesting implementation of world-scale interfaces, and have been explored often in the literature. Seminal work emerged in the early 90's, most notably Xerox PARC's DigitalDesk [Newman 1992, Wellner 1993, Wellner 1991]. Since then, dozens of systems have been proposed and built, demonstrating superposition of content onto physical artifacts [Robertson 1999, Wellner 1993], the use of physical objects for tangible interaction [Fails 2002, Underkoffler 1998], in situ remote collaboration [Junuzovic 2012, Underkoffler 1999, Wellner 1993], and, more generally, interactive applications on the desk surface [Kane 2009, Kim 2014, Pinhanez 2001, Xiao 2013].

However, a notable commonality of these futuristic systems is the minimalist nature of the desk surfaces used – often lacking keyboards, mice, mugs, papers, knickknacks, and other contemporary and commonplace items (Figure 5.1). Today's desk surfaces play host to a wide variety of items of varying shapes and sizes. Moreover, these objects rarely conform to a grid or even common orientation. Desks are also constantly in flux, with items moving, stacking, appearing and disappearing. Example events include sliding a laptop out the way to make room for new work, or resting a fresh cup of coffee onto the work surface.

If digital desks do not account for these basic physical actions, applications can become brittle (e.g., does a mug placed on top of a virtual keyboard inject spurious touch input?) and inaccessible (if a book is placed over an interface, how does one retrieve it?). Further, because physical objects cannot move or change size on their own, the burden of responsiveness falls to the digital elements. Thus, digital applications must employ a variety of strategies to successfully cohabit a work surface with physical artifacts.

To help close this gap, we conducted an elicitation study with ten participants at their personal desks to understand how applications could respond to different events. We then derived a list of ten fundamental interactive behaviors that desk-bound virtual applications should exhibit to be responsive. To demonstrate these behaviors can be achieved practically and in real time, we built a proof-of-concept system called Desktopography with the necessary technical advances to support each behavior. This system had to move beyond prior work in several key ways; for example, our system requires no calibration to the world, allowing the desk scene to be in flux (i.e., there is no notion of a "background"). Further, our touch tracking approach distinguishes human "objects" (arms, hands, fingers) from other objects. This ability is critical for responsive interfaces, which must respond to user movement and input differently from changes to the physical environment (e.g., interfaces should evade from your coffee mug, but not from your hands).

Figure 5.1. Various digitally augmented desks from the academic literature.
Clockwise from top left: Digital Desk [Wellner 1993], Hardy [Hardy 2012], LuminAR [Linder 2010], IllumiShare [Junuzovic 2012], AR Lamp [Kim 2014], Everywhere Displays [Pinhanez 2001], Enhanced Desk [Koike 2001], Bonfire [Kane 2009], I/O Bulb [Underkoffler 1999]. Note the general lack of items in the interactive area. The few physical objects that do have digital interactivity are either tagged (e.g., fiducial markers), are special tangibles, or require a custom interactive table (e.g., FTIR).

5.2 Elicitation Study

To help identify useful interactive strategies, we recruited ten participants (three female, mean age 31) for a one-hour study. This study was conducted at participants' desks (see Figure 5.2 for some examples) for ecological validity. Common desk-bound items included computers, monitors and related accessories; desk phones; stacks and files of papers; carried items such as wallets, phones and keys; personal items such as memorabilia, photographs and gifts; coffee mugs; and books.

Figure 5.2. Example real-world desks of our participants.

We started with a general interview. Participants repeatedly articulated that frequently-accessed items gravitated to the front and center, while other items moved to the periphery. In general, however, the entire desk surface was in flux, with perhaps only a computer monitor and a few

peripheral items (e.g., picture frames) being stable on the time scale of months. Although our sample size was small, it reinforced our assumption (and findings from prior studies) that desks tended to be cluttered and dynamic.

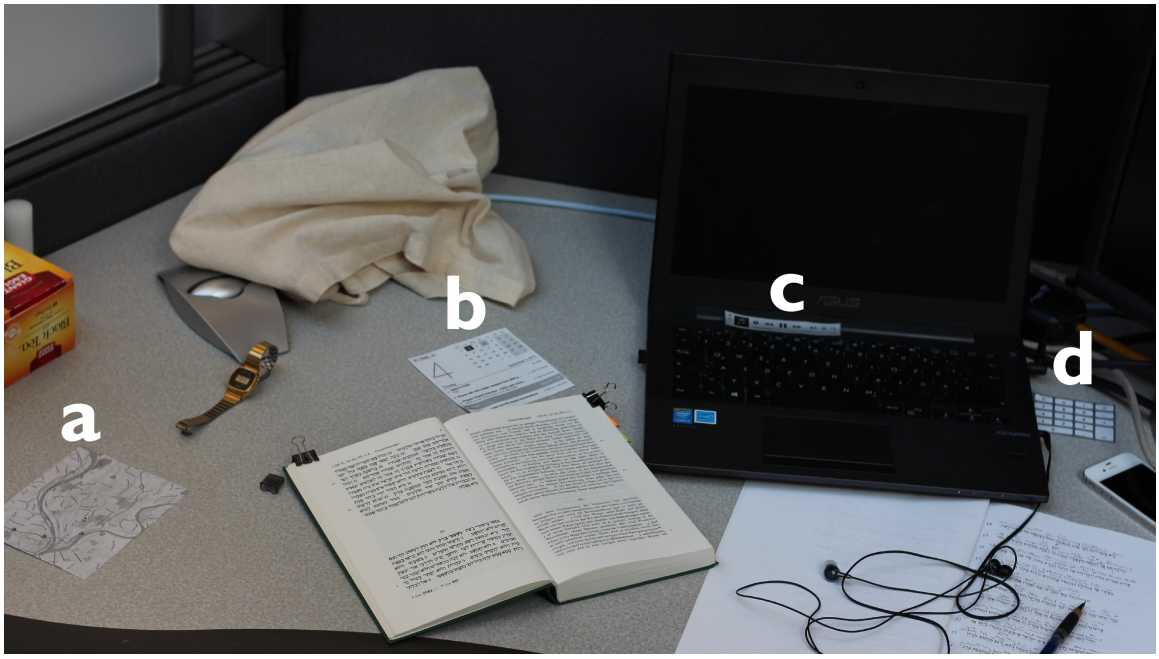

Figure 5.3. Sample arrangement of the paper prototypes in the elicitation study.
a: The map is placed to the side, for reference but not frequent use. b: The calendar is placed nearby so it can be glanced at. c: The music player is snapped to the keyboard for quick access. d: The number keypad extends the keyboard.

Next – and the primary purpose of the study – was to elicit interactive behaviors that digital applications should support. We structured this elicitation study [Nielsen 2004] around a think-aloud exercise with paper prototypes. Specifically, participants were given paper cutouts of four common applications – calendar, map, music player, and number keypad – and asked to imagine them as though they were fully interactive (Figure 5.3).

Participants then placed the (paper) applications on their work surface as they saw fit, thinking aloud about their reasoning for choosing particular locations. We also prompted them with a series of hypothetical situations, such as "what should happen to this application if you put your phone down here?", "… push the cup to the left?", "… pack your laptop in your bag?". Participants

could move, animate, fold or otherwise manipulate the paper interfaces, or explain verbally what should occur in response. The facilitator recorded comments for later analysis and affinity diagramming.

5.3 Distilling Interactive Behaviors

Following the study, we distilled our written notes and participant quotes into three, broad functional categories:

5.3.1 Application Lifecycle

***Summoning**:* Participants articulated a variety of potential strategies for summoning applications, including special gestures (e.g. “double tapping the desk”), persistent buttons (“start button”, “dashboard”, “dock”), spoken commands, and using a conventional computing device (“dragging an interface off computer desktop or mobile phone”, “keyboard shortcut”).

***Closing**:* Several methods for dismissing interfaces were proposed by participants: occluding an interface with an object (to make it “go away”), shrinking an interface substantially, moving an interface to a dedicated “trash can” area, or invoking a special gesture (e.g. “scrunching up”), throwing an object (“flinging” it), or, whimsically, miming a fireball-throwing gesture at it (“hadouken”).

Figure 5.4. Summoning an application.
a-c: The user taps twice with four fingers to bring up a launcher. d: The user moves to select the desired application. e: After lifting the fingers, the application is created.

5.3.2 Layout Control

***Repositioning**:* Participants uniformly expected to be able to reposition interfaces by holding their fingers or hand on the interface and dragging the interface around.

***Reorienting**:* Objects on desks tended to not conform to rectilinear grids; rather, objects were typically oriented according to a radial pattern centered on the user. We observed that users generally rotated interfaces to face themselves, and two users further wanted the ability to rotate interfaces to face visitors (e.g. to show a map to someone).

***Resizing**:* Most participants expected to be able to "pinch" to resize interfaces, though four participants noted that the pinch would be ambiguous (pinching content vs. resizing window). These participants suggesting resizing by using the corners, akin to desktop applications. Finally, some participants noted that they would prefer unused applications to be shrunk and placed out of the way, to be retrieved and expanded on demand.

5.3.3 Cohabitation Behaviors

***Snapping**:* Participants typically placed the number keypad or music player controls near the computer keyboard or mouse. Five participants also described wanting to "snap" or "link" these interfaces to the keyboard, letting it behave as though it was an extension of the keyboard.

***Following**:* Further, four participants expected that these snapped interfaces would automatically "follow" the movements of their associated physical object, unless the interface was manually repositioned.

***Detaching**:* Participants noted that interfaces should "unsnap" if they were covered up or manually "pulled apart" or "torn off" from the object they were snapped to.

***Evading**:* When asked to describe what interfaces should do when occluded, participants were divided. Six participants expected interfaces to "pop elsewhere" or "run away" when occluded, with two further suggesting that interfaces could shrink down to fit the remaining space, and four participants further expecting the interface to "nudge away" or "adjust" in response to partial

occlusions. Conversely, four participants noted that they would simply expect the interfaces to misbehave or ignore input if occluded.

***Collapsing**:* If the available desk space shrunk until interfaces could not find sufficient surface area, participants expected them to shrink substantially or disappear entirely.

These ten verbs form the basis of the interaction techniques we believe are necessary to support responsive interfaces in mixed physical-virtual desk contexts. Notably, while some of our verbs are taken from computer desktop windowing operations (resizing, repositioning, closing), several are designed specifically for cohabitation with physical objects (snapping, following, evading). Next, we describe our proof-of-concept system that provides all ten behaviors in real time, with no fiducial tags (or equivalent), on conventional desks, using commodity hardware.

5.4 Interactive Behavior Implementations

We built Desktopography, a desktop projection and sensing system that offers *example embodiments* of our ten interactive behaviors, which we now describe. The specific technical implementations of these features are described in detail in Section 5.5.

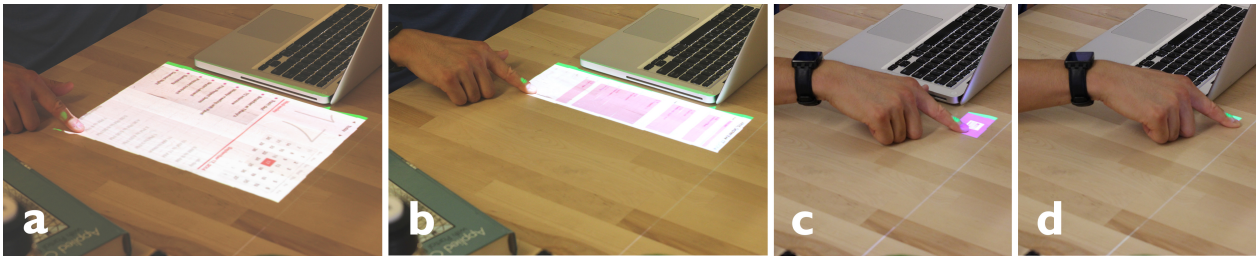

Figure 5.5. Resizing and deleting interactions.
a: The user grabs the *resize* handle in the corner. b: Some interfaces can responsively alter their layout depending on their size; here, the calendar switches from week view to day view. c: Making the interface very small causes it to iconify. d: Shrinking the interface further will *close* it.

5.4.1 Conventional Interactions

To support conventional multitouch interaction, we reserve one- and two-finger interaction for interacting with the application content itself. Users can therefore use familiar touch gestures with the application content, such as one-finger swiping and scrolling, and two-finger pinching and rotating. Thus, all the techniques for application-level manipulation use three or more fingers to avoid ambiguity. Importantly, the interactive behaviors we describe are not specific to the particular manipulation scheme employed. For example, one alternative input method could be to treat digital items more like physical objects, and use physical metaphors for interface manipulation, such as pushing, throwing and stacking (as seen in e.g., BumpTop [Agarawala 2006] and ShapeTouch [Cao 2008]).

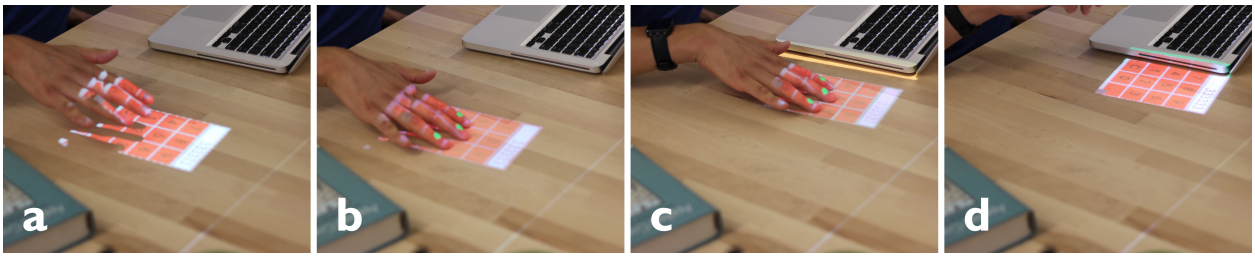

Figure 5.6. Moving and snapping interactions.
a-b: Three fingers on the interface activates *dragging*. c: Dragging the interface near an edge highlights the edge in orange. d: Upon releasing the interface, it *snaps* to the edge, which is highlighted in green.

5.4.2 Summoning

A central feature of all interactive systems is the general ability to trigger actions, a subset of which is summoning applications. Conventional GUIs typically feature static application bars, docks, menus, launchers, or other schemes, which serve as reliable and readily accessible access points. However, a physical desktop may not have sufficient space, let alone a universally available summoning point. Thus, we had to consider several mechanisms for instantiating applications.

Several mechanisms were suggested during our study: special-purpose launcher buttons or taskbars (akin to the “Start” button on Windows computers); special-purpose hand gestures (like

a double-tap on the surface, or a spreading of the fingers); or instantiation via transplantation from the computer itself (dragging an interface off of the computer screen and onto the desk). Ultimately, we decided to implement a special-purpose hand gesture for summoning, as it enabled us to immediately specify the location for an interface without requiring a computer be present, and avoided permanently devoting precious desk space for a special-purpose button. We chose to use a double, four-finger tap as the triggering gesture, as this gesture is rarely casually or accidentally invoked.

Executing the triggering gesture (tapping twice with four fingers on an unoccupied space on the desk) causes a radial menu to appear at the approximate centroid of the fingers (Figure 5.4a-c). The user can then move the fingers towards the desired application and lift their fingers to confirm the selection (Figure 5.4d). Alternately, the users can simply lift the fingers without moving to cancel the menu. In a complete implementation, one of the menu options would permit voice or text input to search for less frequently used applications not listed on the radial menu. Confirming the selection of an application causes the application to appear at the center of the menu (Figure 5.4e). The user can then reposition and resize the application.

5.4.3 Resizing

During our study, every participant suggested using a pinch gesture to resize the interface. However, some noted that this would be ambiguous with respect to the content (which is not usually an issue since the pinch gesture is typically used on mobile devices that only show one application at a time). Further, a pinch gesture does not naturally permit the aspect ratio to be adjusted.

We therefore borrow a familiar mechanism for manual resizing – a draggable 1.5 cm grey box in the lower-right corner of the application (Figure 5.5a). Because applications can (and must) exist at a variety of sizes on desktops, applications ideally implement responsive layout schemes [Zeidler 2013] (Figure 5.5a,b). If applications are resized below a reasonable limit (e.g. less than 4 cm in either direction), applications “iconify”, displaying only an icon of the application and the resize box (Figure 5.5c). This permits applications to be stored on the desk without taking up substantial space.

5.4.4 Deleting

We considered several possible deletion schemes, some suggested by our study. These included special-purpose gestures (e.g. “scrunching up” the interface or “flicking” the interface away), special drag targets (e.g. a “trash can”), or explicit close buttons (like those found on computer desktop windows). Ultimately, we chose to simply extend resizing to provide deletion capabilities: applications can be deleted simply by resizing them below 1.5 cm in either direction (Figure 5.5d). This approach is simple and requires no additional buttons or gestures.

5.4.5 Repositioning and Reorienting

In order to reposition applications manually, we implemented a “drag” strategy. Users can press three fingers to the application and drag as desired, or rotate the hand to reorient the application (Figure 5.6a,b). For iconified applications, which are small, a single finger is used for dragging (as the content is hidden, there is no interaction ambiguity).

5.4.6 Snapping

Users can snap applications to the topological discontinuities (i.e., “edges”) of physical objects on the desk, such as the sides of laptops, books, keyboard, stacks of paper, edges of work surfaces and so on. These snap candidates are detected with the edge finding algorithm described later.

To snap an interface to an edge, users drag the interface towards the target (Figure 5.6b). The nearest object edge to the interface (up to 50 mm away) will be highlighted in yellow (Figure 5.6c). When an object edge is highlighted, releasing the drag will cause the interface to snap to the highlighted edge, aligning the interface edge to the nearest object edge by repositioning and reorienting the interface as necessary. After successfully snapping, the snapped interface edge will be highlighted to denote the snapped state (Figure 5.6d).

5.4.7 Following

As noted during our study, users expect snapped interfaces to follow the objects they are snapped to. In our system, this "following" behavior is implemented by updating the object edges ten times per second (alongside the desk topography), and repositioning snapped interfaces to the new edge nearest to the previous edge (in both position and rotation angle). This causes interfaces to simply follow the objects naturally (Figure 5.7a-c). To avoid jitter due to noise in the edge measurements, interfaces only follow if the edge moves by more than 4 mm.

5.4.8 Detaching

Users may also want to disable snapping or following behaviors, in which case they can detach the interface. This can be achieved by either dragging the interface off the object (Figure 5.7d), or by moving the object rapidly away from the interface (i.e., "tearing" it off). Interfaces will detach if the edge moves more than 40 mm away in a single frame update, which corresponds to a "detach velocity" of 400mm/s.

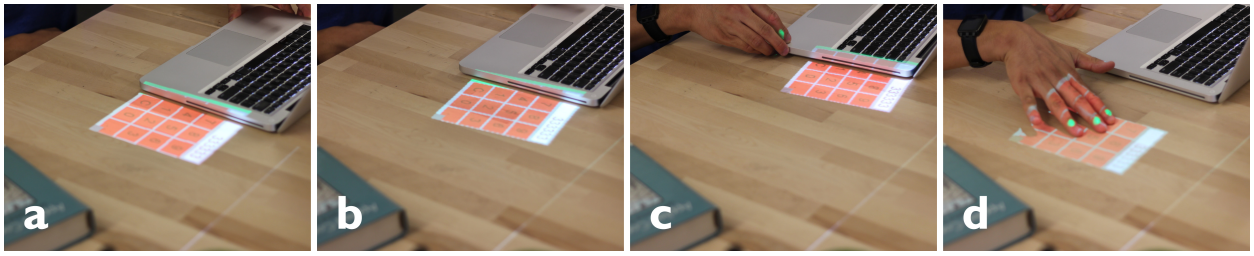

Figure 5.7. Following and detaching interactions.
a-c: Once a virtual interface is *snapped* to an object, it should *follow* if the object is moved. d: The snapped interface can be *detached* by dragging it off.

5.4.9 Evading

Interfaces that are suddenly occluded will move out of the way and seek another empty space to appear on (Figure 5.8b-c). This could happen if, e.g. users rearrange their desk, or shuffle items around. When this occurs, the next topography update will detect a sudden increase in the interface's "roughness", and will include the interface in the subsequent layout pass. This relocates

the interface to a nearby open space on the desk (any relatively-flat area with low roughness), while avoiding other interfaces. Multiple interfaces may participate in optimization simultaneously.

5.4.10 Collapsing

Our optimization-based layout algorithm (described in Section 5.5.6) will also automatically shrink an interface if the available space is insufficient to place a displaced interface. This permits an interface to "squeeze" into a space where it can be displayed (Figure 5.8c). If the available space is very limited, the interface may simply iconify; the user can then choose to clear off space and re-expand the interface as desired.

Figure 5.8. Evading and collapsing interactions.
a: An interface is positioned somewhere. b: The interface is occluded by an object. c: When the user moves his hands away, the interface *evades* the occlusion, and also *collapses* to fit available space.

5.5 Technical Implementation

Achieving our ten interactive behaviors required combining features described in many disparate research efforts into a single novel system, as no prior system has the necessary feature set. More specifically, our system provides rectified projection onto irregular surfaces. To support our co-habitation behaviors, it was necessary to perform object tracking and edge finding in real-time. We also employ an optimization-based layout scheme to automatically find location candidates

for evading interfaces. For finger touch tracking, we extend the approach described in OmniTouch [Harrison 2011], which allows fingers to be disambiguated from other objects. We now describe these processes in greater detail.

Figure 5.9. Our proof of concept projector-camera system fitted into a lampshade.

5.5.1 Hardware

Our hardware prototype consists of a paired depth camera (Asus Xtion) and pocket projector (700-lumen Optoma ML750) as seen in Figure 5.9, right, a configuration similar to that found in [Harrison 2011] and WorldKit (Chapter 4). The hardware was enclosed in a ceiling-hung lampshade fixture (Figure 5.9, left) suspended 90 cm above a desk surface. At this distance, the projected image is 62x39 cm in size. The camera and projector are rigidly affixed together and calibrated with respect to each other (once, e.g., "at the factory") using the seven-point calibration routine described in Section 4.3.2. The hardware is designed to be easily portable and functional on any desk, with no calibration of the desk surface required.

The depth camera produces 320x240 pixel depth images at 30 frames per second. Natural variation due to noise is 4-7 mm at a distance of 90 cm [Khoshelham 2012]. Our software was developed using the OpenFrameworks C++ library, and runs on a 2011 Macbook Pro. The complete

implementation requires less than 15% of the CPU in full operation. It runs touch tracking at camera frame rate (30 fps), geometry updates at 10 fps, and interface draw/update logic at projector frame rate (60 fps).

5.5.2 Disambiguating Object and Human Movement

In order to enable interaction on the ever-changing desk surface, our system needs to readily and accurately distinguish fingers and hands from other objects. For example, to implement the *evade* behavior, we need to have interfaces move away from objects that are placed on top, but not move away from hands or fingers that appear over the interface. Additionally, interfaces must be able to respond to hand and finger input without being affected by other objects that may be moving in the scene.

We therefore detect human arms (which are characteristically long and cylindrical in shape) in addition to fingers during our touch tracking procedure, and use the arm and finger data to label fingers, hands and arms. These human "objects" are then excluded from most topographical considerations. Since the field-of-view of the depth sensor is approximately twice as large as the projection area in our system, we can guarantee that fingers appearing within the projection area will always be accompanied by visible arms in the larger depth image.

5.5.3 Touch Tracking

We did not want to augment our desks with additional sensors (since the system was intended to operate over any unmodified desk surface), and therefore opted to use a depth camera to track touches. Further, while background subtraction and in-air touch tracking enable reliable hand tracking (by identifying and removing the background), we determined that such an approach would be infeasible in a desk environment where objects are liable to move.

Therefore, we decided to extend the "cylinder-finding" finger tracking algorithm proposed in OmniTouch [Harrison 2011]. Specifically, we extract and combine both horizontal and vertical cylinder slices (Figure 5.10b, red and blue lines), enabling the detection of fingers (Figure 5.10b, white)

oriented in any direction (Figure 5.11). We additionally reject finger-like objects (e.g., pens or markers) by detecting cylindrical arm slices and requiring valid fingers to be connected to arms (see previous section). Importantly, this algorithm requires no calibration, no background subtraction, and makes no decisions based on (e.g., potentially lighting dependent) color information. Touch tracking latency is roughly 40 ms (limited by the refresh rate of the depth camera).

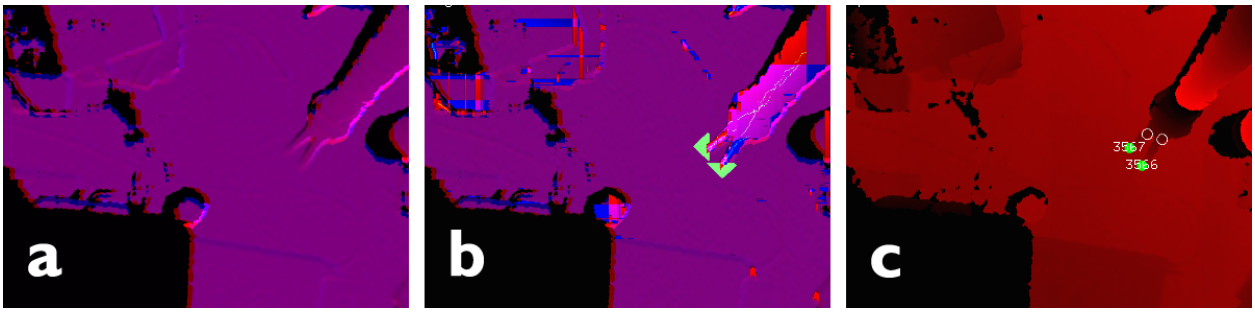

Figure 5.10. Touch tracking steps in Desktopography.
a: Depth gradients (red=dx, blue=dy). b: Unfiltered cylinder slices (red, blue), slices connected into cylinders (white lines) and touch-detection flood-fill (green). c: Detected touches overlaid on depth image. White circles represent the base of the finger.

Figure 5.11. Desktopography tracking fingertips on the desk.
The touch tracking algorithm can find finger tips even when the hand is flat against the surface.

5.5.4 Handling Irregular Surfaces

The “desk topography” is sensed by converting each depth pixel in the projection area into a height value, which is used to construct a mesh. The topography mesh is then re-rendered from an orthographic view, and the depth component of this rendering is extracted into a height map, which converts desktop coordinates into height values above the desk surface. This height map is used to identify flat areas suitable for placing interfaces.

The mesh is also used for projection mapping. The desktop imagery is warped (by UV-mapping the desktop graphics onto the mesh) such that the resulting projected output appears “orthographically correct” no matter how tall or irregularly-shaped the objects atop the desk are. This ensures that projections on flat objects maintain a consistent scale (so that e.g. one virtual desktop millimeter is always one millimeter on a flat surface regardless of surface height).

The system will attempt to update the desk topography ten times per second, but will skip the update if the user is interacting with applications or fingers are detected over the top of an interface. In this way, the system avoids occlusion and interference effects from the user’s hands while maintaining an acceptable update frequency. After completing a topography update, the system will search for edges in the new topography, and then initiate an interface optimization pass to support evading and following behaviors.

5.5.5 Edge Finding

To support our snapping and following behaviors, our system required the ability to detect edges of physical objects. The process starts by filtering the desk height map with a Canny edge filter [Canny 1986] to extract depth discontinuities. The parameters were tuned to extract discontinuities of at least 7 mm without excessive noise (the smallest possible difference given the noise level of the depth sensor). The binary edge image is then fed to OpenCV’s contour-finding routine, which connects adjacent edge pixels into contiguous contours.

Contours are then filtered to remove artifacts (e.g. duplicated and doubled-up contour paths) and short noise-induced contours. We then extract segments from the contours by walking over

the pixels in each contour and outputting a segment whenever the contour abruptly changes direction (by more than 45° in an 8-pixel window). Short segments (less than 20 mm long) are deleted, and the remaining segments are converted to straight lines by using linear interpolation.

5.5.6 Optimization-Based Layout

To support evading and collapsing behaviors, the system must be able to automatically identify open spaces on the surface, and to decide whether the available space is sufficient. However, the irregular and complex topology of desks yields innumerable corner cases that make building a rule-based or heuristically driven layout approach unwieldy and brittle. Thus, we use an optimization-based layout engine to support evading and collapsing interfaces.

Optimization has long been used for spatial problems such as graph visualization [Di Battista 1998] and VLSI layout [Cong 1996] and more recently for problems in perceptually optimized display generation [Agrawala 2001, Lee 2005]. More relevantly, optimization based techniques have also been employed for automatic generation of user interface layouts (see e.g., [Bodart 1994, Fogarty 2003, Gajos 2008, Sears 1993]). As far as we are aware, our work is the first to use optimization-based layout for on-world interactive purposes.

Our approach defines a certain “cost” to every interface configuration, depending on the desk topography and interface positions. It then aims to minimize the total cost of the configuration, subject to a number of “penalties” which guide the process away from undesirable behavior (e.g. to avoid large jumps in position). Furthermore, the optimization-based layout engine can be used, for example, to automatically position new interface elements (if they are summoned e.g. using a voice command alone) or to support recalling a set of interfaces onto a changed desk layout.

The layout engine uses the topographical height map to determine the surface *roughness* as follows. The roughness at each desk point is calculated as the standard deviation of the height map in a 41x41 millimeter square window centered on the desk point. This is efficiently computed by using sliding windows over summed-area tables [Crow 1984]. The “roughness” of a particular interface area is the average roughness across each point in the area; this is efficiently calculated by using a polygon scanline approach coupled with a summed-area table of the roughness map.

Every time the topography is updated (up to ten times a second), the roughness map is updated, the interface positions are adjusted to follow any snapped edge boundaries (to support following behaviors), and the optimization algorithm is executed. The optimizer generates and evaluates a number of candidate interface configurations, starting with the initial configuration. Interfaces which were recently positioned by hand, or whose roughness has not significantly increased since the most recent optimization run are excluded from optimization.

The cost of a particular configuration is computed from the total sum of the interface area roughness values and an assortment of penalties (cost increases) to discourage particular behaviors. A penalty is imposed for moving an object at all, which biases the algorithm towards keeping objects still. Another penalty is applied based on the distance moved and any applicable size change, which biases the algorithm towards small movements (preserving spatial locality). Finally, a large penalty is imposed for any overlap between interface object areas, to prevent the algorithm from optimizing two interfaces onto each other.

The optimization itself is performed using a simulated annealing algorithm. The simulated annealing algorithm maintains a “temperature” parameter that decreases at each iteration of the algorithm. In each iteration, each interface object in the current best configuration is “mutated” to generate a list of new candidate placements; these mutations can consist of moving, rotating or resizing objects. The expected magnitude of each mutation is controlled by the “temperature” parameter, with higher temperatures resulting in more extreme mutations (e.g. longer distance movements). Candidates are combined into new configurations and tested for acceptance according to the current timestep’s “temperature” parameter (which controls the simulated annealing algorithm’s willingness to accept new “best” configurations). The configurations are generated exhaustively starting with the candidates having the lowest roughness values. If this process yields too many combinations, the algorithm switches to generating configurations randomly to avoid combinatorial explosion.

The optimization algorithm terminates after reaching a predefined number of iterations, or if it runs for more than 60ms. This bounds the optimization algorithm and prevents it from running

endlessly. The best configuration found so far is then returned (which may be the same as the initial configuration, if no alternative was superior).

5.6 Conclusion

I have presented a set of ten interaction behaviors for desk interfaces designed to facilitate digital interfaces co-existing with and responding to physical objects. These behaviors were derived from an elicitation study using paper prototypes. I also described the implementation of a proof-of-concept projector-camera system, designed with a unique set of technical features necessary to implement our behaviors. In general, the interaction behaviors presented in this paper serve as a basis for a more complete, modern digital desk system, in which the work surface and its contents are not replaced with digital equivalents, but augmented to add new capabilities. While similar ideas have been explored in prior work, we draw them together in a holistic and grounded manner, and additionally demonstrate technical feasibility.

CHAPTER 6. REFINING ON-WORLD TOUCH INPUT

6.1 Summary

In this chapter, I describe my work on DIRECT (Depth and IR Enhanced Contact Tracking), a new touch-tracking approach that merges depth and infrared image data (from a single sensor) to provide significantly enhanced finger tracking over prior methods (Figure 6.1 and Figure 6.2). Infrared imagery provides precise finger boundaries, while depth imagery provides precise contact detection. Additionally, the use of infrared data allows the system to more robustly reject tracking errors arising from noisy depth data. This approach allows DIRECT to provide touch tracking precision to within a single pixel in the depth image, and overall, finger tracking accuracy approaching that of conventional touch-screens. Additionally, our approach makes no assumptions about user position or orientation (in contrast to all prior systems, which require *a priori* knowledge of finger orientation to correct for undetected fingertips), nor does it require prior calibration or background capture.

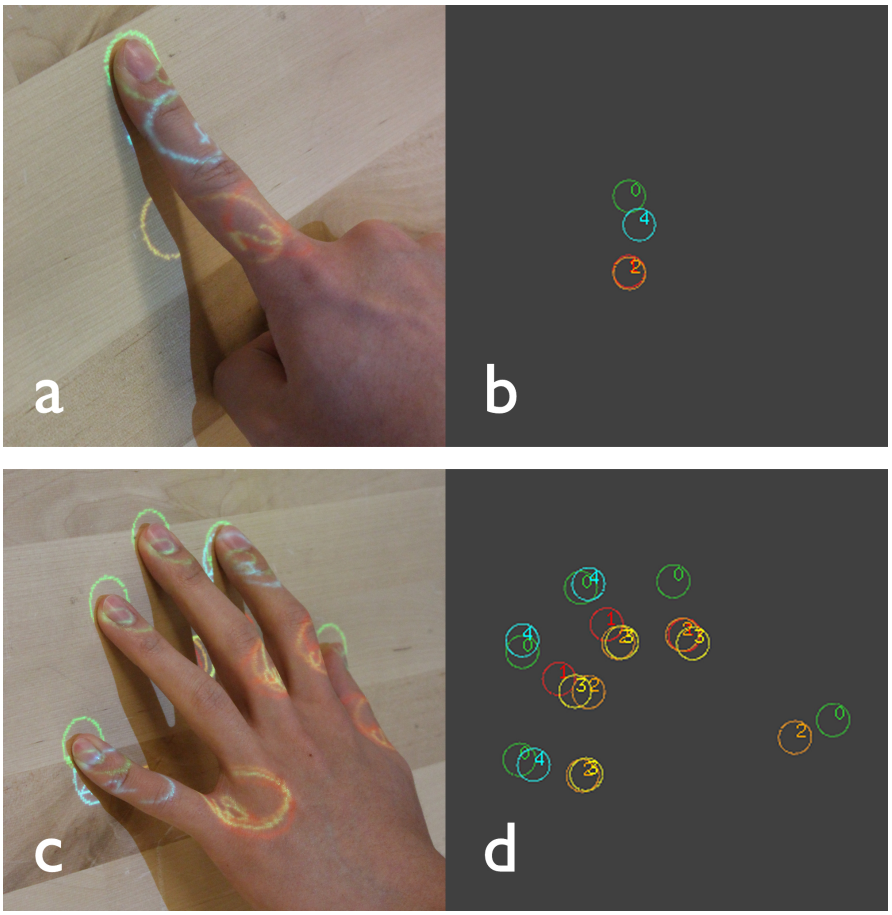

Figure 6.1. Comparison of depth-camera-based touch tracking methods.
Our method (DIRECT) is shown in green and labeled 0. Comparison methods are single-frame model (1/red), maximum distance model [Wilson 2010a] (2/orange), statistical model [Izadi 2011, Xiao 2013] (3/yellow) and slice finding [Harrison 2011] (4/cyan).

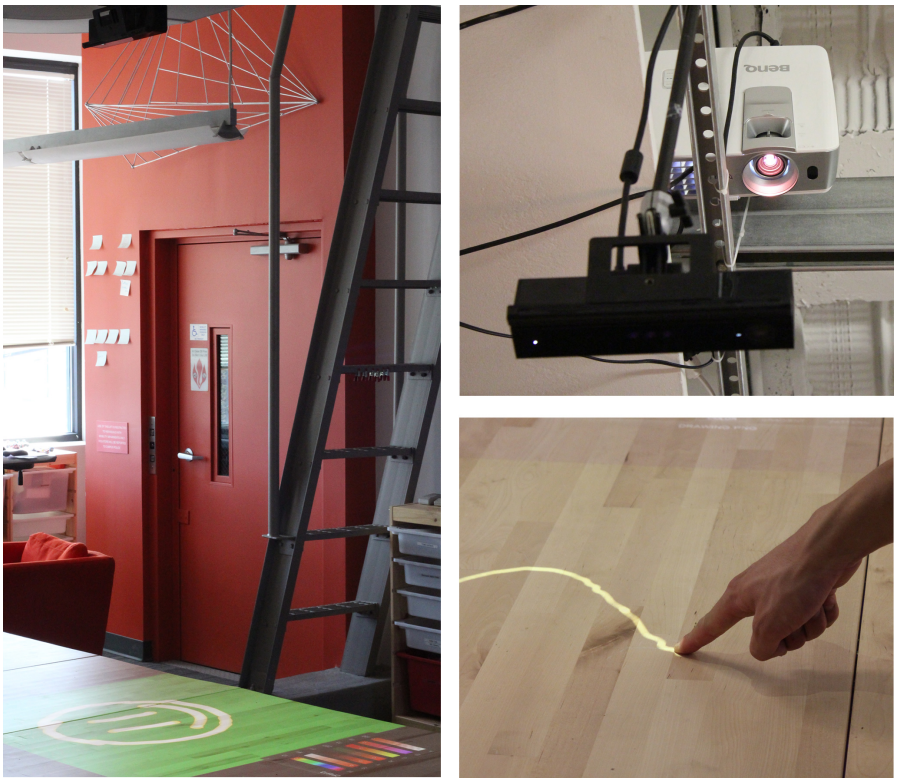

Figure 6.2. DIRECT system setup.
Left: the Kinect depth camera is mounted 1.6 m from the table surface. Top right: the projector and Kinect are rigidly mounted and calibrated to each other (but not the surface). Bottom right: The table surface functions as a touchscreen.

We describe in detail the technical implementation of DIRECT and discuss its capabilities and unique characteristics relative to other touch tracking approaches. We further contribute a multi-technique comparison study – the first evaluation of its type – where we compare four previously published methods against one another, as well as against DIRECT. DIRECT readily outperforms these prior techniques with respect to viable distance, touch precision, touch stability, multi-finger segmentation, and touch detection false positives and false negatives.

6.2 Implementation

We implemented our touch tracking algorithm and comparison techniques in C++ on a 2.66 GHz 3-core Windows PC, with a Kinect for Windows 2 providing the depth and infrared imagery. The

Kinect 2 is a time-of-flight depth camera, which uses active infrared illumination to determine the distances to objects in the scene. It provides 512x424 pixel depth and infrared images at 30 frames per second. A BenQ W1070 projector with a resolution of 1920x1080 is also mounted above our test surface (a wooden table) to provide visual feedback.

The Kinect 2 is mounted 1.60 meters above a large table surface, and the projector is 2.35 meters above the surface (Figure 6.2). At the horizontal edges of the Kinect's field of view, the table surface is 2.0 meters from the Kinect. Both the projector and Kinect are securely mounted to the ceiling, and were calibrated to each other using multiple views of a planar calibration target.

The present configuration allows the projector to project a 1.0×2.0 meter image onto the table surface, with the Kinect capable of sensing objects across the entire projected area. At this distance, each projected pixel is 1.0 mm square, and each Kinect depth pixel is 4.4 mm square at the table surface. Thus, even with this second-generation sensor, a typical fingertip resting on the table is less than 3 depth image pixels wide, underscoring the sensing challenge.

Our approach combines background modeling and anthropometric modeling approaches. More specifically, DIRECT models both the background *and* the user's arms, hands, and fingers using separate processes. Our processing pipeline is carefully optimized so that it runs at camera frame rate (30 FPS) using a single core of the PC.

The touch-tracking pipeline is illustrated in Figure 6.3 and Figure 6.4. Figure 6.3 shows a hand laid flat on the table, which is challenging for depth-based touch tracking approaches because the fingertips generally fuse with the background due to sensor imprecision and noise (Figure 6.3a). In Figure 6.3d, we show that DIRECT can still segment the fingers (labeled in different colors) right down to the fingertip. Figure 6.4 shows another challenging case - a single extended finger raised 60°. This is problematic because there are very few depth pixels available for the fingertip itself. As before, DIRECT is able to segment the crucial fingertip.

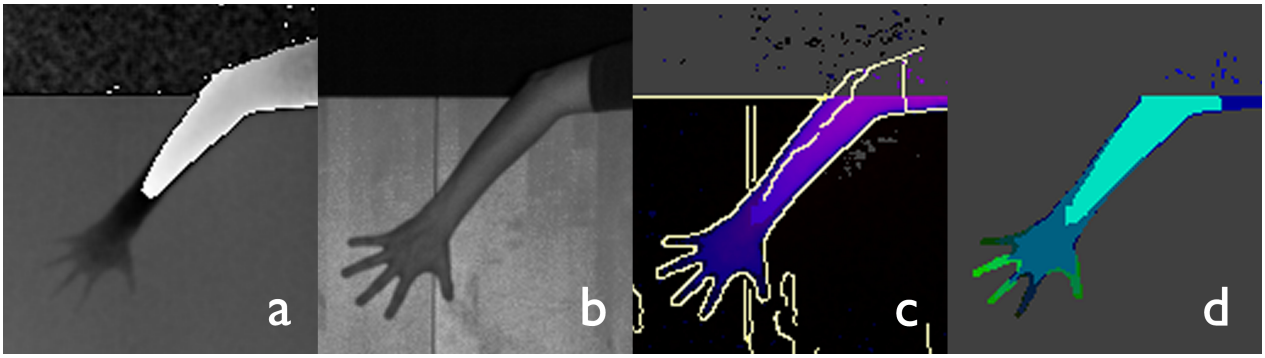

Figure 6.3. Touch tracking process for five fingers laid flat on the table.
(a) depth image, (b) infrared image, (c) infrared edges overlaid on z-score map, (d) segmentation result. In (d), *arm* pixels are cyan, *hand* pixels are blue-green, and *finger* pixels are combined with *fingertip* pixels and shown in various shades of green.

6.2.1 Background Modeling

Our system uses a statistical model of the background, inspired in part by the implementation in WorldKit (Chapter 4). At every pixel, we maintain a rolling window of five seconds of depth data and compute the mean and standard deviation. This model allows us to establish both a highly accurate mean depth background, as well as a noise profile at every pixel in the scene.

With these rolling windows, we support *dynamic* updating of the background model. If the standard deviation exceeds a certain depth-dependent threshold (accounting for higher average noise further from the sensor), the pixel is temporarily "stunned" and its background model mean and standard deviation will be held constant until the moving average drops below the threshold. This approach accurately tracks long-term changes in the environment (*e.g.*, objects moved around), while ignoring short-term changes (*e.g.*, actively-interacting hands and fingers). In our present implementation, the background is updated in a separate thread, running at 15 fps to avoid excessively frequent updates. This type of dynamic background updating is crucial for long-running systems to deal with shifts in the environment (*e.g.*, movement of objects residing on the surface), yet most existing systems (*e.g.*, [Wilson 2010a, Xiao 2013]) use only a single static background model captured during initial setup. We note that highly stationary hands and fingers

could be “integrated into the background” with this approach, though we observed that, in practice, users rarely stay stationary for several seconds over top of active touch interfaces.

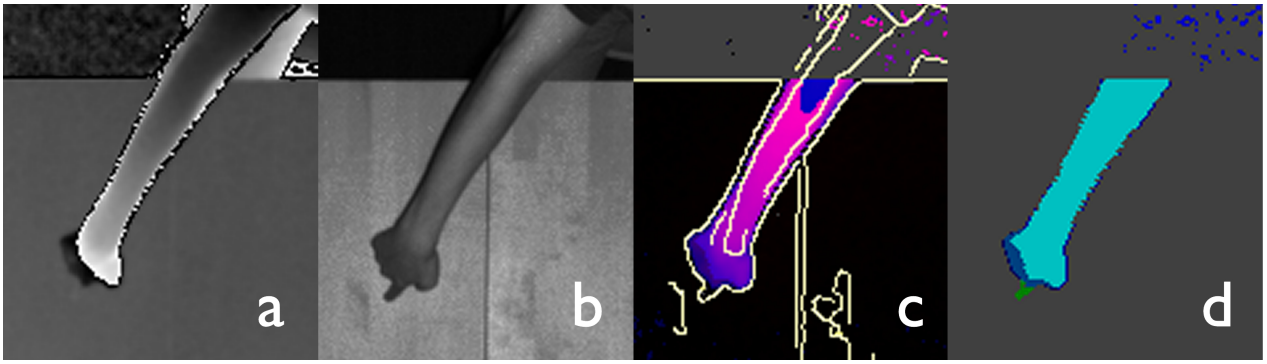

Figure 6.4. Touch tracking process for a finger angled at 60° vertically.
(a) depth, (b) infrared, (c) edges and depth map z-scores, and (d) filled blobs. Refer to Figure 6.3 for a full color key.

6.2.2 Infrared Edge Detection

We use the infrared image primarily to detect the boundary between the fingertip and the surrounding surface. As such, our first step is to detect edges in the infrared image. The Kinect 2’s infrared image comes from the same sensor as the depth data, and thus it is precisely registered to the depth image. We use a Canny edge filter [Canny 1986] to locate candidate edge pixels in the image (7x7 Sobel filter, with hysteresis thresholds of 4000 and 8000), with results shown in Figure 6.5. Note the parameters are not specific to the operating depth or objects in the scene.

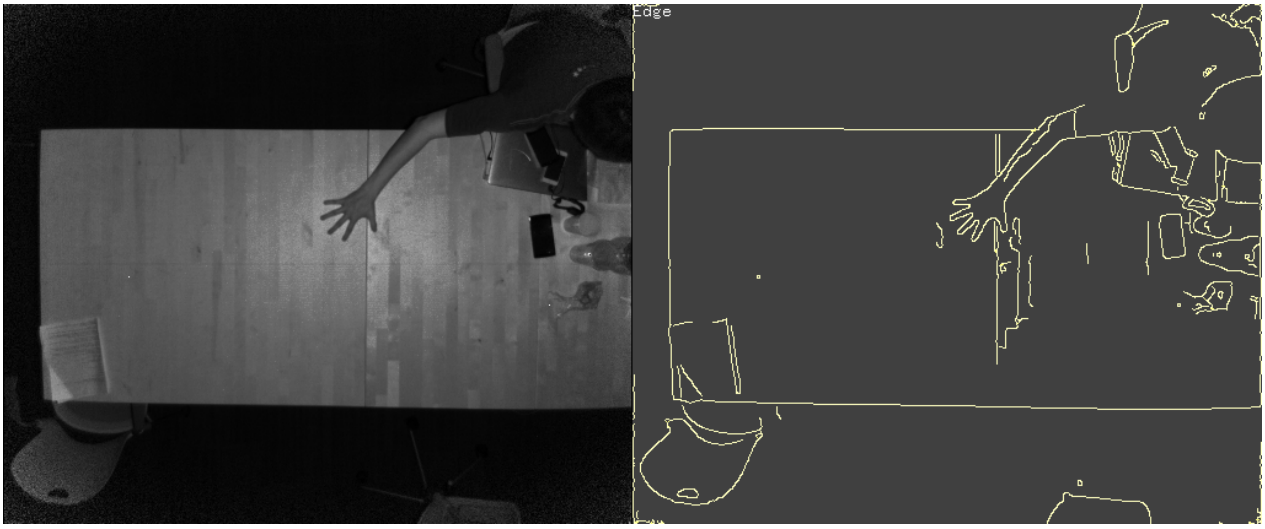

Figure 6.5. Canny edge detection on the IR image.
Left: Kinect IR image. Right: Edge map.

After running the Canny edge filter, some edges may have gaps. These can occur due to *e.g.,* multiple edges meeting. A common gap-filling technique, image dilation followed by erosion, is inappropriate for our case due to the small size of the fingertip. Instead, we employ an *edge-linking* [Maeda 1998] algorithm across the edge map, which walks along Canny edge boundaries and bridges one-pixel breaks between neighboring edges. After applying this algorithm, finger-tips and hands are usually fully enclosed (Figure 6.3c and Figure 6.4c, pale yellow lines).

Surfaces with a very similar infrared albedo to skin could cause issues for edge finding. However, anecdotally, we have found that the shadows cast by the arm and hand (illuminated by the active IR emitter found in many depth cameras), even near the fingertip, helps to increase contrast (as seen in Figure 6.3b and Figure 6.4b).

6.2.3 Iterative Flood-Fill Segmentation

Our touch tracking pipeline consists of a sequence of iterative flood fills, each responsible for a different pixel type: arm filling, hand filling, finger filling, and fingertip filling. When each flood fill completes, it triggers the next fill in the sequence, starting from pixels on its boundary. Critically, each flooded area is *linked* to the parent area from which it was seeded (*e.g.* the finger fill starts

from the hand that it is attached to), forming a hierarchy of filled objects that match their respective anthropometric requirements derived from [Eastman Kodak 2003, Haley 1988, NASA 1995, White 1980]. For a finger to be successfully segmented (and passed as input to an interface), a complete hierarchy must exist – a process that robustly rejects finger-like objects that are not connected to hands, arms and so on (*e.g.*, a whiteboard marker laying on a tabletop).

Arm Stage

The first stage is labeling arm pixels, and merging them into connected *arm blobs* (Figure 6.3d and Figure 6.4d, light blue). Arm pixels are defined as pixels that are at least 5 cm closer to the sensor than the background mean, *i.e.* pixels that are at least 5 cm above the surface. This high threshold is chosen both to unambiguously distinguish human activity from background noise (standard deviations at the edge of the depth map can reach ~1.5 cm, so 5 cm is more than 3 standard deviations away), and to detect human forearms even when laid *totally* flat on the table (6.3 cm is the 2.5th percentile diameter of a human forearm).

Hand Stage

Then, for all arm blobs, our algorithm flood fills downwards towards the hand pixels, defined as pixels that are at least 12 mm from the surface. This threshold was chosen to segment individual fingers apart from the hand (12 mm is the 2.5th percentile thickness of a human finger, allowing us to detect even small fingers lying flat on a table). During this step, the fill is constrained to avoid pixels with high depth variance in their local neighborhood. This constraint ensures that this flood fill does not simply fill into noisy pixels surrounding the arm. The result is a *hand blob* attached to the parent arm blob (Figure 6.3d and Figure 6.4d, dark blue blob). If multiple hand blobs are found, only the largest is taken (to avoid noise-induced "phantom hands").

Finger Stage

In the third stage, the algorithm fills from the hand blob into finger pixels, which are defined as pixels that are at least one standard deviation above from the surface mean. These pixels are above noise, but are otherwise very close to the surface. Because of this, we constrain the fill to stay within the boundaries derived from the infrared edge map. In the event of a hole in the edge

map, the fill will stop at below-noise pixels and thus will not fill too far. This process produces a number of *finger blobs* attached to the hand blob, and the point at which the finger attaches to the hand is called the *finger base*.

Fingertip Stage

Finally, for each finger blob, the algorithm fills further into the below-noise pixels (fingertip pixels). At this point, the depth map is not used (as fingers generally merge with noise), and only the infrared edge constrains the fill. A vast majority of frames fill successfully, however occasionally, a gap in the edge map will cause the flood to escape outside the finger. To mitigate this, our flood fill stops and flags an *overfill error* if the fill extends more than 15 cm from the finger base. This value allows for both the longest human fingers (mean length 8.6 cm, SD 0.5) and is 2 standard deviations above the mean palm center to fingertip length (mean 12.9 cm, SD 0.8), which is the worst-case scenario if the hand fill stage was only partially successful in flooding into the hand. Additionally, this permits the use of grasped pens, markers or styluses used as pointing devices, while still rejecting out-of-control flood fills that spill out onto the background.

If an overfill condition is detected, DIRECT deletes the flooded fingertip pixels and returns a failure indication. This allows the system to gracefully fall back to depth-only touch tracking in the event that the IR image is unusable for any reason (*e.g.* because there are holes in the edge image). Crucially, this enables the algorithm to work in cluttered or complex environments when the edge image may be damaged or unusable, albeit with reduced performance. Otherwise, if no overfill occurs, the fingertip pixels are added to the parent finger blob. The resulting finger blobs are seen in Figure 6.3d and Figure 6.4d (varying shades of green).

6.2.4 Touch Point Extraction

During both the finger fill and tip fill, we record the distance of each finger pixel to the finger base. For each detected finger, we simply place the fingertip at the pixel with the highest such distance. We found that this pixel correlates extremely well with the fingertip's actual location, and furthermore that this point is stable enough that touch position smoothing is unnecessary.

If the tip filling failed due to an over-fill, the fingertip's position can be estimated using forward projection, by using the arm and hand positions to determine the orientation of the finger. However, the resulting estimate will be substantially noisier, a fact which can be conveyed to a higher-level filtering or smoothing algorithm.

6.2.5 Touch Contact Detection

To detect if the fingertip is in contact with the surface beneath it (*i.e.* to distinguish *hover* from *contact*), we examine the 5x5 neighborhood around the tip pixel. If any pixel is more than 1 cm from the background, we mark the finger as hovering, otherwise the finger is marked as touching the surface. We then apply hysteresis to avoid rapid changes in touch state. Although this touch detection approach is simplistic, it is surprisingly robust; we attribute this to the high precision of our fingertip tracking.

6.2.6 Touch Tracking Properties

The DIRECT approach merges aspects of optical tracking, background modeling and anthropometric finger modeling approaches, and therefore exhibits some unique properties relative to other methods. Compared to depth-only methods, the touch points detected by DIRECT are more stable, as the infrared estimation provides pixel-accurate detection of the fingertip. Consequently, DIRECT requires no temporal touch smoothing to work well, allowing it to have very low latency (15 ms average latency from input depth data to output touch point) without sacrificing accuracy.

While DIRECT uses ideas from finger modeling, it does not directly model the shape of fingers, nor does it make assumptions about the shape of the touch contact area. Therefore, DIRECT is capable of tracking touches when the fingers assume unusual configurations, such as holding all fingers together, performing gestural touches, or holding objects such as pens and styli.

Lastly, DIRECT provides some additional information about the hand and fingers beyond just the touch point. Specifically, it also provides the orientation of the finger (the vector from the finger base to fingertip), the pose of the hand, and the arm associated with each touch. This metadata

could be used to implement interaction techniques beyond simple multitouch, *e.g.* using the finger angles for rotational input [Wang 2009], or to manipulate 3D objects [Xiao 2015], and using the arm data to enable bimanual interactions.

6.3 Comparative Techniques

To compare the performance of our technique, we implemented four representative depth-camera-based touch tracking methods from the literature (see also Related Work). An example frame of tracking output from these implementations can be seen in Figure 6.1.

Of note, many of our comparison techniques were originally developed using the predecessor of the Kinect 2 we use (the "Kinect for Windows", henceforth called Kinect 1 for simplicity). The Kinect 1 sensor uses structured light, which projects an infrared speckle pattern, as opposed to the time-of-flight method used in the Kinect 2. The speckle pattern renders the infrared image virtually unusable for tracking, precluding DIRECT-style sensing. The Kinect 1 features both lower depth image pixel resolution and lower depth resolution (~4 mm per depth unit at a distance of 2 meters) than the Kinect 2. We have thus adjusted and tuned the comparison techniques to work as well as possible with the Kinect 2 sensor.

We also did not use any calibration with these techniques. Touch tracking systems often employ a calibration stage where a user touches several points to establish a mapping between the sensor and known physical points. However, if the user calibrates without significantly changing their orientation, this calibration can mask certain orientation-dependent biases (as we show in our results) and make the system dependent on the user and finger orientation. Hence, in our study, we apply only a global calibration between the depth camera and the projector, and do not calibrate using detected finger positions.

All of our comparison techniques have a certain number of "magic numbers" that must be tuned for correct operation of the system. Furthermore, precise adjustments of these numbers can be used to trade off between *e.g.* touch accuracy and false touch detection. For the study, we tuned these numbers to keep false positives acceptably low, as these are especially damaging to the

user experience. Specifically, we aimed to reduce false positives to less than one false-positive per five seconds within our target area (2 m^2) when no fingers are present. Although this is still a high rate of false positives, we found that attempting to further reduce this rate led to unacceptable insensitivity in two of our comparison techniques. We tuned each comparison technique independently, communicating with the system's authors when necessary to best reproduce the technique.

Tests on all techniques were done without smoothing of the touch data. Smoothing could be applied to the output of any of these approaches and might increase accuracy. However, this would come at the cost of increased latency and would tend to hide the merits of the technique itself.

6.3.1 Single-Frame Background Model

Our first comparison technique is a common, naïve technique often used for simple touch tracking. It uses a background model consisting of a single captured depth frame. Candidate touch pixels are simply those that lie between a minimum and maximum distance threshold from the background (in our implementation, between 7 and 15 mm from the surface).

For all background model implementations, we apply a low-pass boxcar filter followed by thresholding. A connected-components pass then extracts touch blobs, following the implementation in [Wilson 2010a]. Although most naïve implementations do not perform this filtering, we found it necessary to avoid excessive noise in the contact tracking.

6.3.2 Maximum Distance Background Model

The second comparison technique models the background using the *maximum* depth value seen over a window of time. This effectively implements a conservative noise model of the background. Like the single-frame approach, the captured depth is then processed through a low-pass filter and thresholded, followed by a connected-components pass to segment finger touches.

Wilson [Wilson 2010a] implements a variation on this technique [pers. comm.], using a histogram to choose *e.g.*, the 90th percentile depth value at each pixel to eliminate outliers in the original Kinect data. With the Kinect 2, we did not observe such outliers, and so our maximum distance model is effectively the same as the histogram method. Wilson tested their system using a Kinect at a maximum distance of 1.5 metres from a table. At that distance, Wilson reported anecdotally that the positional tracking error was about 1.5 cm.

6.3.3 Statistical Background Model

The final background modeling method uses a statistical approach. This implementation uses the same mean and standard deviation calculation as the DIRECT method. New depth frames are converted into Z-scores based on their differences from the background (specifically, the value of each pixel, minus the mean, divided by the standard deviation), and the Z-scores are then filtered, thresholded and connected as in the other methods.

This method closely resembles the background differencing approach used in WorldKit (Chapter 4). It also aims to capture the essence of the KinectFusion SLAM touch tracking approach [Izadi 2011], in that it integrates the background profile gradually over time, building a statistical model of the environment that is much more accurate than any single frame. However, our replicated approach lacks the spatial averaging of KinectFusion.

6.3.4 Slice Finding and Merging

For our final comparison technique, we chose to implement the slice-finding approach used in OmniTouch [Harrison 2011]. This approach locates cylindrical slices in the depth image using an elastic template, and then links together adjacent slices to form fingers. Of note, unlike the other approaches, OmniTouch requires that the fingers be clearly separated in the depth image. Furthermore, as it does not use a background model, it may detect erroneous touches on the surface due to objects already in the scene; therefore, for the study, we cleared the table surface of all objects.

The original OmniTouch implementation only supported fingers oriented horizontally. We therefore extended OmniTouch by implementing biaxial template matching – locating candidate finger slices in both the X- and Y-axes – and then merging the centroids of these slices into fingers. Our approach demonstrably locates fingers oriented in any direction. To best replicate OmniTouch's true performance, we also implemented the system's finger forward-projection method. As noted previously, because the fingers fuse with the depth background upon touch, it is difficult to estimate the true tip position. In response, OmniTouch uses the detected finger's orientation to extend the estimated touch position 15 mm towards the un-sensed tip [pers. comm.]. This improvement is not directly applicable to the previously discussed background-subtraction methods, as they do not model the finger orientation or shape. Of note, the original OmniTouch system was designed to operate at a distance of just 40 cm.

6.4 Evaluation

To assess the accuracy of DIRECT, we ran an evaluation with 12 participants (3 female; average age 25). All users were familiar with touch interfaces. Each study session lasted for roughly 30 minutes, and participants were compensated $10 USD for their time.

Participants were simply told that the table surface was a touchscreen from the outset, and to touch the surface as they would any ordinary touchscreen. Users were permitted to use either hand (including interchangeably) and to use any finger pose they found comfortable. Users were not required to remove jewelry or roll up sleeves – several users conducted the study while wearing long sleeved shirts, bracelets, watches and rings. Our experiment system ran all five touch-tracking methods (DIRECT plus the four comparison techniques) simultaneously at a consistent 30 fps (the frame rate of the depth sensor).

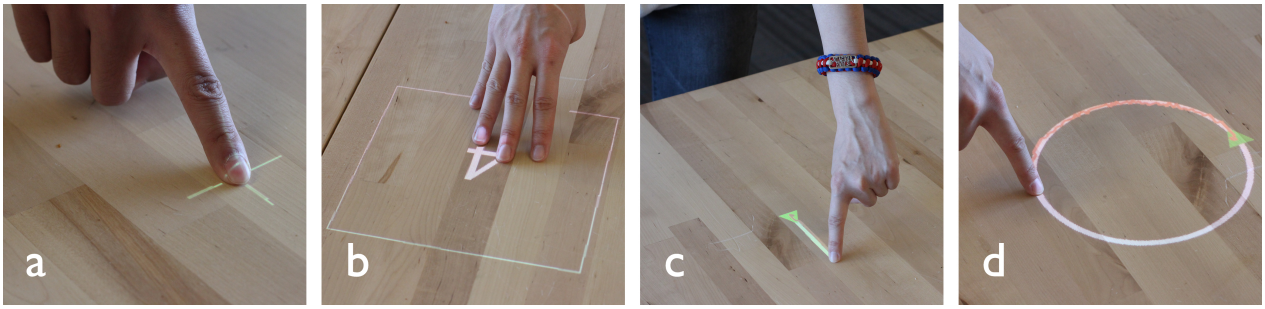

Figure 6.6. Tasks performed by users in the DIRECT study.
(a) crosshair task, (b) multitouch box task, (c) line shape tracing task, (d) circle shape tracing task.

6.4.1 Tasks

Participants completed a series of small tasks, organized into three categories. Task order was randomized per participant to mitigate order effects. For each task, users were instructed to stand along one of the two long edges of our test table (thus changing the orientation of their touches).

Crosshair: Participants placed their fingertip on a projected crosshair (Figure 6.6a), after which the experimenter manually advanced the trial and the touches detected by each tracker were recorded. This task measured the positional accuracy of each finger tracking method and touch segmentation accuracy. Crosshairs were arranged in a 4x8 grid spanning the table surface, but were shown one at a time and in random order. This task was performed twice for each edge of the table.

Multitouch Segmentation: Participants were instructed to place a specific number of fingers within a projected 20 cm square on the table (Figure 6.6b). The experimenter manually advanced the trial and the number of touches reported within the box for each tracker was recorded. This task was intended to measure the multitouch contact reporting accuracy of each technique (*i.e.*, false positive and false negative touches). Six boxes were specified across the length of the table, and the number of fingers varied from 1-5 for a total of 30 trials, randomly ordered. This task was also performed twice for each edge of the table.

Shape Tracing: Participants were instructed to trace a particular projected shape, beginning at the start position indicated with a green triangle (Figure 6.6c,d), and tracing to the end of the path. For each frame between the start and end of the trial, we recorded the touch coordinates for each method. This task was intended to replicate the tracing task used in OmniTouch [Harrison 2011]. There were three instances of this task per table edge, one for each shape: horizontal line, vertical line, and circle.

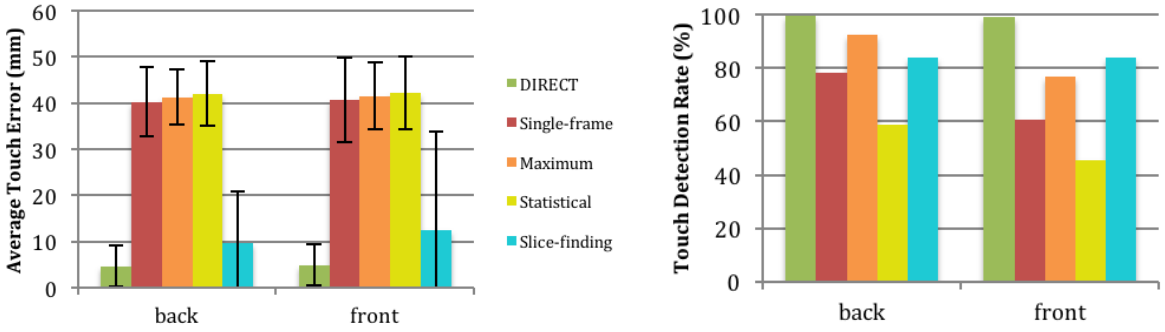

Figure 6.7. Touch error and detection rate for DIRECT and competing methods.
Average touch positional error (left) and touch detection rate (right) for each of the five touch tracking methods. Error bars are standard error.

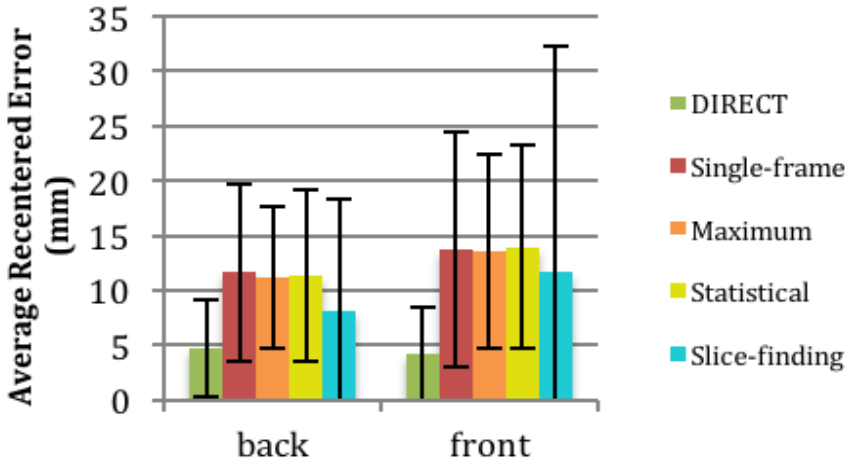

Figure 6.8. Touch error after *post hoc* offset correction.
Average positional error after removing the average offset vector and assuming *a priori* knowledge of the user's orientation. Error bars are standard error.

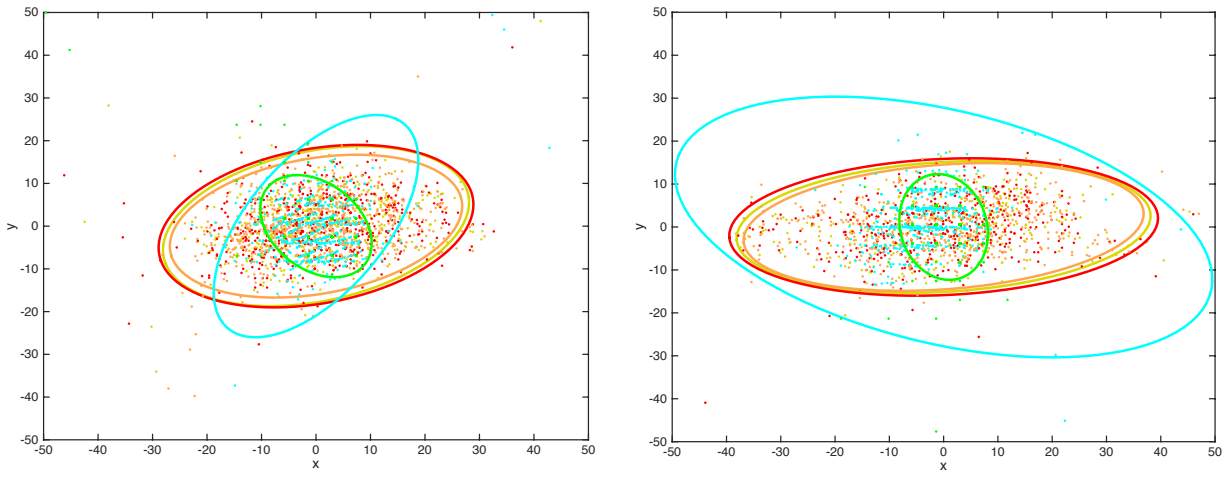

Figure 6.9. 95% confidence ellipses for crosshair task.
Left: from the back of the table; right: from the front of the table. X and Y units are in millimetres; colours are as in Figure 6.8.

6.5 Results and Discussion

In our results, we denote the two edges of the table as "back" and "front". The top of the Kinect's image corresponded to the front edge of the table.

6.5.1 Crosshair

The crosshair task allowed us to test both touch positional accuracy and touch detection rate. Due to potential spuriously-detected touches, we measured accuracy as the mean distance from the finger to the *nearest* detected touch point for each tracker. Touches further than 200 mm from the finger were not counted, since those touches would clearly be erroneous. If no touches were detected in range for a particular tracker, the tracker was considered to have failed to detect the touch.

In total, we collected 768 trials for each side of the table. The touch positional accuracy results are summarized in Figure 6.7. The accuracy results show a slight but consistent increase in accuracy across all trackers when users stood at the front edge of the table.

DIRECT achieved an average Euclidean-distance positional error of 4.8 mm across all trials, with a 99.3% touch detection rate. The next best technique, slice finding, had an average positional error of 11.1 mm and a touch detection rate of 83.9%. The background modeling methods all performed poorly, with average positional errors of over 40 mm and touch detection rates ranging from 52.1% to 84.8%. Put simply, these methods do not have the necessary sophistication to segment small finger contacts at the noise level present when sensing at 1.6 meters.

During development, we noticed that the slice finding method without forward projection performed very poorly (~20 mm average error), so it was clear that finger forward projection was crucial to obtain good accuracy. This is because these methods cannot accurately locate the fingertip in the noise, and so they instead locate a point somewhere along the finger.

Therefore, we also analyzed the accuracy of the four competing approaches by applying a mean offset vector (*i.e.*, a post hoc global offset). This vector depends on knowing the precise finger orientation, and thus the offset correction corresponds to a "calibrating" of the touch algorithm from a *fixed* user position and assuming the finger is extended perpendicular to the table. Consequently, we computed offsets separately for the front and back user positions. Because the prior systems recognize neither the user position nor finger orientation, this result is purely *hypothetical*, but serves as a useful benchmark.

The resulting average *offset-corrected* errors (Figure 6.8) were 4.46 mm for DIRECT (a negligible 0.3 mm improvement), 9.9 mm for the slice finding method (a modest 1.2 mm improvement), and 12.3-12.7 mm for the background modeling approaches (a significant 20-30 mm improvement). To visualize these errors, we also computed the 95% confidence ellipsoids (Figure 6.9) for each tracker (note that the offset correction corresponds to recentering the ellipsoids). These errors are consistent with prior results from Wilson [Wilson 2010a] and OmniTouch [Harrison 2011], suggesting that our offset-corrected comparison implementations are reasonably close to the original implementations.

6.5.2 Multitouch Segmentation

With the multitouch segmentation task, we aimed to measure the false positive and false negative rates for touch detection with multiple fingers present in a small region. Under these conditions, touch trackers might merge adjacent fingers or detect spurious touches between fingers. The differences between the back and front sides were not significant in this task, so the results have been combined. In total, 1440 trials were collected.

Detecting a single extended finger is the easiest task. In single finger trials, DIRECT detected the correct number 95.8% of the time. Single-frame background, maximum frame background and statistical model background achieved 52.8%, 66.3% and 35.1% respectively. Slice-finding was 75.0% accurate.

Detecting several fingers in close proximity is much more challenging when sensing at 1.6 meters. With all trials combined, DIRECT detected the correct number of fingers in 75.5% of trials, more fingers than were present in 2.4% of trials and fewer fingers than were present in 22.1% of trials. The three background modeling approaches – single-frame, maximum frame and statistical model – detected the correct number of fingers 22.2%, 29.2% and 17.3% of the time. Very few trials reported more fingers than were present: 9, 7 and 4 trials respectively (<0.1% of all trials). Instead, these methods tended to miss fingers (77.1%, 70.2%, and 82.4% of trials respectively). Finally, the slice-finding approach detected more fingers in just 9 trials (<0.1% of trials), fewer fingers in 75.4% of trials, and the true number of fingers in 24.0% of trials.

We tuned the comparison technique implementations to minimize spurious touches while nothing was touching the table. However, our multitouch segmentation results suggest that optimizing for this criterion could have rejected too many legitimate touches, reducing touch detection rates. On the other hand, increasing touch sensitivity significantly increases noise and errant touches. For example, decreasing the “low boundary” depth threshold in the maximum-distance background model tracker by a single millimeter results in hundreds of errant touches detected on the surface every second, which is clearly not acceptable.

6.5.3 Shape Tracing

Tracking moving fingers is extra challenging as the exposure time of the depth camera produces motion blur across the moving parts of the frame, reducing accuracy. Further, as already mentioned above, our comparative methods are prone to missing fingers. In the case of finger movement, this will manifest as loss of finger tracking for several frames. During this period, spurious input would often cause the traced path to zigzag. For this reason, it was simply not possible to complete a realistic and useful analysis with our study data.

However, we can use results reported in OmniTouch [Harrison 2011] as one point of comparison. Specifically, OmniTouch reports a mean error of 6.3 mm (SD=3.9 mm) at a sensing distance of 40 cm on a flat notepad. For comparison, DIRECT achieves a mean error of 2.9 mm (mean SD=2.7 mm) at a sensing distance of 160 cm (on a flat table).

6.6 Conclusion

I have described DIRECT, a touch tracking system which strategically merges depth and infrared data from an off-the-shelf depth camera to enable highly accurate touch tracking, even at significant distances from the sensor. Overall, DIRECT demonstrates greatly improved touch tracking accuracy (mean error of 4.9 mm) and detection rate (>99%) – roughly twice as good as the next best method in the literature, and nearly ten times better than classic approaches.

There are also immediate ways to further improve our system. For example, in our present implementation, DIRECT outputs integer coordinates on the depth map, which quantizes the X/Y position to 4.4 mm (mean error due to quantization: 3.1 mm). Averaging pixels at the tip could provide a sub-pixel estimate, further boosting accuracy. Additionally, we could apply temporal smoothing to improve touch position stability at the expense of latency.

To conclude, I hope that DIRECT's significantly improved finger tracking accuracy can open new opportunities in ad hoc touch interfaces and better allow this emerging computing modality to be explored. As we'll see in later chapters, this foundational technology can help move depth-driven systems from proof-of-concept to more practical and widespread use.

CHAPTER 7. TOWARDS THE INFOBULB

7.1 Summary

Now that we have refined touch input and a set of suitable interactions, the major challenge is to miniaturize the components to make them suitable for building a real *infobulb*. In this chapter, I detail LumiWatch, a project to build an extremely small projector and depth sensing unit, tiny enough to be integrated directly into a smartwatch form factor. LumiWatch is the first, fully-functional and self-contained projection smartwatch implementation, containing the requisite compute, power, projection and touch-sensing capabilities to convert the user's arm into a touchscreen-like interface. This watch offers roughly 40 cm^2 of interactive surface area – more than five times that of a typical smartwatch display. I demonstrate continuous 2D finger tracking with interactive, rectified graphics, discuss the hardware and software implementation, and detail evaluation results regarding touch accuracy and projection visibility.

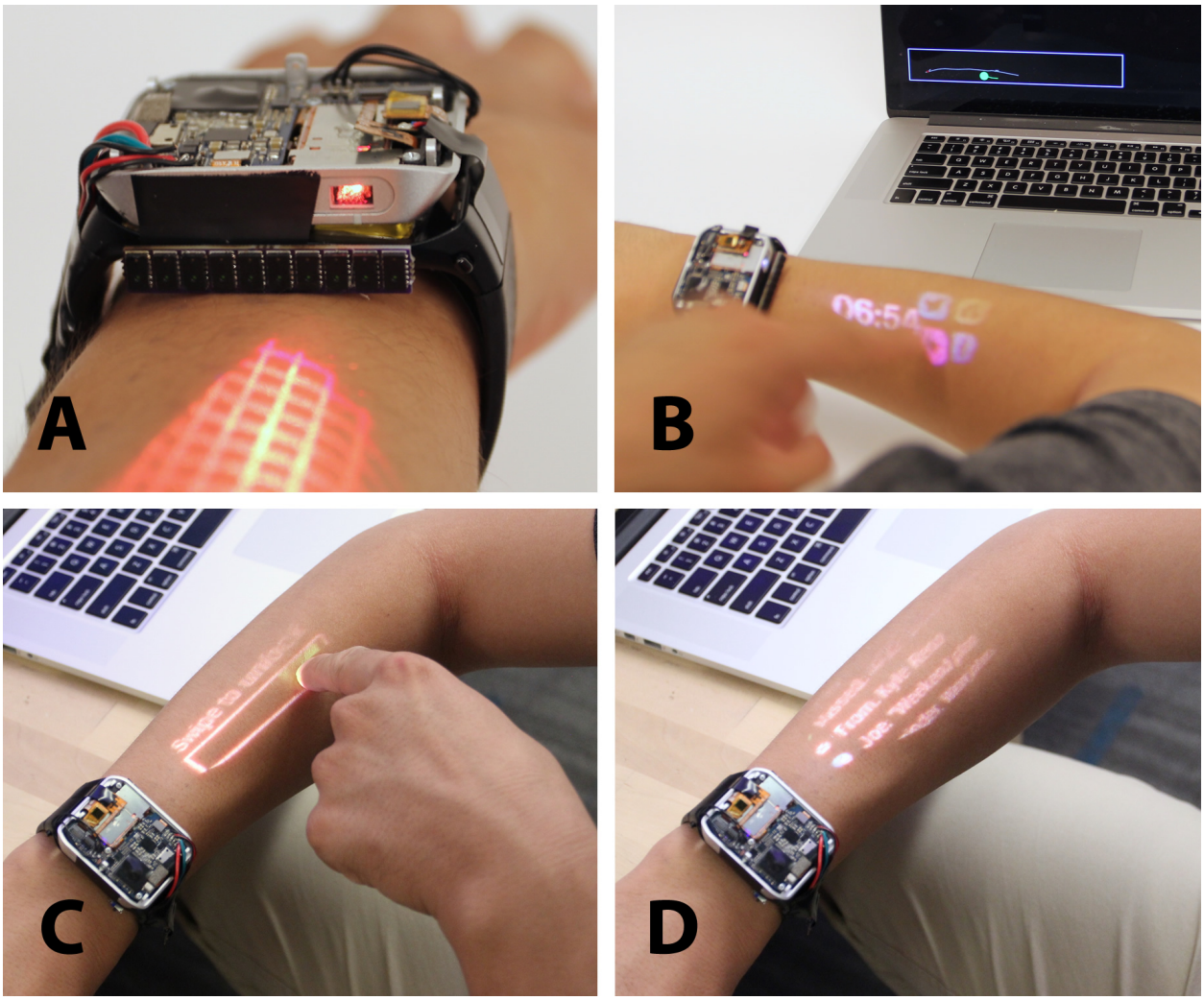

Figure 7.1. The LumiWatch prototype.
(A) LumiWatch provides rectified graphics with touch input on the skin (B). We use a slide-to-unlock mechanism to reject inadvertent touches and provide a rapid projection calibration (C) before apps can be used (D).

7.2 Introduction

Appropriating the human body as an interactive surface is attractive for many reasons. Foremost, skin provides a natural and immediate surface for dynamic, digital projection. Although it introduces some color and physical distortion, the resolution, framerate and overall quality can be high [Harrison 2010c, Harrison 2011, Mistry 2009, Yamamoto 2007]. More importantly, it offers considerable surface area for interactive tasks – many times that of *e.g.*, a smartwatch display.

With today's smartwatches containing multi-core, multi-gigahertz CPUs, one could argue their small touchscreens are the chief bottleneck to unlocking richer and more useful applications. Indeed, several widely publicized, *conceptual* on-skin projection watches have been proposed, most notably the crowdfunded projects Cicret and Ritot.

A second benefit is that our bodies are always with us, and are often immediately available [Saponas 2009, Tan 2010]. This stands in contrast to conventional mobile devices, which typically reside in pockets or bags, and must be retrieved to access even basic functionality [Ashbrook 2008b, Hudson 2010, Saponas 2011]. This generally demands a high level of attention, both cognitively and visually, and can be socially disruptive. Further, physically retrieving a device incurs a non-trivial time cost, and can constitute a significant fraction of a simple operation's total time [Ashbrook 2008a].

Lastly, as the colloquialism "like the back of your hand" suggests, we are intimately familiar with our own bodies. Though proprioception, we can easily navigate a finger to our palm, even with our eyes closed. We have finely tuned muscle memory and hand-eye coordination, providing a high level of input performance, for both absolute touch location and relative gesturing – powerful interaction modalities that worn systems can leverage.

However, despite these significant benefits, building practical, worn projection systems has remained elusive. In order to achieve sufficient projection brightness, sensing robustness and compute power, past on-body systems have employed full-sized components (*e.g.*, USB depth cameras [Harrison 2011, Sridhar 2017], portable projectors [Mistry 2009, Yamamoto 2007]). The resulting size of these systems mean that they must be worn on the upper arm or shoulder, and most often tethered for compute and power [Harrison 2010c, Harrison 2011, McFarlane 2009, Mistry 2009, Yamamoto 2007].

We partnered with ASU Tech to develop *LumiWatch*, a custom, tightly integrated and fully self-contained on-skin projection wristwatch. It incorporates a 15-lumen scanned-laser projector, a ten-element time-of-flight depth-sensing array, quad-core CPU running Android 5.1, and battery

good for one hour of continuous projector operation (or one day of occasional use). Our prototype measures 50×41×17 mm, nominally larger than the production 42 mm Apple Watch Series 3 (43×36×11 mm). With our watch, we demonstrate continuous 2D finger touch tracking on the skin with coordinated interactive graphics. Owing to the shallow angle of projection, our graphics pipeline must rectify interfaces to the complex, non-planar geometry of the arm.

Our hardware and software, taken together, transform the arm into a coarse touchscreen, offering roughly 40 cm^2 of interactive surface area, more than five times that of a typical smartwatch. This interactive area supports common touchscreen operations, such as tapping and swiping, allowing it to offer similar interactions to that of a (single-touch) smartphone. Although obstacles remain for practical adoption, we believe our work demonstrates the first functional projection smartwatch system and constitutes a significant advance in the state of the art.

7.3 Related Work

Given that the surface area of one hand exceeds that of a typical smartphone screen, and humans have roughly 1-2 m^2 of skin surface area in total, it is unsurprising that researchers have tried to appropriate it for digital input. A wide variety of sensing approaches have been considered, which we briefly review.

Optical approaches are most common. For example, SenSkin [Ogata 2013] and SkinWatch [Ogata 2015] both used infrared proximity sensors to detect skin deformations resulting from touch inputs. More similar to our system, [Lim 2015] used a pair of infrared emitters and a photodiode array attached to the side of a watch to track a 2D finger location, while SideSight [Butler 2008] used an array of proximity sensors to sense finger position around a phone. Researchers have also investigated worn cameras for touch tracking, including arm [Sridhar 2017] and shoulder [Dezfuli 2012, Gustafson 2011] vantage points.

Acoustic methods are also powerful. SonarWatch [Liang 2011] and PUB [Lin 2011] both used ultrasonic sonar to measure distance to a finger interacting on an arm's surface, providing 1D touch tracking. It is also possible to instrument the finger with an active signal source, as seen in The

Sound of Touch [Mujibiya 2013]. There are also passive techniques, such as TapSkin [C. Zhang 2016] and ViBand [Laput 2016], which rely on bioacoustic propagation of vibrations resulting from taps to the skin.

There has also been some work in RF and capacitive sensing, for instance, Touché [Sato 2012] used a wrist-worn, swept frequency capacitive sensor to detect gestural interactions between a wearer' two hands. The instrumented smartwatch seen in AuraSense [Zhou 2016] used projected electric fields to enable close-range buttons and sliders on the skin. By using an active ring, Skintrack [Y. Zhang 2016] enabled continuous 2D finger tracking on the skin through RF triangulation. Finally, the skin itself can be instrumented, sidestepping many remote sensing issues. DuoSkin [Kao 2016], iSkin [Weigel 2015] and [Kramer 2011] have all demonstrated flexible, skin-compatible overlays supporting capacitive touch sensing.

Of note, none of the previously-mentioned systems attempted touch tracking and projection in a smartwatch-like device, as the small size requirement, shallow-angle projection, and oblique sensing viewpoint all pose significant challenges. The closest system to this desired form factor is Skin Buttons [Laput 2014], which used fixed-icon (*i.e.*, rendered to film) laser projections coupled with IR proximity sensors to detect clicks to "skin buttons".

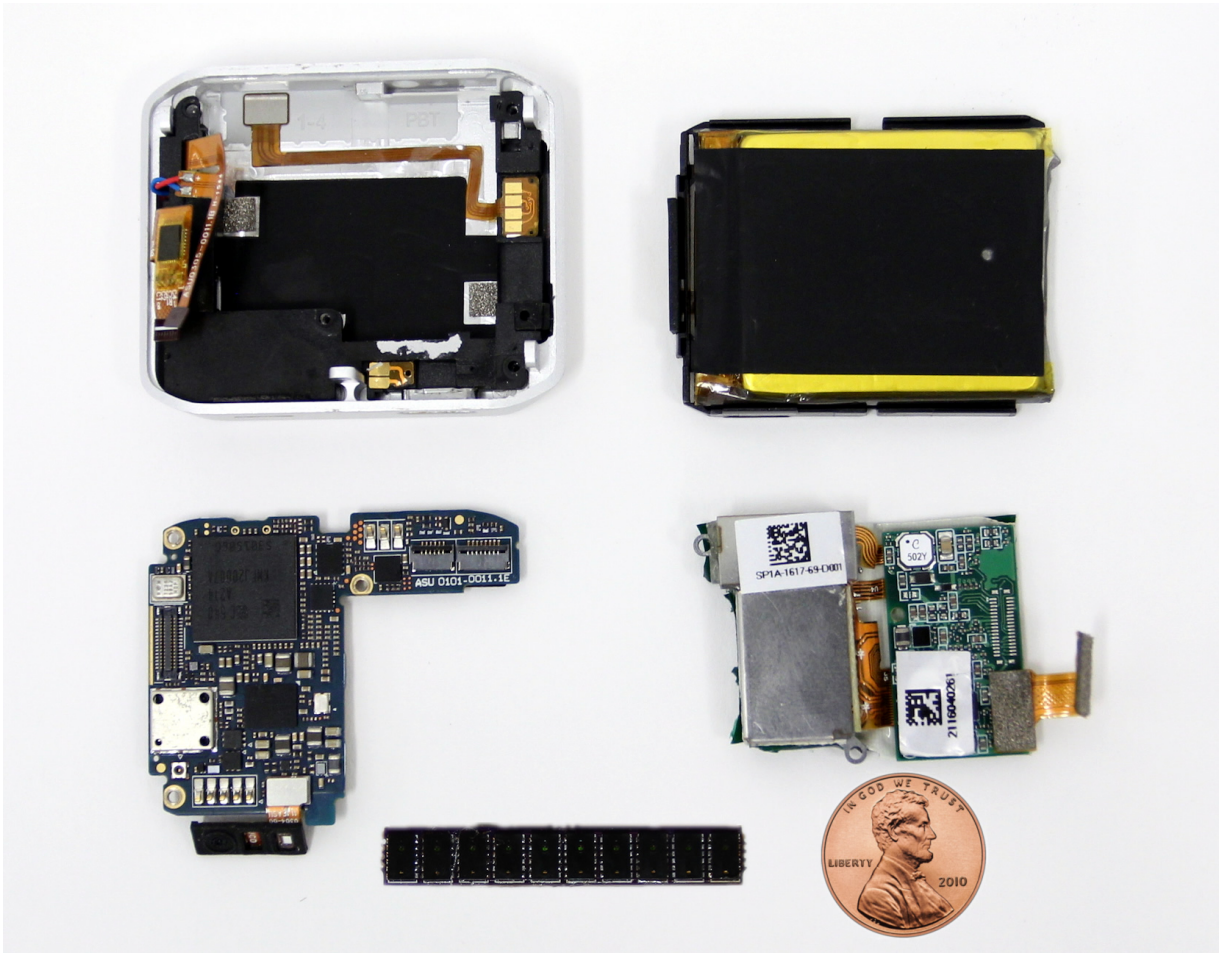

Figure 7.2. Main hardware components of LumiWatch.
Clockwise from top-left: heat-dissipating aluminum shell; battery; projector module and driver; 1D sensor array; and logic board.

7.4 LumiWatch Hardware

Our custom smartwatch hardware consists of five primary components, seen in Figure 7.2: a logic board, projector, depth-sensing array, metal enclosure and battery. It is fully self-contained and capable of independent operation (*i.e.*, no tether to a smartphone or computer). We estimate immediate retail cost would be around $600.

7.4.1 Logic Board

Our smartwatch logic board (Figure 7.2, bottom left) was designed around a Qualcomm APQ8026 system-on-chip, which integrates a 1.2 GHz quad-core CPU, 450 MHz GPU, and Bluetooth 4.0 and WiFi controller. We also added 768 MB of RAM, 4 GB of flash memory, inertial measurement unit (IMU) and ambient light sensor. The component placement on the logic board was optimized to reduce thermal interference from our projector. Our smartwatch runs Android 5.1, with smart-watch-specific applications.

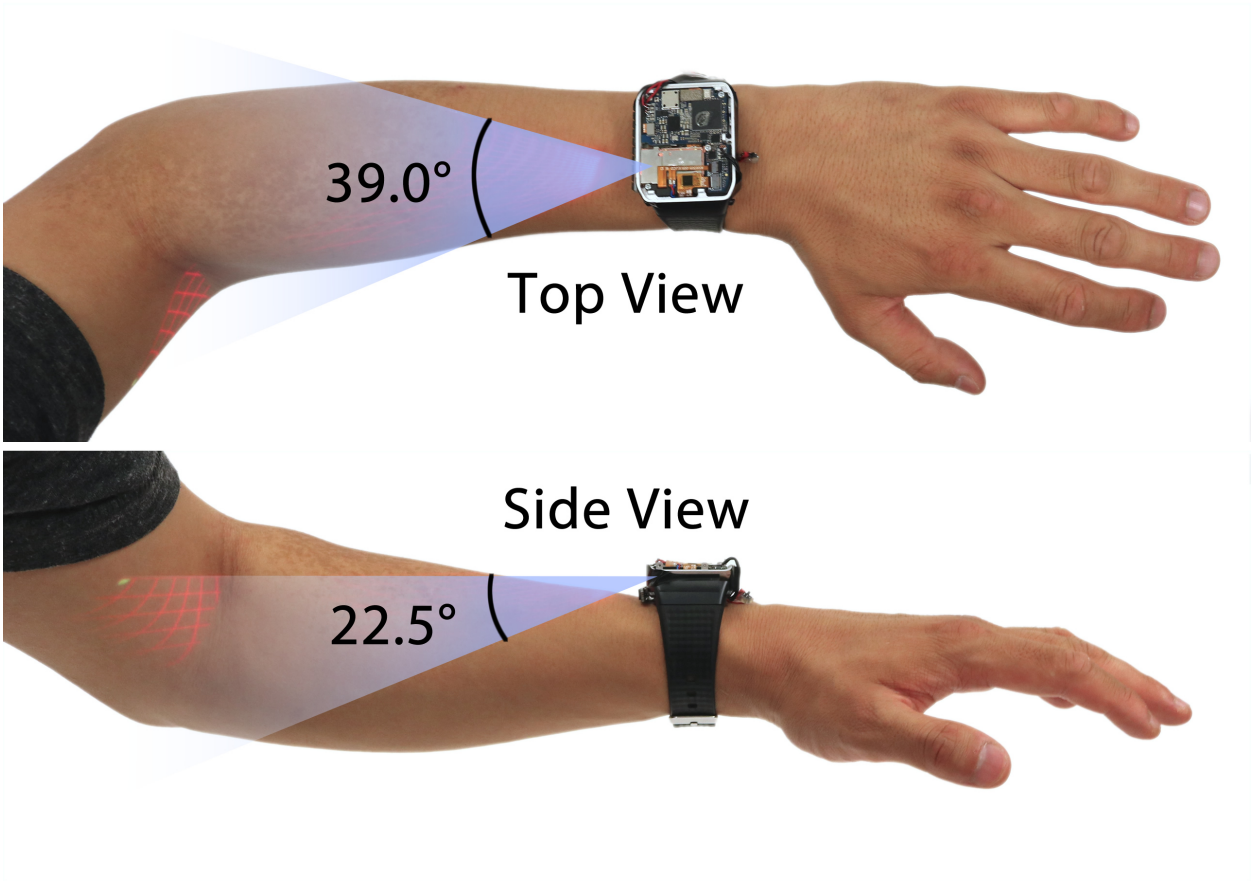

Figure 7.3. Illustration of the projector's field of view.
Top: top view of the arm showing the tangential field of view. Bottom: side view of the arm showing axial field of view.

7.4.2 Projector

We designed and manufactured a custom 15-lumen pico-projector module for our smartwatch. The projector uses three lasers (red, green and blue) with a pair of MEMS mirrors operating in a raster-scan mode (*i.e.,* a scanned laser design [Kiang 1998]). This emits a 1024×600 image at 60 Hz across a 39°×22.5° field of view (Figure 7.3). Our projector module measures 25.8×16.6×5.2 mm (Figure 7.2, bottom-right), and consumes up to 2.7 W of power displaying a maximum-brightness, full-white image. With scanned-laser designs, power usage varies by content, *e.g.*, black imagery requires almost no power. The projector module is paired with custom drive electronics that controls the lasers and mirrors, while exposing a standardized display interface to the Android firmware.

7.4.3 Depth-Sensing Array

To capture touch input on the projected area, we designed a compact 1D depth-sensing array (7×38×3 mm), consisting of ten STMicro VL6180X time-of-flight sensors (Figure 7.2, bottom-right). Each individual sensor determines the distance to the nearest object within a 25° cone by emitting pulsed infrared light and timing the returned reflection. These sensors are connected to a dedicated microcontroller (NXP MK20DX256) over I^2C, which triggers measurements in round-robin fashion to avoid infrared collisions. Capturing all ten measurements takes ~36ms, resulting in a touch tracking frame rate of 27.5 Hz.

7.4.4 Battery & Shell

The smartwatch electronics are contained in an aluminum shell (Figure 7.2, top-left), which assists in dissipating heat generated by the projector and logic board. The entire smartwatch prototype (logic board, projector, and depth sensor) is powered from a 740 mAh, 3.8 V (2.8 Wh) lithium-ion battery (Figure 7.2, top-right), making it fully self-contained. The battery sits below the projector, close to the skin. Combined with the thickness of the enclosure, this causes the center of the projector aperture to sit approximately 13 mm above the surface of the skin, providing a slightly higher vantage point for projection. Under average use conditions, we obtain over

one hour of continuous projection (with CPU, GPU, WiFi, and Bluetooth all active). In more typical smartwatch usage, where the projection would only be active intermittently, we expect our battery to last roughly one day.

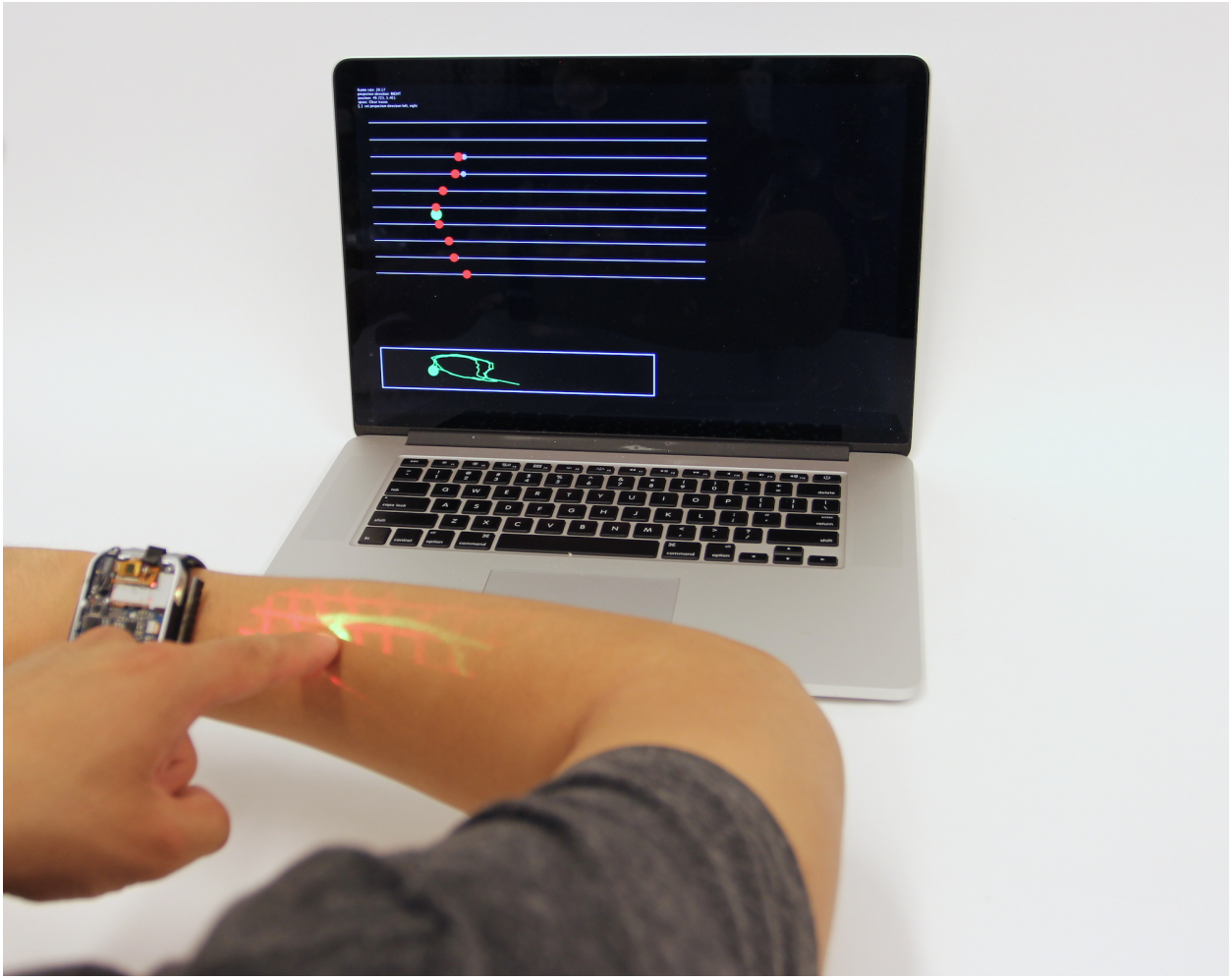

Figure 7.4. LumiWatch touch tracking in action.
Top of screen: raw data from our ten time-of-flight sensors (red dots), with estimated touch point shown in green. Bottom of screen: resulting touch paths. On arm: current path is projected for debugging.

7.5 Touch Tracking

We use our 1D depth-sensing array to track fingers on or very near to the surface of the arm. The conical fields of view of each individual time-of-flight sensor overlap significantly, so that a single

finger will always appear in several sensors simultaneously (Figure 7.4). However, the finger will always be closest to the sensor most directly inline with the sensing axis, allowing us to infer a finger's tangential *y* position on the arm (in addition to the axial *x* position along the arm provided by the distance measurement itself). Thus, our 1D depth-sensing array gives us the ability to track the 2D position of a finger on the arm.

Our finger-tracking algorithm works in stages. First, the incoming sensor values are smoothed with an exponentially-weighted moving average (EWMA) filter to reduce noise and jitter (Figure 7.4 top, red dots). Next, sensors with invalid measurements or distances > 150 mm are removed from consideration. If there are at least two sensors remaining, the algorithm detects a finger. It uses the minimum distance measurement among all sensors as the raw *x* position of the touch point. Then, to find the *y* position, it computes a weighted average of the reported sensor positions:

$$w_i = \frac{1}{x_i - x + d}, \qquad y = s \cdot \frac{\sum_{i=1}^{10} i w_i}{\sum_{i=1}^{10} w_i}$$

where *d* is a tuning parameter that mitigates noise (set to 5 mm in our implementation) and *s* is the pitch between sensors (here, *s* = 3.78 mm). This average heavily weights x_i values near the minimum *x*, thus effectively allowing it to average between multiple noisy minima while ignoring values far from the minimum. Finally, the *x* and *y* values are smoothed with an EWMA filter and reported to running applications.

On top of the basic *x* and *y* finger position, we detect single finger taps and swipes, to enable richer application input. Both detection mechanisms operate on *finger strokes*, which consist of the entire trace of a finger's position from initial detection to disappearance. A short finger stroke (total distance travelled less than 20 mm) is inferred as a finger tap, while long finger strokes are considered swipes. The position of a finger tap is taken as the point of minimum velocity during a short stroke. The swipe orientation (horizontal or vertical) is determined by comparing the total *x* and *y* travel distances across the long stroke; a stroke that travels at least 3 times further in the *x* direction as the *y* direction is considered horizontal. This factor of 3 accounts for the proportions of the arm, as strokes along the arm were observed to be significantly longer than vertical ones.

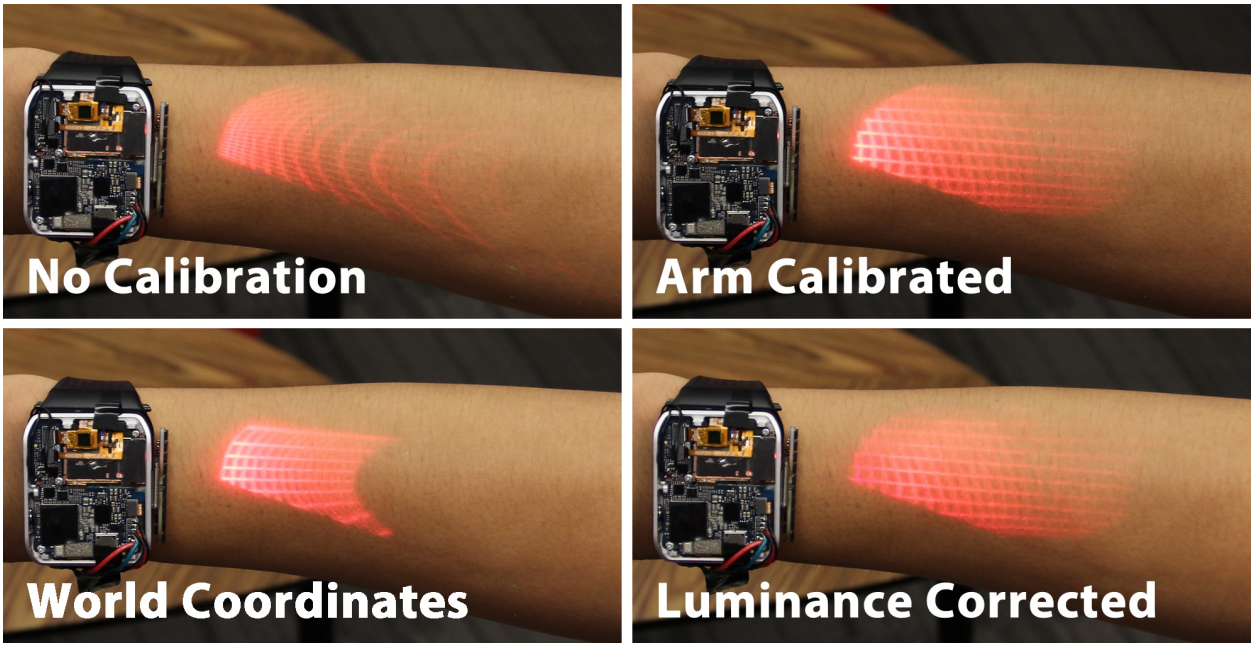

Figure 7.5. Various stages of projection calibration.

7.6 Projected Output

Most digital projectors emit a rectangular image that ideally falls on to a flat, perpendicular surface. However, from the perspective of our smartwatch, an arm's surface is neither flat nor perpendicular, but rather irregularly curved and nearly parallel. This introduces a number of challenges, which we address through a combination of hardware design and efficient software calibration. Our graphical rectification pipeline is lightweight enough to allow application software to run at our projector's native 60 frames per second.

7.6.1 Hardware Design

First, the hardware is designed to elevate the projector as high as possible without wasting space. To do this, we made the projector the top most component in our watch, allowing the aperture to lie 13 mm above the surface of the skin. At this height, a pixel 150 mm from the projector on the arm's surface is less than 5° from parallel. A shallower projection angle results in larger pixels and coarser projection, as well as increased sensitivity to variations in arm angle and geometry.

To keep the smartwatch thin, the projector is mounted so that the shorter 600-pixel axial projection axis extends horizontally across the skin, with the field of view extending from 0° horizontal at pixel 600 (parallel to an imaginary flat arm) down to 22.5° below the horizontal at pixel 0 (pointing down towards the arm). See illustration in Figure 7.3. This results in a gap of approximately 13 *cot*(22.5°) = 31 mm between the projector and the projected image on the arm. Here, we note that increasing the field of view of the projector will slightly reduce this gap but will have a significant negative impact on the fidelity of the image far from the projector (i.e., larger pixels). Thus, increasing the *y*-axis field of view is not necessarily desirable for a projection smartwatch.

7.6.2 World Coordinate Rectification

Figure 7.5, top-left, shows the default projection without any correction (*i.e.*, projecting as if onto a perpendicular flat plane). The first software correction step is to achieve projection in "world coordinates," that is, determining the mapping between the projector's image coordinates and real-world 3D coordinates. These world coordinates also correspond to touch input positions, making this correction doubly crucial. As this calibration depends only on the hardware alignment between the projector and touch sensor, it can in practice be performed "at the factory." In our system, we calibrate by finding the projector pixel coordinates corresponding to ten non-coplanar points in the real world, and then use a least-squares optimization algorithm to compute the projection matrix which best maps these 3D points to the 2D projector coordinates. With this projection matrix, we can specify draw commands in world coordinates, and the resulting projected pixels will appear at those coordinates. Notably, our smartwatch's GPU performs this correction with no additional computational load because rendering with a projection matrix is already performed by default in a 3D context, making this a very efficient calibration.

7.6.3 Arm Calibration

On its own, this "world correction" would be sufficient to display on a flat shallow-angle surface, such as a table. However, the arm is not a flat surface, but rather an irregular cylinder-like object.

The curvature of the arm introduces severe distortions relative to a flat surface (Figure 7.5, bottom-left), and so the next step is to calibrate to the arm. This stage depends on the wearer's arm geometry. When this information is not available, a "default" calibration for an average arm can be used, which will still be significantly superior to no arm calibration at all (Figure 7.5, bottom-left vs. top-right).

Ideally, arm calibration is performed using a 3D model of a user's arm. However, we created a simple model by interpolating between two ellipses representing the user's wrist and forearm cross-sections. We render all application content to an offscreen texture, then map the texture onto the model, effectively performing projection mapping onto the arm. This rendering process removes the curvature of the arm (Figure 7.5, top-right). Rendering a textured 3D model is quite efficient using our onboard GPU, so this calibration step can be performed with minimal overhead.

7.6.4 Luminance Correction

Next, we perform luminance correction, as the pixels near the projector are brighter than those farther away (as seen in Figure 7.5, top-right). This is achieved by pre-computing a map of the approximate distance from the projector to each pixel on the arm. This map is then transformed into a luminance correction map by using the relation:

$$factor = clamp(dist^2 / maxdist^2, 1/16, 1)$$

This configures a correction which scales the luminance by the square of the distance, but which is clamped to avoid reducing the luminance too much for close pixels; *maxdist,* which we set to 70 mm, configures the furthest distance to apply the correction, beyond which pixels are displayed at full brightness. Put simply, this correction reduces the brightness of the pixels closest to the projector, providing a more uniform appearance (Figure 7.5, bottom-right).

7.6.5 Dynamic Projection-input Calibration

The prior projection calibration steps were all fixed steps that could be performed once per user or device, and then saved for future use. However, the precise angle between a wearer's wrist and forearm cannot be calibrated *a priori,* because this angle can differ each time the watch is worn or wearer lifts their arm, necessitating some kind of dynamic calibration. For example, a difference of just one degree in the tilt of the projector can produce significant shifts in the projected output – e.g., a pixel that appears 130 mm away with the arm at 0° would appear at 110 mm with the arm angled just 1° towards the projector.

To solve this problem, we leverage the familiar *swipe to unlock* gesture often used on smartphones. Upon lifting the smartwatch to initiate an interaction, the user is prompted with a "swipe to unlock" slider (Figure 7.1C). The slider begins at a constant axial distance of 70 mm based on the default arm angle, but may appear in a different place depending on a user's true arm angle. When a user "grabs" the slider's handle, the finger's axial position is recorded, and the true arm angle is recovered by comparing the actual axial position with the assumed position of the slider and applying trigonometry. In a similar fashion, the touch points generated by the swipe allow us to dynamically calibrate our world coordinate transform. Upon completion of the swipe gesture, the recovered calibrations are applied for the remainder of that interactive session (*i.e.*, until the user stops interacting with the watch).

Thus, this simple unlock gesture serves three purposes: 1) it allows the arm angle to be seamlessly and intuitively calibrated without an explicit calibration routine, 2) it provides the necessary dynamic calibration for aligning the projector and touch sensor, and 3) it verifies the user's intent to interact with the system (mitigating *e.g.* false inputs).

7.7 Evaluation

We performed two focused studies to evaluate the performance of our system. The first study evaluates input performance, while the second tests projection performance.

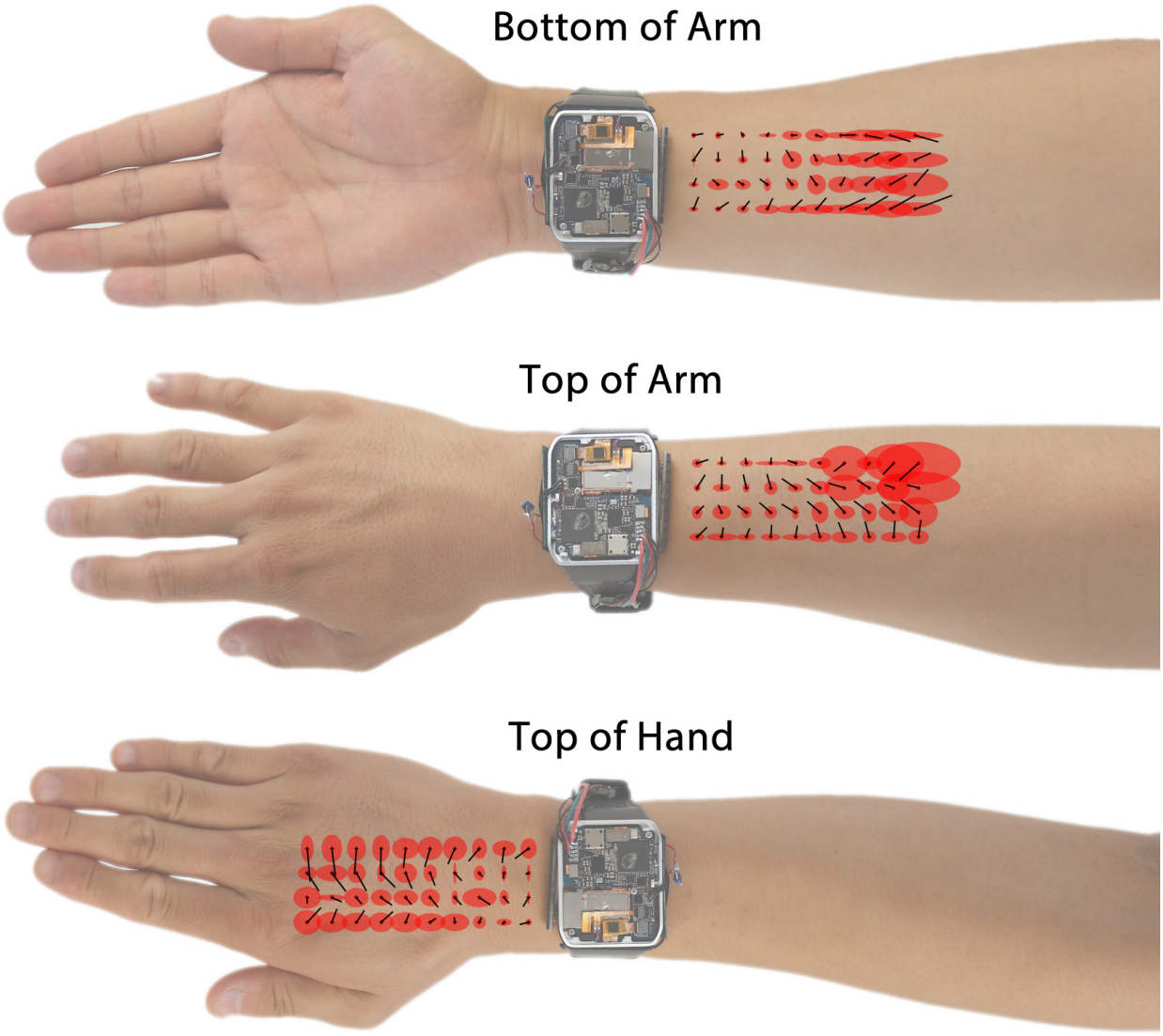

Figure 7.6. Touch input study results.
Rows show our three tested projection locations with an illustration of results from the touch study. Red ellipses are sized to ±1 standard deviation, black lines represent mean displacement across all users.

7.7.1 Study 1: Input Performance

We recruited five participants (one female, two left-handed) to evaluate the touch input performance of our system, who were compensated $20 USD for the 30-minute study. We measured the wrist and forearm circumferences for each user, which ranged between 133~168 mm and 197~302 mm respectively. Participants wore the watch on the arm opposite their dominant hand

in three different projection positions (tested in random order): bottom of arm, top of arm, and top of hand (illustrated in Figure 7.6). Note that we did not test input performance on the palm because the hypothenar and thenar eminences occluded sensing and projection. Participants sat in a chair and held their arm up as if to check the time on a wristwatch.

For each projection position, users performed a *tap* task and a *swipe* task. Before the tap task, the experimenter used a flexible stencil to draw a 4×10 grid of dots, spaced 10 mm apart, extending outwards from the watch. The user was instructed to touch their finger to each dot in a sequential order. The system ran the tap detector in real-time, recording any tap event received. After each touch, the experimenter would click a key to record the trial (even if no touch was registered by the system).

In the swipe task, users performed four different swipe gestures (left, right, up, and down) ten times each in a random order. Participants were instructed to perform a swipe direction, after which the experimenter clicked a key to advance to the next trial. The system ran the swipe detector in real-time, recording each gesture detected.

In total, we collected 600 touch points (40 dots × 3 projection locations × 5 participants) and 600 swipe gestures (4 swipe directions × 10 repeats × 3 projection locations × 5 participants).

Touch Accuracy Results

There were no significant differences in tap accuracy across the three projection locations. Over all 600 trials, a tap was detected in 596 of them, for an aggregate tap detection accuracy of 99.3%. The mean absolute error over all trials was 7.2 mm, with a 95th percentile absolute error of 15 mm. The error disproportionately affected points far from the sensor array. Figure 7.6 shows a plot of mean displacement from the requested point (black line) and the standard deviation (red ellipse) for each point and projection location.

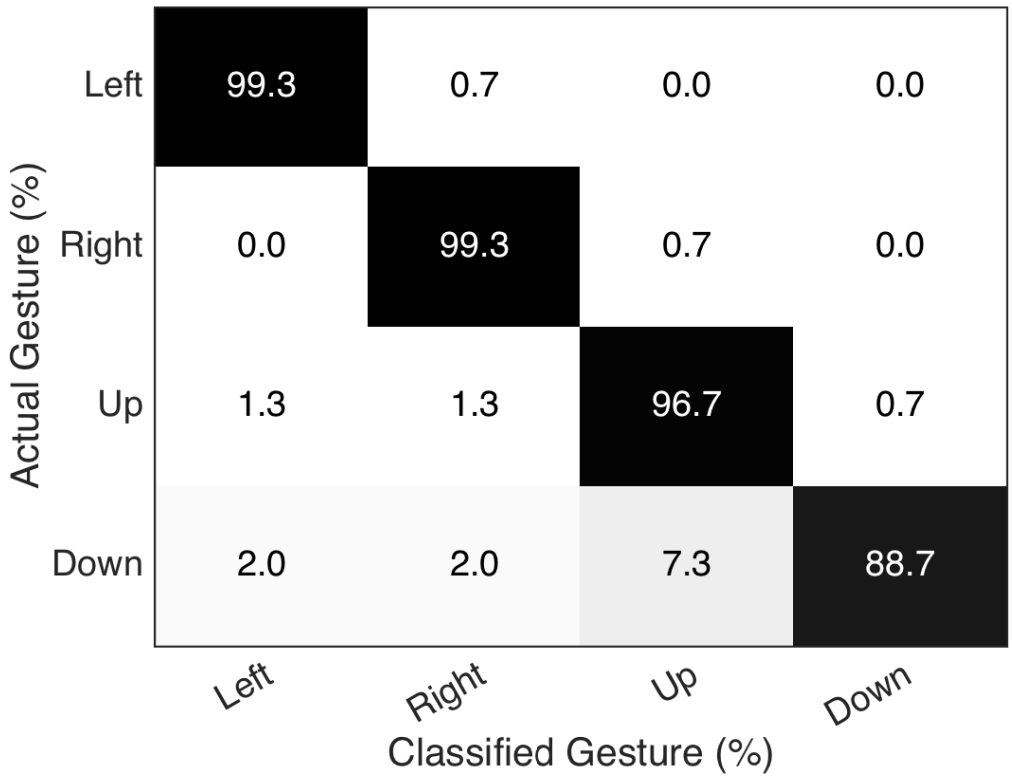

Table 7.1. LumiWatch Swipe classification confusion matrix.

Swipe Accuracy Results

For swipes, the detection accuracy was 100%, with an average swipe-direction classification accuracy of 96.0%, (detailed breakdown offered in Table 7.1). The biggest source of error was down-swipes being misclassified as up-swipes. From observing participants during the study, we suspect this is due to users navigating their finger to the start of a down-swipe too close to the skin, causing the pre-gesture motion to be classified as a swipe itself, leading to an incorrect classification.

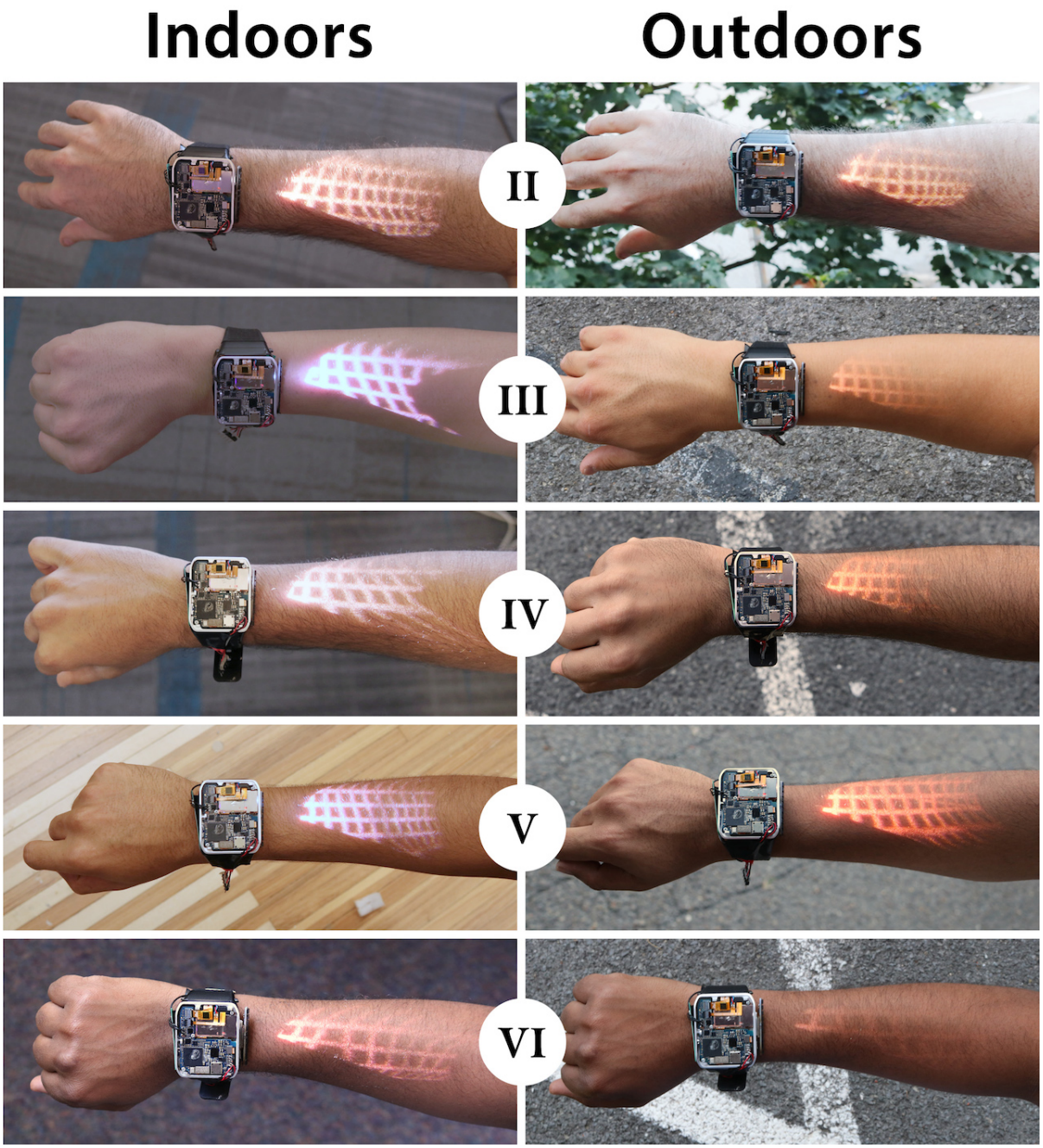

Figure 7.7. Results from the projection performance study.
Roman numerals indicate Fitzpatrick skin type.

7.7.2 Study 2: Projection Performance

Skin color significantly affects projection visibility due to variable absorption of light. In our second study, we wanted to evaluate projection performance across skin tones and lighting conditions. We recruited five participants, one for each skin type from II through VI on the Fitzpatrick scale [Fitzpatrick 1975], with varying levels of hair (Figure 7.7). Each participant was asked to wear the watch on their right arm for consistency (projecting onto the top of the arm), and then swipe the unlock interface to present a rectified 10 mm grid. Notably, the arm calibration used for this study was the same for all participants, *i.e.* no per-user arm model was employed. To evaluate the projection quality under different lighting conditions, this process was performed once indoors under typical office fluorescent lighting, and once again outdoors on a sunny day.

Visibility

Photos of each participant and condition are shown in Figure 7.7. Projections were clearly visible indoors on all skin types, and projections were generally visible outdoors (with the exception of skin type VI). All projected interfaces were hard to see in direct sunlight (also true of phone screens).

Rectification Accuracy

To evaluate the accuracy of our rectified graphics, we took photographs and *post hoc* fit lines to the projected grid. We first computed the angle of each line relative to the watch, comparing it against the intended axes. On average, grid lines deviated from the ideal by a mean absolute error of 10.7° (SD=8.0), with a 95th percentile deviation of 29.0°. For axially-aligned grid lines, the mean absolute deviation was 3.7° (SD=3.2) with a 95th percentile deviation of 8.4°. Several of our grids show a slightly "skewed" projection (*e.g.*, Figure 7.7, III indoors) due to twisting of the watch face, resulting in higher error for tangential lines.

Scale Accuracy

To assess scale accuracy, we measured the size of each projected grid square, and compared it with the intended size of 10×10 mm. In the axial direction, the mean absolute error was 22.0%

(SD=19.4), with a 95th percentile error of 57%. In the tangential direction, the mean absolute deviation was 13% (SD=9.4), with a 95th percentile error of 33%.

7.8 Interactions

We developed a simple APIs that allows developers to treat projected arms as conventional touchscreens. Developers author their interfaces using a fixed DPI (2 pixels per mm in our current implementation), and the resulting imagery is transferred to a texture and rendered onto the arm in a rectified manner. Due to varying arm angles, the available surface area might change from session to session; our API can provide the extent of the projected area so that applications can adjust their appearance accordingly. Touch and swipe events are also provided by the API via callbacks, and applications can additionally access the raw finger position for more advanced sensing purposes. These APIs enable our LumiWatch to readily support standard single-finger touch interactions on the arm.

7.8.1 Finger Taps

Our sensor array has a limited vertical field of view due to its orientation, causing it to only register fingers that are within ~1 cm of the skin surface. Thus, touching a finger to the skin and then lifting it more than ~1 cm off the surface is considered a tap-and-release gesture (i.e., a single finger tap). For example, a user can use this tap ("click") gesture to select and launch an application from the watch's home screen (Figure 7.1B).

7.8.2 Continuous Finger Tracking

Our sensor array provides continuous tracking of an in-range finger. Thus, the user can provide continuous positional input by moving their fingertip across the skin. For instance, users could use to draw (Figure 7.4), perform stroked input (*e.g.* Graffiti [MacKenzie 1997] or Unistroke Gestures [Goldberg 1993] for text input), or adjust a continuous variable (*e.g.*, slider or knob; Figure 7.1C). Continuous input also lends itself to common, single-touch motion gestures, such as panning a map, scrolling a list, or swiping to select/dismiss an item (Figure 7.1D).

7.9 Discussion & Limitations

7.9.1 Scanned-Laser Projector

The primary advantage of our scanned laser design, over similar small-form-factor designs such as LCoS or DLP, is that the light source is fully redirected towards projection. In LCoS or DLP systems, a large percentage of the light source is redirected or absorbed to produce black or dark imagery. By using a scanned laser, we save power and simultaneously enable brighter images for a fraction of the lumens – our internal tests suggest that our projector is equivalent to about 100 lumens of DLP projection for typical content (*e.g.*, videos). Scanned-laser designs are also focus-free (*i.e.*, in focus at all distances), which simplifies the optics, allowing our design to be compact.

However, a major drawback of scanned lasers is laser safety. Large projectors do not use scanned lasers because the laser power would exceed safe limits in the event of direct retinal contact. A commonly cited analysis of picoprojector laser safety [Buckley 2010] finds 17 lumens to be the upper bound for class 2 safety. However, a later analysis [Frederiksen 2011] suggests that existing pulsed-laser analyses are too conservative for scanned laser projectors, and proposes that projectors up to 39 lumens could be safely classed as class 2. The IEC, which oversees laser safety standards, has yet to offer guidance specific to scanned laser projectors, and so we use a more conservative value of 15 lumens in our prototype. In the future, as the laser safety issues are better understood and regulated, higher-power designs may appear, which will provide better usability in a wider array of conditions.

7.9.2 Heat Dissipation

The inherent small size of smartwatches limits their heat dissipation capability, which causes our watch to heat up considerably during projection. To avoid damaging the components, we set a strict 65°C limit on internal component temperatures before shutting down the projector. In practice, this limit can be reached within minutes if a full-brightness white image is displayed continuously.

Vents and fans are a common solution to this problem, but cumbersome in a small and energy-limited form factor. There are a few potential strategies to improve heat dissipation. First, our current design dissipates very little heat at the watch-skin interface, as we placed the battery at the bottom of the watch body. A future design could incorporate a metallic case thermally coupled to the logic board and projector, which could dissipate some heat to the wearer. A second, more radical possibility is to redesign the watch as a wristband, with hot components better distributed, and also using the watch and straps as heat sinks. This would provide a larger surface area for heat dissipation (both to skin and air), though it increases manufacturing complexity.

7.9.3 Irregular Usable Area

Our projected area has an irregular shape, which makes developing applications more challenging than traditional rectilinear screens. Moreover, the farthest extent of the projection has limited resolution and brightness, complicating application development further. Currently, the projection resolution on the axial axis is around 17 px/mm near the wrist (30 mm from the projector), but only around 1.8 px/mm toward the elbow (130 mm from the projector). This loss of resolution implies a corresponding decrease in projection brightness. Ideally, when projecting on the arm, we would drive the mirrors at variable rates to regularize the resolution and brightness across the projection area, a challenge we leave to future work.

On a more minor note, our current design places the projector towards one corner of the watch for logic board design and heat dissipation reasons. This creates a slightly pointed projection, as opposed to trapezoidal (as would be seen if the projector was centered on the arm). Again, this introduces a layer of complexity that must be dealt with at the application layer.

7.9.4 Projection Angle

Many of our early designs used a projector that was much lower to the arm's surface (~8 mm), but we found that this produced poor results on hairy arms as a large fraction of the light was lost before illuminating the skin. Furthermore, the hairs themselves become illuminated and interfere with content rendered on skin below. This suggests that there is an ideal minimum height

for the projector (roughly 10-14 mm), below which projection may simply be impractical no matter how good the projection technology is. Additionally, a shallow angle of projection also results in occlusions (shadows cast by fingers) whenever a user touches the arm, which is unavoidable in this form factor.

7.9.5 Touch Sensing

Our current touch sensing approach only provides accurate finger position estimates in a rectangle bounded by our 1D sensor array's width. Outside of this region, a finger will not be in line with any sensor, and instead lies simply within its cone of sensing. This causes our algorithm to judge the finger to be further away than it actually is, reducing positional accuracy. One possible solution would be to use slightly bowed sensor array, providing not a rectangular sensing area, but a cone, better matching the geometry of the arm, though this would be harder to fabricate. A second issue is that our current approach uses ten discrete time-of-flight sensors, which are relatively large and expensive (compared to other commodity components). Therefore, we plan to explore if the number of sensors can be reduced with minimal impact to accuracy.

7.10 Conclusion

In this chapter, I described LumiWatch, a first-of-its-kind smartwatch with integrated projector and touch sensing in a commercially viable form factor. Developing this prototype required solving a number of difficult problems, including development of a suitable projector module, shallow-angle projection onto curved arms, and accurate 2D finger tracking. Through a combination of custom hardware and software, this prototype smartwatch provides a large touchscreen-like interface directly on a wearer's arm. Providing touch sensing capabilities on the arm is yet another way in which on-world sensing can be achieved. Furthermore, the various components of LumiWatch, such as the miniaturized computer, projector and touch sensor, are necessary ingredients in realizing ubiquitous on-world interaction.

CHAPTER 8. ON-WORLD INTERACTION IN AR

Projecting virtual interfaces onto physical environments is, by definition, a form of mixed-reality – a way of mixing the virtual and physical domains. Recently, the subject of heads-up mixed-reality, in which users wear a display that overlays virtual content onto their perceived environment, has gained significant prominence, with commercial devices such as the Microsoft HoloLens and Magic Leap gaining traction. The head-mounted display form factor also offers a promising avenue to achieving on-world interfaces, and merits exploration as an alternative embodiment.

Mixed-reality devices allow users to superimpose realistic, 3D content directly over the environment, enabling a wealth of interactive possibilities. However, current-generation augmented mixed devices typically provide only in-air gestures and voice commands for input. Neither modality provides the precision or tactile sensation present with common computer input systems, including touchscreens and mice. Further, both modalities are fatiguing to use for long periods of time, due to the gorilla arm effect for in-air gestures and the repetitive, unnatural nature of speech input as a control interface. These limitations present a significant obstacle to the use of augmented reality in domains such as 3D object design, architecture, engineering, and art. It is difficult to imagine, for example, working on a building design for an entire day using only voice commands and gestural input, even given the advantages of augmenting reality with 3D visuals.

In this chapter, I describe MRTouch, a novel multitouch input solution for head-mounted mixed reality systems built in collaboration with Microsoft. MRTouch enables users to reach out and directly manipulate virtual interfaces affixed to surfaces in their environment, as though they were touchscreens. Touch input offers precise, tactile and comfortable user input, and naturally complements existing popular modalities, such as voice and hand gesture. The research prototype combines both depth and infrared camera streams together with real-time detection and tracking of surface planes to enable robust finger-tracking even when both the hand and head are in motion. The prototype is implemented on a commercial Microsoft HoloLens without re-

quiring any additional hardware nor any user or environmental calibration. Through a performance evaluation, I demonstrate high input accuracy, with an average positional error of 5.4 mm and a minimum button size of 16 mm for 95% accuracy, across 17 participants, 2 surface orientations and 4 surface materials. Finally, I demonstrate the potential of our technique to enable on-world touch interactions through 5 example applications.

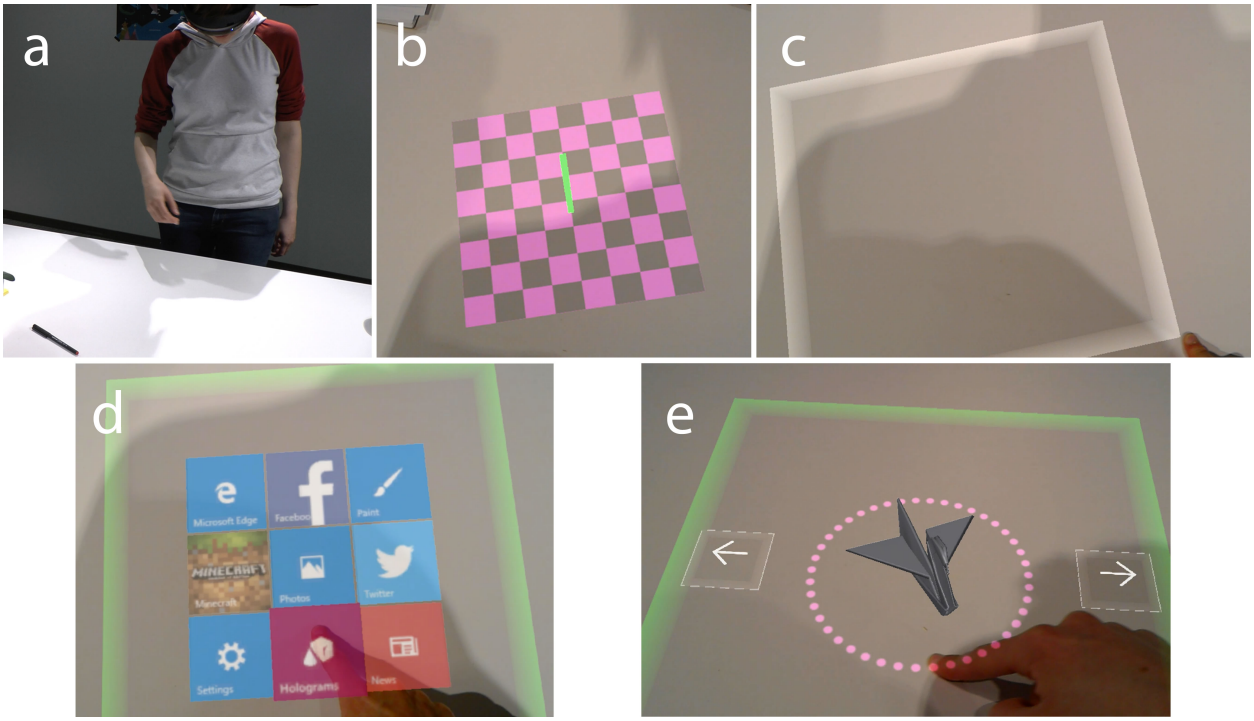

Figure 8.1. MRTouch enables touch interaction in head-mounted mixed reality.
(a) When a user approaches a surface, MRTouch detects the surface and (b) presents a virtual indicator. (c) The user touch-drags directly on the surface to (d) create a launcher and start an app. (e) In this app, the user uses touch to precisely rotate a 3D model.

8.1 Introduction

Head-mounted, mixed-reality devices can overlay realistic 3D content onto the physical environment, enabling a wealth of interactive possibilities. Current-generation commercial mixed reality devices, like Microsoft HoloLens or Meta 2, are primarily driven by in-air hand gesturing, gaze and voice. While convenient, these indirect input modalities are not particularly precise or rapid

[Cabral 2005, Chatterjee 2015]. Furthermore, these input modalities are fatiguing to use for long periods of time – neither gaze nor voice input are suitable for continual input while gestural input suffers from "gorilla-arm" effects [Hincapié-Ramos 2014], precluding prolonged use. In response, many systems ship with accessory physical controllers, offering finer-grained input, though at the expense of occupying the hands with a special-purpose device that then impedes free-hand manipulations.

On the other hand, touch interaction is precise, tactile, familiar to users, and comfortable to use for extended periods. However, interestingly, it has been overlooked as an input modality in head-mounted mixed reality systems, despite the inherent ability for many systems to affix virtual interfaces to physical surfaces (which are then *e.g.*, gestured at instead of touched).

To explore the feasibility and utility of enabling touch interaction in mixed reality settings, we developed *MRTouch*: a multitouch input solution for mixed reality. By overlaying appropriate virtual content, our approach transforms ordinary surfaces into expansive virtual touchscreens, enabling interactions with coordinated virtual content. MRTouch is implemented on Microsoft HoloLens and requires no additional tracking infrastructure or hardware beyond the existing onboard cameras – users need only put on the headset to obtain touch tracking capabilities.

8.2 Related Work

The most common intersection of touch input with mixed reality is for providing haptic feedback, which adds a sensation of physicality to virtual objects. To this end, a number of haptic interaction techniques have been proposed for mixed reality, including special-purpose haptic controllers (*e.g.,* articulated probes [Adcock 2004, Massie 1994], gloves [Benko 2005b, Burdea 1992], handheld devices [Benko 2016] or even body suits [Lindeman 2004]), in-air haptics [Carter 2013, Sodhi 2013] or robotically-presented textures [Araujo 2016].

For providing touch input, many systems use special-purpose touch input devices. Toucheo [Hachet 2011] augments a touchscreen table with a reflected stereoscopic display. In mobile augmented reality applications, touch input can be provided on the mobile device itself, providing a

form of indirect touch input [Hürst 2013]. With head-mounted displays, touchpads can be mounted on the side of the device or even across the entire front face of the device [Gugenheimer 2016] to enable indirect touch interaction. Alternatively, touch input can be provided on handheld controllers [HTC] or on a dedicated mobile device [Medeiros 2013].

More closely related to our conceptual approach, systems can also co-opt existing objects in the environment to provide input and haptic feedback. Vrui [Kreylos 2009] uses video pass-through to allow users to use real keyboards and mice to provide input to a virtual reality system. Haptic Retargeting [Azmandian 2016] uses perceptual warping in VR to allow users to grasp physical objects as proxies for virtual ones. [Walsh 2014] proposes projecting onto existing objects to co-opt them into a tangible user interface. Finally, Annexing Reality [Hettiarachchi 2016] opportunistically matches virtual objects with physical objects present in the user's environment, mirroring user's interactions with the physical objects onto the virtual objects.

8.3 Implementation

The MRTouch prototype is implemented on a Microsoft HoloLens development kit device, sold commercially for developer use since 2016. The HoloLens features a time-of-flight depth camera similar to the Microsoft Kinect for Xbox One [Microsoft Kinect], which can operate in both short-throw (maximum distance ~1 m) and long-throw modes, used for hand tracking and environment sensing, respectively. We use the short-throw mode exclusively, due to reduced noise characteristics.

HoloLens features a customized ASIC, dubbed the Holographic Processing Unit (HPU) used for depth data processing, hand tracking, localization and mapping, combined with a traditional Intel CPU running at 1 GHz with 2 GB of RAM. The HPU allows HoloLens to map the 3D environment and to provide a continual estimate of its pose and position within the space. However, unlike KinectFusion [Izadi 2011], the computed environment map is too rough to be directly useful for touch detection.

8.3.1 Software Architecture

MRTouch uses only the raw short-throw depth data and infrared imagery from the depth camera through a private API. The algorithm should therefore be portable to any device which provides a similar API, *e.g.* Google Project Tango devices. The tracking pipeline runs at 25 FPS and consists of three software components: 1) *Image Streamer*, 2) *Tracker Engine* and 3) *Client Library*.

Image Streamer extracts the raw depth data and infrared imagery from the on-board depth camera and exports it over a TCP socket, allowing us to stream data to either a Tracker Engine running locally on the HoloLens device, or to a GUI-enabled debug version of the Tracker Engine running on a connected desktop computer (shown in Figure 8.2).

Tracker Engine is implemented as a C++ program designed to run natively on HoloLens' Intel CPU. It receives depth and infrared imagery from Image Streamer (Figure 8.3a, b), and estimated head pose and position data from the spatial mapping API on the HoloLens device. Tracker Engine maintains a set of *known* touch surfaces in world coordinates. A touch surface is a 3D-positioned rectangle, represented uniquely by three 3D corner points (its bottom-left, bottom-right and top-left corners). On each frame, the tracker locates each known surface in the current depth frame, then acquires additional surfaces by fitting planes to the observed depth data. It then detects touch points over the visible planes, implements touch filtering and hover detection, and reports resultant surface plane and touch data to any connected client applications.

To receive touch data, a client application links against the *Client Library*, which is written in C#. This library opens a TCP connection to Tracker Engine, through which it receives plane position data and touch information. The library normalizes these positions into the application's coordinate system, and presents a simple event-based framework for consuming new plane and touch data. The client can also use the library to transmit new plane data to the Tracker Engine component, *e.g.,* if the user launches a new application on a surface (thereby locking the surface into the *known* set), or moves an existing in-air application onto a surface.

In the following sections, we detail the touch and surface tracking process performed by the Tracker Engine.

8.3.2 Surface Refinement and Detection

Many prior ad hoc touch systems first calibrate a background model of the environment, allowing them to perform a simple depth subtraction to extract hands and fingers. However, on a head-mounted display, the camera may be constantly in motion, precluding development of a simple background model. Furthermore, although HoloLens can determine its own pose and position in the physical world, the estimated position is not perfectly accurate, causing established planes to appear to shift in position relative to their physical substrates as the head moves around.

Consequently, the Tracker Engine component performs a depth-based refinement of each visible known plane, using the random sample consensus (RANSAC) algorithm [Fischler 1981]. This algorithm first projects the original plane into the depth image, producing an image mask. Each RANSAC iteration selects three random depth pixels from the mask and counts the number of pixels in the mask that approximately fit the resulting plane (the inliers). The final output is the plane with the largest number of inliers. This refinement procedure is robust to objects and hands above the touch plane, as these obstructions would be considered outliers and thus ignored.

Our procedure also fits a single *ephemeral* plane to each frame (Figure 8.2c), defined as the plane centered at the user's gaze center (if present). This is also achieved by using RANSAC, by selecting three points at random near the gaze center and selecting the plane that fits the most depth points across the entire image. Unlike known planes, the ephemeral plane is not stored – it is discarded after the touch tracking step is complete and recomputed on the next frame. The ephemeral plane is used to allow users to walk up and touch surfaces without previously defining touch areas, providing a way for users to instantly begin using a touch surface.

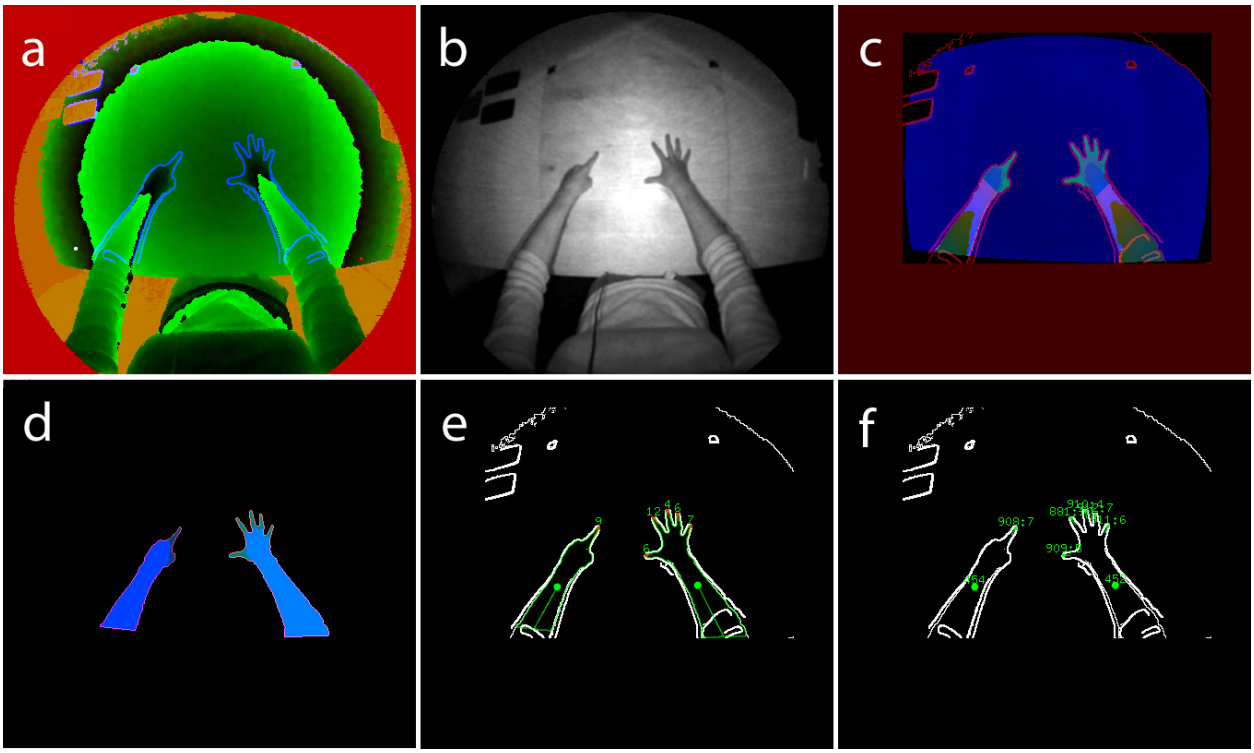

Figure 8.2. MRTouch touch tracking pipeline.
(a) depth data and (b) infrared reflectivity data from HoloLens depth camera. (c) detected ephemeral plane (blue), flood-filled hands (light blue) and fingers (teal). (d) extracted hand masks and contours. (e) hands, fingers and estimated finger distance from plane. (f) hands and fingers after touch filtering, with IDs corresponding to objects in past frames

8.3.3 Touch Finding

For each visible surface plane, including the ephemeral plane, Touch Engine initiates touch finding. The overall touch tracking approach merges depth data with infrared data, providing precise fingertip positions, an approach first demonstrated in DIRECT (Chapter 6). However, while DIRECT required a stabilized background and noise profile, our MRTouch implementation uses a RANSAC-refined touch plane, eliminating the prior profiling requirement and enabling true ad hoc touch tracking with no prior calibration of the surface.

Touch finding begins by re-projecting all depth pixels within the plane boundaries into *heights* relative to the surface plane (blue channel of Figure 8.2c). Then, the algorithm computes an *edge map* by merging a Canny edge map [Canny 1986] of the infrared image with a threshold-based edge map of the heightmap (a pixel is labelled as a depth edge if it differs from a nearby pixel by

more than 50 mm). Next, the algorithm flood fills pixels that are more than 40 mm off the surface, combining connected regions into tentative *high regions* (which typically includes arms, hands, and fingers that are far off the table). It then rejects high regions that do not contact the edge of the touch plane (*i.e.,* regions that lie entirely in the plane), as these are likely to simply be objects sitting on the surface.

From each remaining high region, the system fills downwards towards regions that lie closer to the surface, while respecting the edge map (Figure 8.2c, teal regions). The edge map ideally stops the flood fill at the user's fingertips, providing a clean segmentation even when the user's fingertips disappear into the background noise. If the flood fill doesn't stop (filling farther than a reasonable human finger length, 15 cm), this fill operation is rolled back, and the algorithm perform a more cautious fill that avoids filling into noisy background pixels (effectively returning to a depth-only tracking approach). On complex infrared surfaces, this ensures that the finger is still located, albeit with lower precision.

This step is followed by a smoothing procedure applied to the resulting hand+finger mask, and the conversion of this smoother mask into a contour map (Figure 8.2d). Finally, the algorithm walks across the contour, extracting convex points into fingertip points (Figure 8.2e). This contour-finding process ensures that the fingertip is extracted, even when the finger is very short (*e.g.,* if it is viewed from an oblique angle). Furthermore, as the segmentation is performed on the combined mask, this approach naturally segments hands and fingers in-air as well, enabling in-air hand gesture detection.

8.3.4 Touch Tracking and Filtering

Hand points (centroids) and fingertip positions, in world coordinates, are retained between frames to enable touch tracking and filtering. Hands from the current frame are matched to hands in the previous frame by Euclidean distance with a fixed upper limit on movement (*i.e.,* assuming hands do not move more than 10 cm in a single frame, or 2.50 m/s). Then, for each matched pair of hands, fingers are matched by applying the hand movement vector to the previous finger positions, then matching individual fingertips by Euclidean distance (again applying the

same upper limit on movement distance). By matching hands first, it is possible to track rapidly-moving hands without mixing up fingers. Figure 8.2f shows the final hand and finger positions.

To detect if the fingertips are touching or not, our algorithm analyzes a 7x7 patch of pixels centered on the fingertip's contour position. Each patch is split into *S*, the set of pixels within the hand+finger mask, and *T*, the set of pixels outside the mask. The estimated height of the finger is then given by

$$\max(z_s \mid s \in S) - \min(z_t \mid t \in T)$$

where the use of *max* and *min* help to stabilize the estimated finger and background distances against noise, as well as choosing points that have maximum discriminative power.

The hand position, fingertip position, and fingertip height values are smoothed using an exponentially-weighted moving average filter, providing resilience to noise and tracking failures. To confirm contact with the surface, the algorithm applies a simple pair of hysteresis thresholds – a fingertip is declared as touching the surface if the smoothed fingertip height descends below 10 mm, and declared to have left the surface if its height later ascends past 15 mm. This hysteresis approach mirrors the hover detection approaches seen in other systems, such as Wilson [Wilson 2007], OmniTouch [Harrison 2011] and DIRECT.

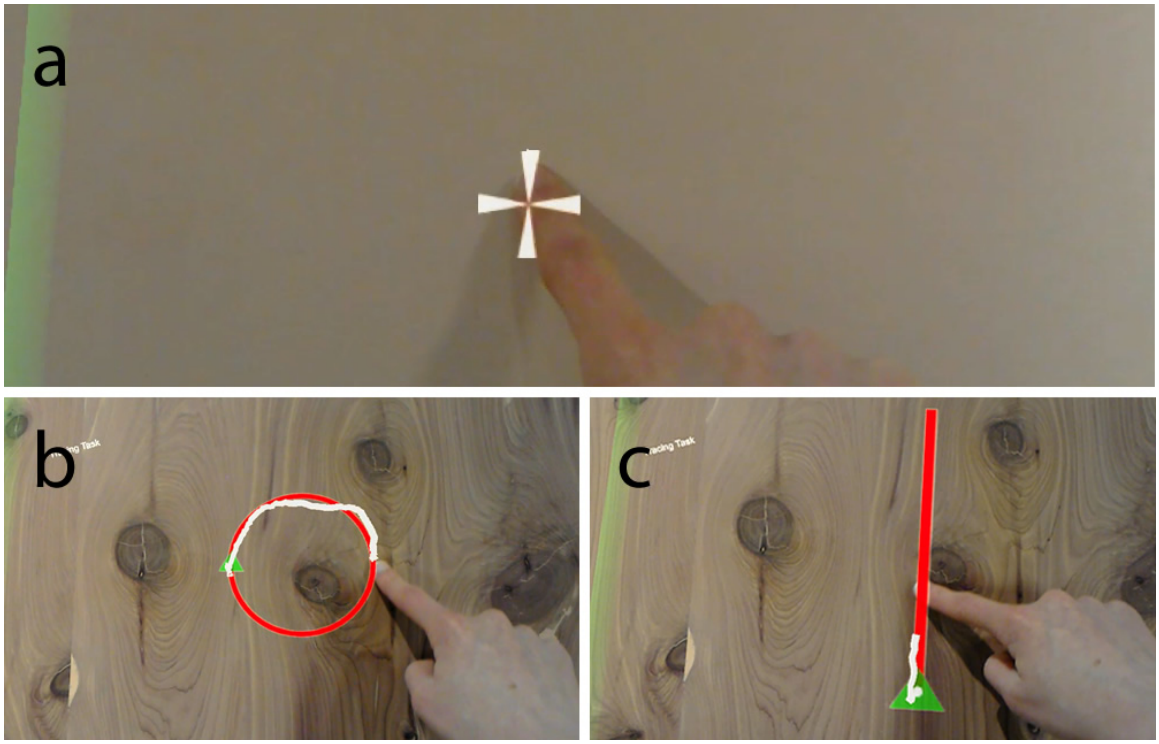

Figure 8.3. MRTouch experiment tasks.
(a) crosshair clicking task, (b) circle tracing task, and (c) line tracing task.

8.4 Experiment

To quantify the spatial tracking accuracy and performance of MRTouch, we performed a user study with 17 participants – 5 females, average height 68.8 in (174 cm), mean age 34, 2 left-handed, with Fitzpatrick skin types ranging from II to VI [Fitzpatrick 1975].

Participants were asked to perform a series of touching and tracing tasks on various surfaces in two separate orientations, to check the system's robustness to surface conditions and surface orientation. Users performed all tasks with their dominant hand, and held a wireless mouse in their other hand, which was used to advance trials.

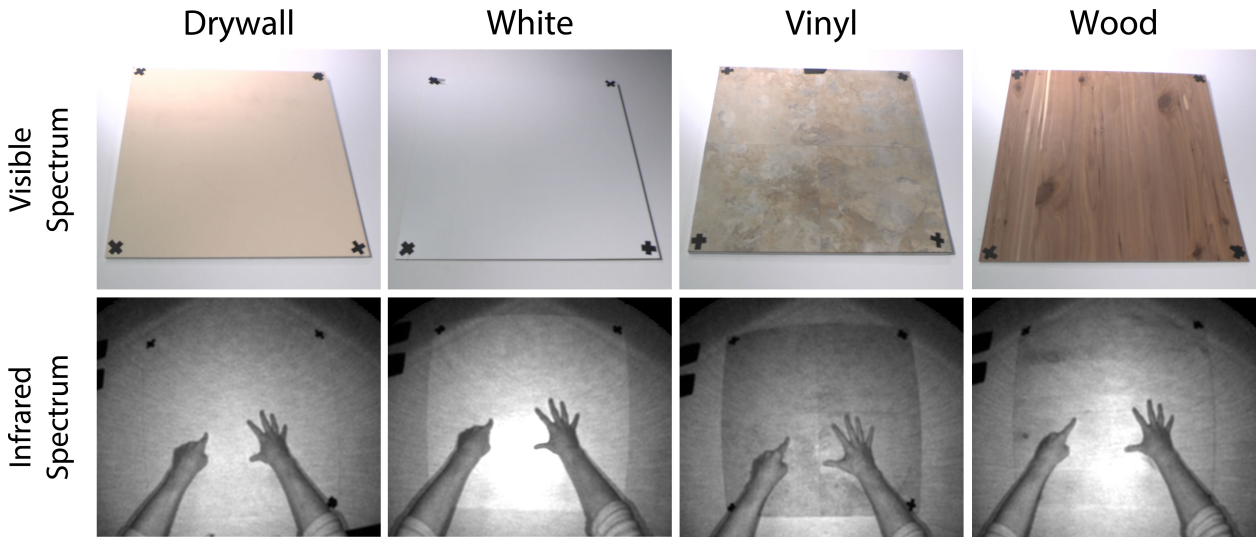

Figure 8.4. Surfaces used in the experiment.
User hands shown in infrared for contrast and comparison.

8.4.1 Surfaces

We chose a set of four surface types representative of common, everyday surfaces in homes and offices, as well as providing a range of different surface profiles in infrared:

Drywall: a piece of drywall painted uniformly with acrylic paint, representing most typical wall surfaces. Under infrared illumination, this surface appears relatively bright and diffuse.

White: a piece of medium-density fibreboard (MDF) coated with melamine, producing a smooth and glossy surface typical of contemporary tables, desks and cabinets. Under infrared illumination, this surface appears bright and specular.

Vinyl: a set of vinyl tiles with slightly rough exterior stone-like textures, used to represent stone and textured surfaces typical of countertops and floors. Under infrared illumination, this material is dark and diffuse.

Wood: sanded plywood with a cedar veneer, used to represent wooden surfaces typical of tables and desks. Under infrared illumination, this material appears darker, specular and non-uniform.

Each surface was cut to a 60×60 cm square tile, allowing them to be swapped in and out during the experiment. Images of these surfaces in visible light and under infrared are shown in Figure 8.4.

Prior to the study, we hypothesized that the darker, more complex infrared background of the vinyl surface would degrade our infrared tracking, and thus decrease accuracy.

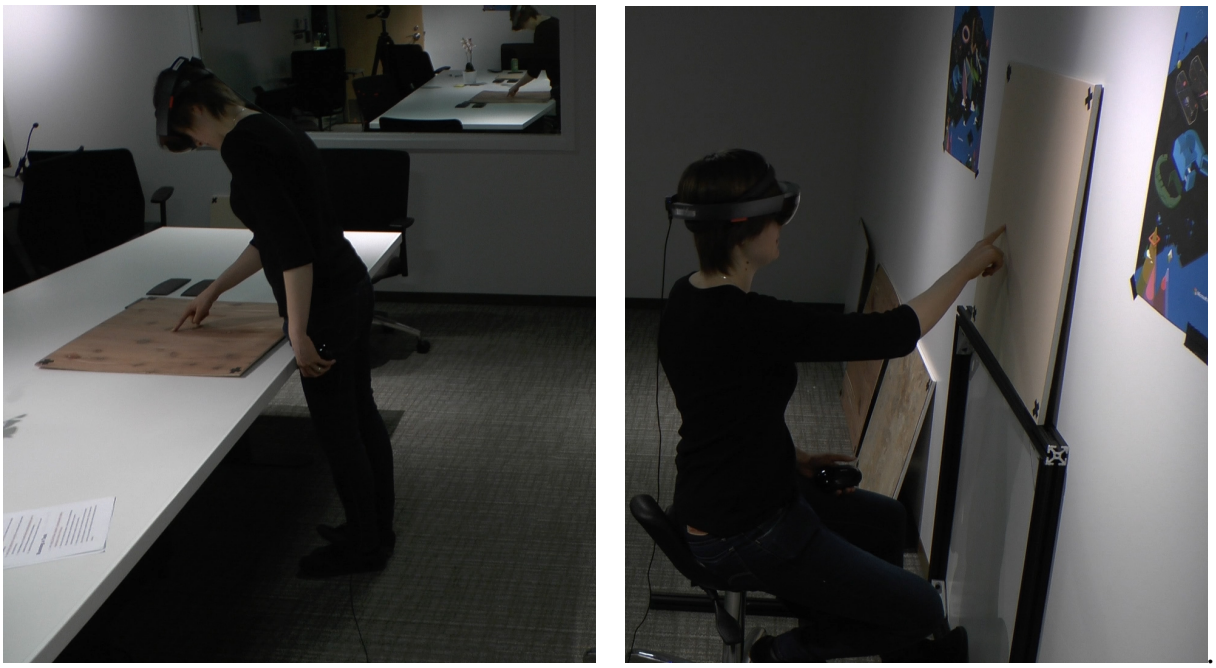

Figure 8.5. Surface orientations used in the study.
Left: Table orientation. Right: Wall orientation.

8.4.2 Orientations

We also tested the system in two common orientations: with the user seated and touching a vertically-oriented (wall-mounted) surface, and with the user standing and touching a horizontal (table-mounted) surface. In the Wall condition, the user sat on an adjustable chair, with the virtual surface centered at chest level (Figure 8.5, right), with the HoloLens at an average distance of 31 cm from the wall surface. In the table condition, users stood 6 inches from the table edge and looked down (Figure 8.5, left), averaging 51 cm from the HoloLens to the table's touch surface.

8.4.3 Tasks

For each combination of orientation and surface type (8 conditions in all), users performed two tasks: *crosshair targeting* and *tracing*.

The *crosshair targeting* task required the participant to place their fingertip on the surface corresponding to the displayed crosshair (Figure 8.3a), then click the mouse to confirm that they were touching the surface. The system recorded all detected touch contacts and their locations on click, allowing us to study the system's contact detection rate. There were 16 crosshair locations, placed in a 4×4 grid spaced evenly across a 20×20 cm square area. Crosshairs were displayed in random order.

In the *tracing* task, each participant was presented with eight shapes to trace: four circles (Figure 8.3b; each combination of clockwise/counterclockwise and 10 cm diameter/20 cm diameter) and four lines (Figure 8.3c; running left/right/up/down, all 20 cm long). A green arrow indicated where the participant should begin the trace, and the system automatically ended the trial when the participant completed the shape. Participants then lifted their fingers clear of the surface and clicked the mouse to start the next trial.

Prior to the main experiment, participants performed one trial each of the crosshair and tracing tasks to familiarize themselves with the experimental procedure; these training trials were not included in the final analysis.

8.5 Results

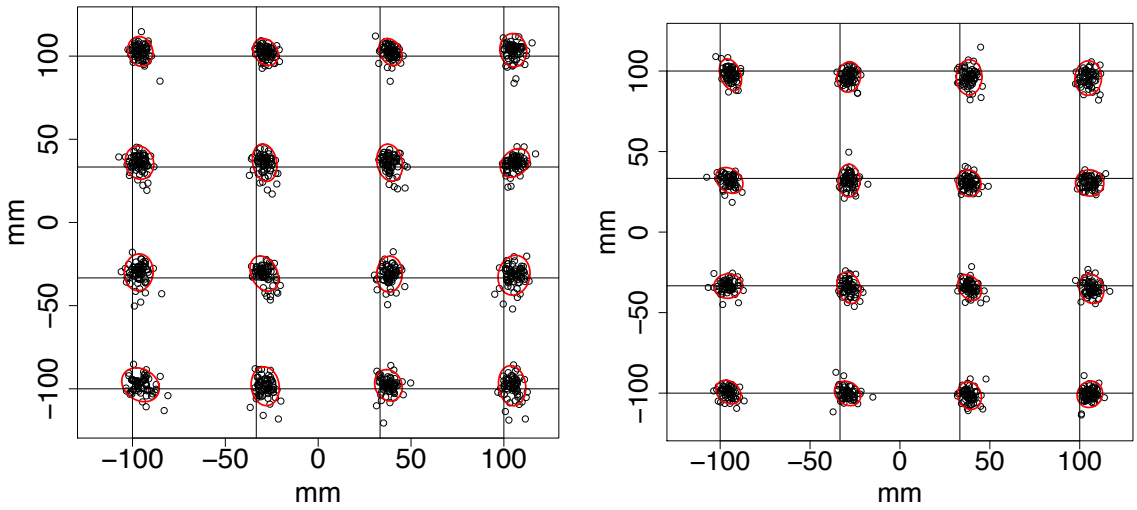

Figure 8.6 Scatterplot of touch points.
Left: *wall* condition; right: *table* condition. Red ellipses indicate 95% confidence region. Grid intersections correspond to crosshair positions.

8.5.1 Crosshair Targeting

Across 17 participants, we obtained 2176 crosshair trials, recorded at the moment that the participant pressed the mouse button. Of these, 77 (3.54%) reported no touch contact, so the system located at least one touch point 96.5% of the time. In 416 trials (19.12%), the system detected two or more touch contacts; of these, 49 (2.25%) found three or more touch contacts. In the event of multiple detected contacts, touch accuracy was computed using the contact nearest to the target.

The distributions of touch contact counts were significantly different (p<0.0001, ANOVA) between the Wall and Table conditions. In the Wall condition (1088 trials), only 2 trials (0.2%) reported no touch contact and 185 trials (17%) reported two or more contacts. In contrast, on the Table, 75 trials (6.9%) reported no touch contact, and 240 trials (22%) reported two or more

contacts. No significant differences (p>0.01, ANOVA with Tukey HSD) in touch contact count were found between materials.

Spatial Accuracy

Analysis of our crosshair data demonstrated a systematic offset between the target positions and the received touch position. Reported touch positions were an average of 5.0 mm to the right of the crosshair (Figure 8.6) with a negligible (<0.1 mm) vertical shift. This shift held constant across all orientations, materials, and user handedness, and thus, for further analysis, we subtracted the global average offset from all touch points. In practice, we would add this global offset correction as part of the touch detection pipeline, as it is user-, material- and orientation-independent. To enable direct comparison with the OmniTouch system, we followed the same data analysis procedure as Harrison et al. [Harrison 2011] and removed 44 outlier points (2.0%) which lay more than three standard deviations from the target point. Across all remaining points, we achieved a global mean Euclidean error (μ) of 5.4 mm (SD=3.2 mm).

Spatial accuracy showed significant (p<0.0001, ANOVA) differences between the Wall (μ=5.7 mm, Figure 8.6, left) and Table (μ=5.0 mm, Figure 8.6, right) orientations. In terms of materials, pairwise Tukey HSD comparisons showed that the Vinyl material (μ=6.2 mm) was significantly less accurate than all other materials (Drywall μ=5.0 mm p<0.0001, White μ=5.3 mm p<0.0001, Wood μ=5.0 mm p<0.0001), but no other significant differences were found (p>0.01).

Plotting the 95% confidence ellipses for the Wall and Table conditions (Figure 8.6) shows that, on average, a 16 mm diameter button would capture 95% of touches, which compares favorably with the 15 mm button diameter for capacitive touchscreens found by Holz, *et al.* [Holz 1997].

8.5.2 Tracing

For our shape tracing task, we continually computed the absolute Euclidean distance between each incoming touch point and the nearest point on the shape. Because graphical feedback was provided, we did not apply any post-hoc offset correction, reasoning that users would naturally adjust their motions to compensate for any observed inaccuracies. The mean Euclidean distance

across all users, conditions, and orientations was 4.0 mm (SD=3.4 mm), with no significant differences across shapes, materials or orientations.

8.5.3 Latency

A static analysis of latency in our research prototype (as determined through timestamping various stages in the pipeline) suggests an expected latency between 105~175 ms, or a mean latency of 140 ms, from physical touch event to the first displayed frame incorporating the event. However, due to smoothing factors, the observed latency is somewhat higher, at around 180~200 ms in empirical testing.

8.5.4 Comparison with Existing Approaches

In terms of spatial accuracy, our results compare favorably to past ad-hoc touch tracking approaches. Wilson [Wilson 2007] does not provide a formal evaluation of spatial touch accuracy but suggests a positional error of 7 mm at a sensor distance of 75 cm from a fixed, pre-calibrated surface. KinectFusion [Izadi 2011] demonstrates but does not formally evaluate the touch tracking methodology. DIRECT demonstrates 4.8 mm average Euclidean error, albeit with a significantly different setup over a rigidly pre-calibrated table surface.

Finally, OmniTouch [Harrison 2011] shows an average positional offset of 11.7 mm (dependent on surface orientation), compared with our 5.0 mm global offset. OmniTouch also plotted the 95% confidence ellipses to evaluate its accuracy; in the best condition (touching on a wall), the confidence ellipses measure an average of 28 mm in diameter (compared with MRTouch's 16 mm buttons).

8.6 Discussion

8.6.1 Click Detection

Our crosshair study demonstrated a surprisingly high rate of both missed touches (3.5%) and spurious extra touches (19%). From reviewing the study logs and recordings, we determined that

poor hover sensing was primarily to blame for both issues. When we missed touches, the finger itself was often located but the finger's height did not cross our heuristic threshold. On the flip side, we observed that many participants held their thumbs out while performing the crosshair task, which were correctly detected as an additional finger. However, in a few trials, the thumbs were detected as touching the surface. This often happened when participants unconsciously brought the thumbs close to (but not touching) the surface during the experiment, and the lack of visual feedback meant that people were unaware that the system was detecting a contact event.

Contact detection was not the primary subject of our initial MRTouch implementation; instead, we sought to tackle the orthogonal problem of spatial accuracy. Because divining the finger's distance from the surface is difficult solely from an overhead view, we felt that solving the contact detection problem was out of scope for this work, and thus chose a hover detection implementation similar to those used by other ad-hoc touch tracking systems we reviewed. However, we expect that more sophisticated finger height estimation algorithms could produce better results in the future, and we look forward to exploring this issue further.

8.6.2 Latency

Latency is a crucial consideration for touch input. While our system latency of ~180 ms is comparable with the higher end of touchscreen latencies (50~200 ms, per [Ng 2012]), people can notice latencies as low as 20 ms [Jota 2013]. The latency problem is often exacerbated on large displays (*e.g.,* wall-sized displays) because sensing larger displays typically requires more processing time, and simultaneously users can move much faster across the display (increasing the observed gap between physical actions and virtual responses). Thus, for larger displays, people may be even more sensitive to latency issues.

We emphasize that our system latency is largely a prototyping limitation. Most of the HoloLens' existing tracking algorithms run on the dedicated HPU ASIC, which has very low latency access to depth data combined with high input priority (*i.e.* low-latency access to the system input event queue). Our prototype does not currently run on the HPU, but rather on the HoloLens' CPU. We

estimate that the latency for an HPU implementation would average around 60 ms (depth camera average latency + touch pipeline processing time + rendering average delay + display latency), which is comparable to high-end commercial touchscreens.

Decreasing latency is thus a major topic of future work. Besides an HPU implementation, we could also explore techniques such as forward prediction to forecast the user's finger position and pre-touch [Hinckley 2016] to predict when the user will contact the surface.

8.6.3 Effects of Orientation

We found significant orientation-dependent effects on detected touch count. In our study, participants sat much closer to the Wall (31 cm) than the Table (51 cm). Our analysis of the data suggests that the longer Table distance increased depth noise but not infrared noise, causing more touch contacts to exceed the predefined hover thresholds, resulting in reduced touch detection accuracy.

Although the Wall orientation was significantly less accurate than the Table orientation (5.7 mm vs 5.0 mm), this difference is relatively slight, suggesting that the use of the infrared map helps ensure a high level of spatial accuracy. Future work will need to account for varying depth sensor noise levels in order to provide accurate click detection.

8.6.4 Effects of Material

From our study, we found that touch tracking had significantly worse spatial accuracy on the vinyl material. This may be due to a combination of surface roughness (which increases depth noise) and dark infrared profile (which reduces contrast with the hands). However, even on that material, we still achieve a high degree of spatial accuracy relative to prior work.

In general, some materials will perform worse than others, and in the extreme case, some materials will not work at all. For instance, we initially considered a fifth material, MDF coated in black paint, which absorbed a significant amount of infrared radiation (*i.e.,* it is black in the infrared

spectrum too). We were unable to use this material for the study, though, because the low reflectivity prevented the depth camera from even sensing the surface. Furthermore, transparent surfaces (*e.g.,* glass) will likely never be supported by depth-based touch tracking, necessitating some means of distinguishing suitable from unsuitable surfaces and conveying this in the interface so that the user's expectations are properly set.

The infrared reflectivity of a surface is likely to be a strong predictor of its ability to support MRTouch interactions – for instance, dark or highly patterned surfaces may not support interactions as well. Therefore, at a distance, we should be able to determine how well touch tracking will work before users attempt to interact with the surface, and we can present feedback to users to allow them to choose an appropriate surface. For example, we could choose to only highlight suitable surfaces with a surface indicator (Figure 8.1b), so that users would learn to associate the indicator with the ability to perform touch interaction.

8.7 Mixed-Reality Touch Interfaces

8.7.1 Creating Touch Interfaces

When a person walks up to a surface that is not hosting an existing interface, a visual representation of the computed ephemeral surface plane (Figure 8.1b) is shown to indicate that the surface supports touch interaction. Inspired by OmniTouch [Harrison 2011], the user can simply touch the surface to begin defining the interactive region. As the user drags their finger along the surface, the system displays a proposed rectangular interactive area on the surface (Figure 8.1c) oriented according to the user's point of view. After the finger is released, the interactive region instantiates with an application menu (Figure 8.1d), from which the user can select an app to run on that region (Figure 8.1e). In a practical implementation, a delimiting input (*e.g.,* a voice command or unique hand gesture) could be added before the initial drag input to avoid inadvertent input on inactive surfaces (the Midas touch problem).

8.7.2 Standard Multitouch Interaction

Applications running on the surface support standard single-touch and multi-touch inputs, such as clicking, panning and zooming. Furthermore, the ability to support standard multitouch input opens the possibility of running traditional tablet PC applications within virtual touchscreens (*e.g.,* a document reader or web browser), greatly expanding the number of available applications.

8.7.3 Interactive Control of 3D Objects

Touch input can also be used to control 3D objects, *e.g.* adjusting rotation, size and scale, by manipulating appropriate on-surface control elements (as proposed in [Benko 2005b]). In our example, the person can select a 3D object in their view, dock it by moving it closer to the surface, and then use on-surface touch controls (*e.g.,* a rotation dial, Figure 8.1e) to manipulate the object. Crucially, the precision of touch input allows users to make fine adjustments to objects.

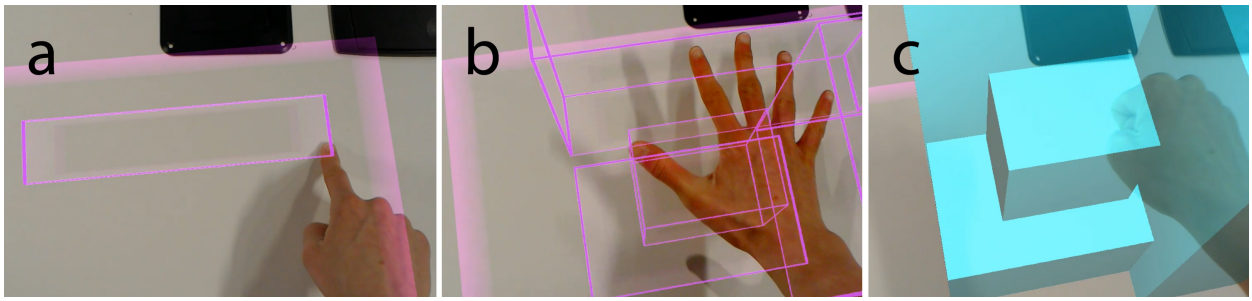

Figure 8.7. Blueprint extrusion app.
(a) The user draws a blueprint with their finger. (b) They use an in-air open-hand gesture to extrude it, using hand height to set the extrusion distance. (c) The user closes their hand to lock-in the extrusion.

8.7.4 Combining Touch Interaction and In-Air Gestures

MRTouch naturally supports finger input both on and off the surface, making it possible to combine touch interaction on a surface with gestures performed above it. For instance, after drawing the blueprint for a building using touch input (Figure 8.7a), a user can use an in-air gesture (Figure 8.7b) to adjust its *z*-height, thus naturally extruding a 2D cross-section into a 3D volume (Figure 8.7c). Many more interactions are possible; see *e.g.,* [Benko 2005b, Chen 2014] for an exploration of mixed-modality gestures.

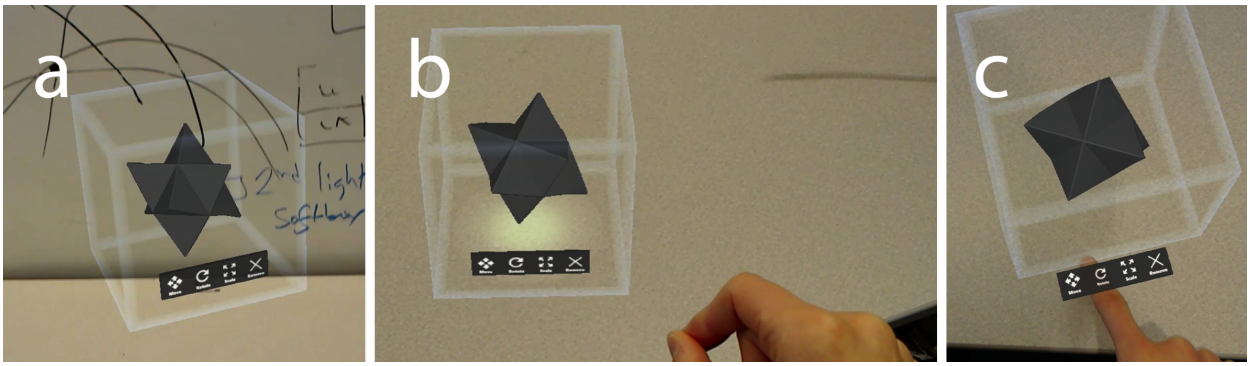

Figure 8.8. Snapping interfaces to touch surfaces.
(a) AR apps often float in mid-air with gaze/gesture-input buttons. (b) With MRTouch, the user can move the app near a surface, which highlights in response. (c) When the user lets go, the interface and toolbar snap to the surface, enabling touch interaction.

8.7.5 Converting In-Air Interfaces into Touch Interfaces

HoloLens applications often feature menus or toolbars, which users ordinarily interact with using head gaze and hand gestures (Figure 8.8a). With MRTouch, a person can move such an application onto a surface (Figure 8.8b), whereupon the in-air menu transforms into a touch-enabled toolbar (Figure 8.8c). By enabling a seamless transition between on-surface and in-air modes, a person can choose an interaction modality that suits their work. Furthermore, because of the higher precision of touch input, the on-surface mode could be used to surface additional functionality, *e.g.*, expanding a simple palette of image operations into a Photoshop-like toolbar.

8.8 Conclusion

In this chapter, I have described another embodiment of the on-world interaction paradigm. MRTouch brings un-instrumented touch input, on a wide variety of common surfaces, to head-mounted mixed reality. It demonstrates that future on-world systems need not confine themselves to projection alone – the rich domain of immersive augmented reality synergizes remarkably well with ad-hoc touch input and on-surface display. Further, the unique advantages of a head-mounted display – portable, capable of projecting 3D content, and private – make it a compelling application domain for on-world interaction.

CHAPTER 9. CONCLUSION AND FUTURE WORK

On-world interactive systems allow computing to escape the narrow confines of screens and devices, and out onto the wider world around us. Enabling such rich interactions on the environment requires solving a bevy of challenges, from the technical questions of input/output, to the design questions of interaction techniques and capabilities, to the overarching considerations of integrating and designing complete interactive systems. Through the past eight chapters, I have explored solutions to each of these areas, culminating in a body of work that I believe significantly advances and defines the state of the art in on-world interactions. We started, in Chapter 3, with an exploration of the prior pieces that led me to explore on-world computing. In Chapters 4 through 6, we systematically built the technical and interactive components necessary for functional on-world computation. Finally, in Chapters 7 and 8 we explored complete embodiments of the on-world paradigm, demonstrating that these types of systems are well within reach. In this final chapter, I conclude by summarizing the major contributions, and highlighting promising avenues for future exploration.

9.1 Summary of Contributions

9.1.1 WorldKit – On-World Projection and Touch Sensing

In Chapter 4, I introduced WorldKit, which paired a projector with a depth sensor to bring interactivity to the environment. WorldKit provided a number of technical innovations – familiar programming abstractions to make on-world programming simpler and more intuitive, behind-the-scenes projection rectification to display proper, flat graphics on any surface, and depth-based contact tracking to detect user interactions.

Furthermore, WorldKit broke new ground in terms of interaction, bringing this variety of technical capabilities together into a highly accessible form. It enabled easy and familiar programmatic access to advanced sensing and projection capabilities, and provided a new on-world user experience based on the dynamic instantiation of interfaces *when and where they are needed*.

This was facilitated by the use of a simple "painting" gesture which was used to position, size and define interactive components of an application on the world, allowing users to seamlessly map an application's interface elements onto their environment. WorldKit also explored the power of on-world touch input and suggested that touch input could be a primary way to interact with interfaces. Finally, WorldKit demonstrated that this type of system was achievable with commodity off-the-shelf technology (an ordinary projector and a Microsoft Kinect gaming sensor), rather than requiring specially-instrumented and designed spaces (*e.g.* CAVEs). Furthermore, it demonstrated that the programming model for such a system could strongly resemble standard desktop GUI programming, making it much easier for developers to transfer their existing knowledge to this new domain. These insights carried forward to the remaining works described in this thesis.

9.1.2 Desktopography

With Desktopography in Chapter 5, the key focus was to define a set of interactions that users expected to be able to perform on on-world interfaces. By running an elicitation study to explore user expectations and insights, we were able to distill down to a common set of operations. Alongside traditional, virtual GUI interactions such as touch interactions, resizing, moving and closing, we also found interactions between physical and virtual objects, such as snapping, evading, and collapsing, which have not been previously explored in the literature. These interactions exposed the similarities and differences between in-screen and on-world interaction styles and highlighted the importance of having on-world digital interfaces cohabit properly with in-world physical objects.

To realize the system as a fully-functional prototype, I refined WorldKit's programming model to make it possible to run ordinary, unmodified applications on the world, supplanting the need for customized programming interfaces. Desktopography also featured a sophisticated set of technical features necessary to implement the on-world interactive behaviours, encompassing dynamic environment sensing, desk surface mapping, object edge detection and a metaheuristic optimization-based layout engine to automatically place interfaces on the environment.

9.1.3 Refining On-World Touch Input

One of the major challenges highlighted by both WorldKit and Desktopography was the need for high-precision touch input. While this could be achieved by instrumenting the surface with sensors, and/or instrumenting the user with hand-worn gadgets, these sorts of instrumentation can be intrusive, costly and complex, and so I chose to tackle the problem of camera-based touch tracking. I built DIRECT, which significantly advanced the state of the art in camera-based touch tracking. DIRECT merged the depth and infrared data streams from a single depth sensor to provide superior finger detection and tracking, enabling high-precision touch input with individual fingers. With an average positional error of 4.9 mm, it became possible to truly treat surfaces as high-precision touch screens, and demonstrated a touch input accuracy and detection rate far superior to prior works in the area. DIRECT made it possible to convert an ordinary table into a large touchscreen canvas, with accuracy comparable to early touchscreens, without requiring any modification to the table nor anything worn by the user.

9.1.4 Miniaturization and Integration

From Chapter 5, we suspected that a compact form factor was important for wider adoption. Ideally, we would be able to create a tiny projector/computer/camera hybrid device which could be unobtrusively installed throughout an environment to bring every surface to life digitally. As no existing projector and camera pairing fit the bill completely, we turned to building the system ourselves. The result was a tiny picoprojector, small enough that we integrated it into a smartwatch. Combined with a proximity-based touch sensor, we were able to create a functional touchscreen interface directly on a user's arm – a different sort of "on-world" interaction.

LumiWatch consists of a tightly-integrated and fully self-contained smartwatch, comprising a battery power source, 15-lumen scanned laser projector, 10-element time-of-flight depth sensing array and a quad-core CPU running Android. Together with the custom projection correction and touch sensing algorithms developed specifically for this project, LumiWatch transforms the wearer's arm into a 40 cm^2 touchscreen surface – over 5x the area of a traditional smartwatch's physical display. The hardware design, projection rectification and touch sensing approach are

all novel components from a technical standpoint and will all be valuable components in any future compact on-world projection system.

9.1.5 On-World Interactions in Augmented Reality

In Chapter 8, I explored a fully-realized embodiment of the on-world interaction paradigm within head-mounted augmented reality. Supporting the desired style of touch interaction required developing novel solutions to several technical issues: developing a method for real-time detection of surface planes suitable for hosting touch-enabled applications, building a robust finger-tracking pipeline able to segment touch inputs even while the user's head is in motion (an evolution over DIRECT), a practical, self-contained implementation of this approach on an off-the-shelf Microsoft HoloLens, and a user study assessing the system's tracking accuracy which finds very high accuracy across a variety of materials and orientations. In addition to these technical improvements, the MRTouch project also presented a number of new interactions made possible by this approach. With MRTouch, a user simply touches the surface to define an interactive area and instantiate an application. MRTouch senses touch and in-air gestures over the application area, enabling highly flexible manipulation of 3D content. We demonstrate a few applications that make use of this powerful combination of precise touch input and expressive gesture input.

9.2 Future Work

Each piece of work that is completed opens up new research possibilities and questions. The field of on-world computing is vast, and the tools that I have built to explore the space will be of great help in navigating the work that is yet to come. In this section, I highlight and describe some of the most promising unanswered research questions which have the potential to further our understanding and our capabilities in the new on-world paradigm.

9.2.1 Improvements to Touch Tracking

The performance of the touch tracking algorithm is of paramount importance to the practicality of an on-world system. While DIRECT and MRTouch demonstrated significant strides over past

work, there is still significant room for improvement. Ultimately, I believe the goal should be nothing less than parity with physical, capacitive touchscreens.

The first major question concerns hover tracking accuracy. While DIRECT had high spatial accuracy, it is not as accurate when distinguishing an actual touch from a user's finger hovering over the surface. Although its touch detection performance is much improved over a typical infrared- or color-camera touch tracking approach, it still detects a touch when the user's finger comes within 5mm of the surface – in other words, a finger hovering this far over the surface cannot be reliably distinguished from a finger touching the surface. The reason for this ambiguity is two-fold: first, a depth camera sees only the top of a finger, not the bottom, meaning that it is impossible in a completely general sense to determine if a user is touching the surface or whether they simply have a thinner finger. The second issue is that depth cameras have significant noise, which is often exacerbated by placing the finger very close to the surface. Consequently, it can be very difficult to determine what the actual distance to the finger is.

There are several ways in which this hover disambiguation can be performed more accurately. Machine learning, for instance, presents a potential solution. By learning the difference between a touch and a non-touch using a small window of depth data surrounding the fingertip, it might be possible for a machine learning algorithm to automatically distinguish these states. In concert with this, it may also be possible to develop stronger heuristic methods to detect a touch, taking into account the finger's width (to infer finger thickness), the distance to the sensor, and the fitted plane model. Improving hover disambiguation directly improves the user experience, as they can more naturally lift their fingers between touches without worrying about the touch contact staying active.

The second major question asks how the touch latency can be reduced. Latency is an important factor in the user experience. High touch latency delays the user's input feedback cycle, which slows down all interactions and results in lower interaction bandwidth. On-world interactions have stricter latency requirements than e.g. mobile phone screens, because the larger surface area encourages faster motions. For a fixed touch latency value, the faster motions cause the

user's position to diverge more from the sensed position when compared with an equivalent interaction on a mobile phone. Furthermore, the low update rate of typical depth cameras (usually 30 fps) combined with their sophisticated, computationally-expensive depth processing stages produces high sensor latency, even before any touch processing takes place.

To alleviate these latency issues, we could explore *forward prediction* – the use of historical touch data to predict where the user's touch location will be in the future. Although this is not perfectly accurate, forward-predicted touch positions may help to reduce perceived latency, and therefore achieve the goal of shortening the input feedback cycle. Secondly, we could employ *pre-touch* techniques to predict when a user's finger will contact the surface, allowing the system to respond pre-emptively to touch events. Both techniques have been previously explored in the touchscreen literature, but have not been explored for on-world interactions where they could have greater impact.

The final question asks whether it is possible to perform on-world touch sensing at an even larger scale than DIRECT. While DIRECT successfully instrumented an entire table surface. I conjecture that it should be possible to instrument part, or even all of a room with a single sensor, minimizing the intrusiveness of the system and radically expanding the reach of the on-world interaction approach. However, the major issue that has to be resolved here concerns the spatial resolution of the depth camera, and potential noise effects from sensing touches at very long distances. If these can be solved, or worked around, it would significantly improve the practicality and simplicity of on-world computing.

9.2.2 Moving Beyond Touch

Technologically, it is possible to detect hands *above* the surface simply by computing the surface-relative height of incoming depth pixels. This in turn enables the system to mix in-air gestures and touch input. In-air gestures provide high levels of expressivity and natural 3D-space interaction, while touch provides haptic feedback, precise control and non-fatiguing operation. In-air gestures also form a natural "out-of-band" input modality with respect to touch input, allowing them to e.g. summon or activate interfaces which can then be touched. Therefore, these two

modalities naturally complement each other, making it possible to have an input system which offers both high expressivity and high precision. I have seen this winning combination in past research on in-air gestures for mobile devices (e.g. Air+Touch [Chen 2014]) and my own explorations on the topic (e.g. Gaze+Gesture [Chatterjee 2015]).

A large variety of hand gestures are possible and could be explored in future work. Hand gestures could be used to summon interfaces (e.g. a "launching" or "summoning" gesture to pull up an application list), to destroy interfaces (e.g. a "crunching up", "throw away" gesture), to resize and reposition interfaces, and various other interface management gestures. Within an application, gestures could be used to manipulate content (e.g. select, place or resize an object), with touch interactions used to refine the configuration. Or, an application might use hand gestures to implement simple discrete actions, such as controlling media playback or navigating back/forward in a web browser, with touch interactions used for more complex or precise manipulations like scrubbing, scrolling, text selection, and so on.

9.2.3 APIs & SDKs

Moving up one level, a major consideration for a new computing canvas is determining how developers will interact with it. One of the classic ways to obtain a large application library is to provide a mechanism for running existing applications. Our existing systems, such as WorldKit, DIRECT and Desktopography, all have the capability to display rectified graphics and receive multitouch input, making it possible in principle to run arbitrary touch-sensitive applications. Thus, it may be possible to develop a compatibility layer to run arbitrary mobile-optimized webpages on the world, effectively treating surfaces like giant tablets. This enables the execution of existing mobile webapps directly on the world, providing a breadth of existing functionality, as well as enabling the development of new on-world applications using well-known web APIs.

In mobile development, there is frequently a distinction between "web apps" and "native apps". Web apps have access to a subset of the device's functionality through the web browser, whereas native apps have access to the full breadth of functionality offered by the device manufacturer. Examples of functionality available to native apps, but not web apps, include access to certain

sensors (e.g. video camera, high-speed accelerometer data), access to more private data (such as contact lists, calendar appointments, etc.) and the ability to run in the background. Analogously, webapps in our on-world operating environment will have access to touch input and graphical output, but will not have access to data such as the physical environment geometry, objects present in the world, or execution context (e.g. which room, surface or space the application is running in).

WorldKit touched briefly on some of these issues, with the notion of "synthetic sensors" which can sense various aspects of the environment, and with a programming interface that provides this type of data. In future work, I could expand upon these early APIs to provide access to data like surface-rectified depth and colour data, recognized objects (e.g. recognized edges, per Desktopography), environment geometry (e.g. the surface orientation and height, to determine where the interface is relative to the user) and advanced user input data (e.g. hand gestures, finger orientation). This "world API" will enable the development of more sophisticated "native" on-world applications.

9.2.4 App Management

Mobile devices now have many strategies for managing large collections of applications, and for obtaining new applications (e.g. through a store). However, these issues are largely unexplored in the space of on-world applications, and so exploring ways of managing existing applications – switching between applications, summoning and launching new applications, and closing down unused applications. While Desktopography touched on some of these behaviors briefly, there is much more work to be done in this space. For example, one of the major challenges of an on-world interface is in moving your digital context from one space to another – for example, when moving between the study and the kitchen, the system should select a set of applications that will be immediately displayed in the new space. This needs to consider both context and need – perhaps the user wants to keep the document they were reading, and add a recipe application, but does not want to keep their appointment calendar.

As for obtaining new applications, I suggest asking what an “app store” for on-world applications might look like. Applications might need to declare what kinds of physical requirements they need – space required, whether they cooperate with specific objects in the environment – and also consider the privacy implications of allowing applications access to the camera sensors in the on-world system. Users will want to run different kinds of applications on their environment, and so it is interesting to consider how to convey the application requirements of a sophisticated on-world application to the end user.

9.2.5 Building the InfoBulb

Lightbulbs in existing environments already provide illumination across surfaces and spaces, and so these venerable fixtures have been long considered promising candidates for introducing ubiquitous interactivity. Underkoffler *et al.* [Underkoffler 1999] articulated an early vision of the lightbulb as a computational device, but without a practical bulb-sized implementation. In Desktopography, I built a prototype depth camera/projector pair which fits into a standard lamp shade, but lacks on-board computational capabilities. With LumiWatch, I demonstrated that both projectors and depth sensors could be made small enough to fit within the narrow confines of a smartwatch, paving the way for a real, compact “information lightbulb”, or “InfoBulb”. In future work, this bulb can be used as a platform to explore on-world interaction on a wider scale.

Figure 9.1. An early prototype of the InfoBulb concept.
At left: standard light bulb socket for power. At right, picoprojector and small depth camera, connected to offscreen laptop. Future InfoBulb devices will incorporate a computer directly in the design, requiring no tethers to external components or power.

The InfoBulb would serve as a drop-in replacement for existing lightbulbs, comprising a power supply, computer, camera package, and projector. Once installed into a light fixture, the InfoBulb will project out a computer interface, rather than a single colour of light, transforming the underlying surface into an expansive multi-touch surface. Crucially, these leverage existing lighting infrastructures to bring computation to the environment, rather than requiring the construction of new infrastructure (wiring, new surfaces, hardware installation etc.).

Based on our results from LumiWatch, it should be possible to build the InfoBulb from the commodity parts that are now available – small form factor computer, picoprojector and a low-power depth camera. Using commodity hardware will allow us to build multiple identical devices easily and reliably, expanding the scope of potential explorations.

9.3 Conclusion

Computing technology is now ubiquitous, but our interactions with computers are still confined to small screens and input devices. The vast power of computing technology is held fully separate from the physical world, and this has arguably limited the opportunities for computing to synergize more fully with our physical environments. Although the current computing landscape fulfills the literal meaning of the phrase “ubiquitious computing”, the true vision of ubiquity maintains that computing will fade into the background over time, becoming simply embedded in our environments without calling attention to itself. Instead, we see the opposite trend today. Our interactions with computers are largely binary – you are either working with the computer, or with your physical environment. The rise of the smartphone has pushed this trend even further, demanding your full user attention in order to make use of most functions.

This thesis offers one potential way to allow computing to escape the boundaries of the glass prison and enable interactions directly on the world around you. By addressing the problem of on-world computing from technical, interactive and practical angles, it demonstrates the immediate feasibility of on-world interactions and the potential that it holds. Crucially, we explore this form of technology not as a way to supplant the physical world, but as a way to augment it – to bring digital capabilities to physical spaces. The new on-world computing paradigm presages a potential rethinking of the relationship between physical environments and the users who use them, pointing towards a future in which we no longer think of computers as discrete units, but as simply one of the many elements that make up our environments.

CHAPTER 10. REFERENCES

[Adcock 2004] Adcock, M., Hutchins, M. and Gunn, C. Haptic Collaboration with Augmented Reality. *ACM SIGGRAPH 2004 Posters (SIGGRAPH '04)*, pp. 41, 2004.

[Agarawala 2006] Agarawala, A. and Balakrishnan, R. Keepin' It Real: Pushing the Desktop Metaphor with Physics, Piles and the Pen. In *Proc. CHI '06*, 1283-1292.

[Agrawala 2001] Agrawala, M. and Stolte, C. Rendering effective route maps: improving usability through generalization. In *Proc. SIGGRAPH '01*, 241-249.

[Akaoka 2010] Akaoka, E., Ginn, T., and Vertegaal, R. DisplayObjects: prototyping functional physical interfaces on 3d styrofoam, paper or cardboard models. In *Proc. TEI '10*. 49-56.

[Arai 1995] Arai, T., Machii, K, Kuzunuki, S. and Shojima, H. InteractiveDESK: a computer-augmented desk which responds to operations on real objects. In *CHI Companion '95*, 141-142.

[Araujo 2016] B. Araujo, R. Jota, V. Perumal, J. X. Yao, K. Singh and D. Wigdor. Snake Charmer: Physically Enabling Virtual Objects. In *Proc. TEI '16*, 218-226.

[Arduino] Arduino. http://www.arduino.cc

[Ashbrook 2008a] Ashbrook, D., Clawson, J. R., Lyons, K., Starner, T. E and Patel, N. Quickdraw: the impact of mobility and on-body placement on device access time. In *Proc. CHI '08.* 219-222.

[Ashbrook 2008b] Ashbrook, D., Lyons, K., and Starner, T. An investigation into round touchscreen wristwatch interaction. In *Proc. MobileHCI '08*. 311-314.

[Augsten 2010] Augsten, T., Kaefer, K., Meusel, R., Fetzer, C., Kanitz, D., Stoff, T., Becker, T., Holz, C., and Baudisch, P. Multitoe: high-precision interaction with back-projected floors based on high-resolution multi-touch input. In *Proc. UIST '10*. 209-218.

[Avrahami 2002] Avrahami, D. and Hudson, S.E., Forming interactivity: a tool for rapid prototyping of physical interactive products. In *Proc. DIS '02*, 141-146.

[Azmandian 2016] Azmandian, M., Hancock, M., Benko, H., Ofek, E. and Wilson, A. D.. Haptic Retargeting: Dynamic Repurposing of Passive Haptics for Enhanced Virtual Reality Experiences. In *Proc. CHI '16*, 1968-1979.

[Bâce 2016] Bâce, M., Leppänen, T., de Gomez, D. G. and Gomez, A. R.. ubiGaze: ubiquitous augmented reality messaging using gaze gestures. In *Proc. SIGGRAPH ASIA '16*, Article 11, 5 pages.

[Bancroft 1985] Bancroft, S. An algebraic solution of the GPS equations. *IEEE Trans. Aerospace and Electronic Systems*, vol. AES-21, pp. 56-59, Jan. 1985.

[Barnett 2009] Barnett, A. 2009. The dancing body as a screen: Synchronizing projected motion graphics onto the human form in contemporary dance. *Comput. Entertain.* 7(1), Article 5. 32 pages.

[Baudisch 2004] Baudisch, P., Cutrell, E., Hinckley, K. and Gruen, R., Mouse Ether: Accelerating the Acquisition of Targets Across Multi-Monitor Displays. In *Proc. CHI 2004*, 1379-1382.

[Bazo 2014] Bazo, A. and Echtler, F. Phone proxies: effortless content sharing between smartphones and interactive surfaces. In *Proc. EICS '14*. 229-234.

[Benko 2005a] Benko, H. & Feiner, S., Multi-Monitor Mouse. In *Proc. CHI 2005*, 1208-1211.

[Benko 2005b] Benko, H., Ishak, E. W. and Feiner, S. Cross-Dimensional Gestural Interaction Techniques for Hybrid Immersive Environments. In *Proc. IEEE VR '05*, 209–116.

[Benko 2007] Benko, H. & Feiner, S., Pointer Warping in Heterogeneous Multi-Monitor Environments. In *Proc. Graphics Interface,* 2007, 111-117.

[Benko 2012] Benko, H., Jota, R. and Wilson, A. Miragetable: freehand interaction on a projected augmented reality tabletop. In *Proc. CHI '12*. 199–208.

[Benko 2016] Benko, H., Holz, C., Sinclair, M. and Ofek, E. NormalTouch and TextureTouch: High-fidelity 3D Haptic Shape Rendering on Handheld Virtual Reality Controllers. In *Proc. UIST '16*, 717-728.

[Bi 2011] Bi, X., Grossman, T., Matejka, J., Fitzmaurice, G. Magic desk: bringing multi-touch surfaces into desktop work. In *Proc. CHI '11*.

[Billinghurst 2004] Billinghurst, M., Kato, H. and Poupyrev, I. Collaboration with tangible augmented reality interfaces. In *Proc. HCI Int.*, 234-241, 2004.

[Bimber 2005] Bimber, O. and Raskar, R. 2005. *Spatial Augmented Reality: Merging Real and Virtual Worlds*. A. K. Peters, Ltd., Natick, MA, USA.

[Bodart 1994] Bodart, F., Hennebert, A., Leheureux, J. and Vanderdonckt, J. Towards a dynamic strategy for computer-aided visual placement. In *Proc. AVI '94*, 78-87.

[Bolt 1980] Bolt, R. A. Put-that-there: Voice and gesture at the graphics interface. In *Proc. SIGGRAPH '80*, 262-270.

[Bondarenko 2005] Bondarenko, O. and Janssen, R. Documents at Hand: Learning from Paper to Improve Digital Technologies. In *Proc. CHI '05*, 121-130.

[Brooks 1997] Brooks, R. A. The Intelligent Room Project. In *Proc. International Conference on Cognitive Technology '97*, 271.

[Buckley 2010] Buckley, E. 2010. Eye-safety analysis of current laser-based scanned-beam projection systems. Journal of the Society for Information Display, 18: 944–951.

[Burdea 1992] Burdea, G., Zhuang, J., Roskos, E., Silver, D. and Langrana, N. A portable dextrous master with force feedback. *Presence: Teleoper. Virtual Environ* 1(1), January 1992, 18-28.

[Butler 2008] Butler, A., Izadi, S., and Hodges, S. SideSight: multi-"touch" interaction around small devices. In *Proc. UIST '08*. 201-204.

[Cabral 2005] Cabral, M. C., Morimoto, C. H. and Zuffo, M. K. On the usability of gesture interfaces in virtual reality environments. In *Proc. CLIHC '05*, 100-108.

[Caffery 1998] Caffery, J. and Stuber, G. L. Overview of radiolocation in CDMA cellular systems. *IEEE Commun. Mag.*, 36(4), pp. 38-45, Apr. 1998.

[Canny 1986] Canny, J. A Computational Approach to Edge Detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 8(6), 1986, 679-698.

[Cao 2008] Cao, X., Wilson, A.D., Balakrishnan, R., Hinckley, K., Hudson, S.E. ShapeTouch: Leveraging Contact Shape on Interactive Surfaces. In *Proc. Tabletop '08*, 129-136.

[Carter 1981] Carter, G. C. Time delay estimation for passive sonar signal processing. *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. ASSP-29, pp. 463-470, June 1981.

[Carter 2013] Carter, T., Seah, S. A., Long, B., Drinkwater, B. and Subramanian, S. UltraHaptics: multi-point mid-air haptic feedback for touch surfaces. In *Proc. UIST '13*, 505-514.

[Chang 2005] Chang, J. S., Kim, E. Y., KC Jung and Kim, H. J. Real time hand tracking based on active contour model. In *Proc. ICCSA '05*, 999-1006.

[Chatterjee 2015] Chatterjee, I., Xiao, R. and Harrison, C. Gaze+Gesture: Expressive, Precise and Targeted Free-Space Interactions. In *Proc. ICMI '15*, 131-138.

[Chen 2014] Chen, X.A., Schwarz, J., Harrison, C., Mankoff, J. and Hudson, S.E. Air+touch: interweaving touch & in-air gestures. In *Proc. UIST '14*, 519-525.

[Chen 2015] Chen, H., Lee, A. S., Swift, M. and Tang, J. C. 3D Collaboration Method over HoloLens™ and Skype™ End Points. In *Proc. ImmersiveME '15*, 27-30.

[Cohn 2011] Cohn, G., Morris, D., Patel, S.N., and Tan, D.S. Your noise is my command: sensing gestures using the body as an antenna. In *Proc. CHI '11*. 791-800.

[Cong 1996] Cong, J., He, L., Koh, C. and Madden, P.H. Performance optimization of VLSI interconnect layout. *Integr. VLSI J.* 21(1-2), 1996, pp. 1-94.

[Crow 1984] Crow, F. Summed-area tables for texture mapping. In *Proc. SIGGRAPH '84*. 207–212.

[Dezfuli 2012] Dezfuli, N., Khalilbeigi , M., Huber, J., Müller, F., and Mühlhäuser, M. 2012. PalmRC: imaginary palm-based remote control for eyes-free television interaction. In *Proc. EuroITV '12*. 27-34.

[Di Battista 1998] Di Battista, G., Eades, P., Tamassia, R. and Tollis, I.G. 1998. Graph Drawing: Algorithms for the Visualization of Graphs (1st ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Donnelly 2009] Donnelly, L., Patten, D., White, P. and Finn, G. 2009. Virtual human dissector as a learning tool for studying cross-sectional anatomy. *Med Teach.* Jun 2009, 31(6):553-555.

[Dorfmuller-Ulhaas 2001] Dorfmuller-Ulhaas, K. and Schmalstieg, D. Finger tracking for interaction in augmented environments. In *Proc. IEEE and ACM Int. Symp. Augmented Reality 2001*, 55-64.

[Eastman Kodak 2003] Eastman Kodak Company. Kodak's Ergonomic Design for People at Work, 2nd Edition. 2003, pp. 48-49.

[Fails 2002] Fails, J.A. and Olsen, D. Light Widgets: Interacting in Everyday Spaces. In *Proc. IUI '02*, 63-69.

[Fischler 1981] Fischler, M. A. and Bolles, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* 24(6), June 1981, 381-395.

[Fitzpatrick 1975] Fitzpatrick, T. B. 1975. "Soleil et peau" [Sun and skin]. *Journal de Médecine Esthétique* (2): 33–34.

[Fogarty 2003] Fogarty, J. and Hudson, S.E. GADGET: A Toolkit for Optimization-Based Approaches to Interface and Display Generation. In *Proc. UIST '03*, 125-134.

[Frederiksen 2011] Frederiksen, A., Fieß, R., Stork, W., Bogatscher, S. and Heußner, N. 2011. Eye safety for scanning laser projection systems. *Biomed Tech* (Berl). 2012 May 31;57(3):175-184.

[Gajos 2008] Gajos, K.Z., Weld, D.S. and Wobbrock, J.O. Decision-theoretic user interface generation. In *Proc. AAAI '08*, 1532-1536.

[Gajos 2010] Gajos, K.Z., Weld, D.S., and Wobbrock, J.O. Automatically generating personalized user interfaces with Supple. *Artificial Intelligence*, vol. 174, 12-13 (August 2010), 910-950.

[Gartner 2015] Gartner, Inc. 2015. “Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016.” November 10, 2015. http://www.gartner.com/newsroom/id/3165317

[Gavaghan 2011] Gavaghan, K. A., Anderegg, S., Peterhans, M., Oliveira-Santos, T., and Weber S. Augmented reality image overlay projection for image guided open liver ablation of metastatic liver cancer. In *Proc AECAI ’11*, 36-46.

[Gebhardt 2014] Gebhardt, C., Rädle, R. and Reiterer, H. Integrative workplace: studying the effect of digital desks on users' working practices. In *CHI EA ‘14*, 2155-2160.

[Goldberg 1993] Goldberg, D. and Richardson, C. Touch-typing with a stylus. In *Proc. CHI ’*93, 80-87.

[Good 1984] Good, M.D., Whiteside, J.A., Wixon, D.R., and Jones, S.J. Building a user-derived interface. *Commun. ACM* 27, 10 (October 1984), 1032-1043.

[Greenberg 2001] Greenberg, S. and Fitchett, C. Phidgets: easy development of physical interfaces through physical widgets. In *Proc. UIST ‘01*. 209-218.

[Gugenheimer 2016] Gugenheimer, J., Dobbelstein, D., Winkler, C., Haas, G. and Rukzio, E. FaceTouch: Enabling Touch Interaction in Display Fixed UIs for Mobile Virtual Reality. In *Proc. UIST ’16*, 49-60.

[Gustafson 2008] Gustafson, S., Baudisch, P., Gutwin, C, and Irani, P., Wedge: Clutter-Free Visualization of Off-Screen Locations, In *Proc. CHI 2008*, 787-796.

[Gustafson 2011] Gustafson, S., Holz, C. and Baudisch, P. Imaginary phone: learning imaginary interfaces by transferring spatial memory from a familiar device. In *Proc. UIST ’11*, 283-292.

[Hachet 2011] Hachet, M., Bossavit, B., Cohé, A. and J-B de la Rivière. Toucheo: multitouch and stereo combined in a seamless workspace. In *Proc. UIST ’11*, 587-592.

[Haley 1988] Haley, J. Anthropometry and mass distribution for human analogues. Volume 1, 1988. Aerosp. Med. Res. Lab Wright-Patterson, Ohio.

[Han 2005] Han, J. Y. Low-cost multi-touch sensing through frustrated total internal reflection. In *Proc. UIST ‘05*. 115-118.

[Hardy 2012] Hardy, J. Experiences: a year in the life of an interactive desk. In *Proc. DIS ‘12*, 679-688.

[Harrison 2008] Harrison, C. and Hudson, S.E. Scratch input: creating large, inexpensive, unpowered and mobile finger input surfaces. In *Proc. UIST ‘08*. 205-208.

[Harrison 2009] Harrison, C. and Hudson, S. Abracadabra: wireless, high-precision, and unpowered finger input for very small mobile devices. In *Proc. UIST '09*. 121-124.

[Harrison 2010a] Harrison, C., Wiese, J. and Dey, A. K. "Achieving Ubiquity: The New Third Wave." *IEEE Multimedia*, 17, 3 (July-September 2010), 8-12.

[Harrison 2010b] Harrison, C. Appropriated Interaction Surfaces. *IEEE Computer Magazine*, June 2010, 43(6). 86-89

[Harrison 2010c] Harrison, C., Tan, D. and Morris, D. Skinput: appropriating the body as an input surface. In *Proc. CHI '10*, 453-462.

[Harrison 2011] Harrison, C., Benko, H. and Wilson, A.D. OmniTouch: wearable multitouch interaction everywhere. In *Proc. UIST '11*. 441-450.

[Harrison 2012] Harrison, C., Ramamurthy, S. and Hudson, S. E. On-body interaction: armed and dangerous. In *Proc. TEI '12*, 69-76.

[Hartmann 2006] Hartmann, B., Klemmer, S.R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A. and Gee, J., Reflective physical prototyping through integrated design, test, and analysis. In *Proc. UIST '06*, 299-308.

[Hettiarachchi 2016] Hettiarachchi, A. and Wigdor, D. Annexing Reality: Enabling Opportunistic Use of Everyday Objects as Tangible Proxies in Augmented Reality. In *Proc. CHI '16*, 1957-1967.

[Hincapié-Ramos 2014] Hincapié-Ramos, J. D., Guo, X., Moghadasian, P. and Irani, P. Consumed endurance: a metric to quantify arm fatigue of mid-air interactions. In *Proc. CHI '14*, 1063-1072.

[Hinckley 2004] Hinckley, K., Ramos, G., Guimbretiere, F., Baudisch, P. and Smith, M. Stitching: pen gestures that span multiple displays. In *Proc. AVI '04*. 23-31.

[Hinckley 2016] Hinckley, K., Heo, S., Pahud, M., Holz, C., Benko, H., Sellen, A., Banks, R., O'Hara, K., Smyth, G. and Buxton, W. Pre-Touch Sensing for Mobile Interaction. In *Proc. CHI '16*, 2869-2881.

[Hodes 1997] Hodes, T.D., Katz, R.H., Servan-Schreiber, E. and Rowe, L. Composable ad-hoc mobile services for universal interaction. In *Proc. MobiCom '97*, 1-12.

[Holz 1997] Holz, C. and Baudisch, P. The generalized perceived input point model and how to double touch accuracy by extracting fingerprints. In *Proc. CHI '10*, 581-590.

[HTC] HTC Corporation. HTC Vive Controller. https://www.vive.com/us/accessory/controller/

[Hudson 2006] Hudson, S.E. and Mankoff, J. Rapid construction of functioning physical interfaces from cardboard, thumbtacks, tin foil and masking tape. In *Proc. UIST '06*. 289-298.

[Hudson 2010] Hudson, S.E., Harrison, C., Harrison, B.L. and LaMarca, A. Whack gestures: inexact and inattentive interaction with mobile devices. In *Proc. TEI '10*, 109-112.

[Hürst 2013] Hürst, W. and van Wezel, C. Gesture-based Interaction via Finger Tracking for Mobile Augmented Reality. *Multimedia Tools and Applications* 62(1) January 2013, 233-258.

[Intel 2012] Intel Corporation. Object Aware Situated Interactive System (OASIS). Retrieved April 7, 2012: http://techresearch.intel.com/ProjectDetails.aspx?Id=84

[Ishii 1990] Ishii, H. TeamWorkStation: Towards a Seamless Shared Workspace. In *Proc. CSCW '90*, 13-26.

[Ishii 1997] Ishii, H. and Ullmer, B. Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proc. CHI '97*, 234-241.

[Ishii 1999] Ishii, H., Wisneski, C., Orbanes, J., Chun, C., and Paradiso, J. PingPongPlus: design of an athletic-tangible interface for computer-supported cooperative play. In *Proc. CHI '99*. 394-401.

[Izadi 2011] Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A. and Fitzgibbon, A. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *Proc. UIST '11*. 559-568.

[Jones 2010] Jones, B., Sodhi, R., Campbell, R., Garnett, G., and Bailey, B. Build your world and play in it: Interacting with surface particles on complex objects. In *Proc. ISMAR '10*. 165 - 174.

[Jota 2013] Jota, R., Ng, A., Dietz, P. and Wigdor, D. How fast is fast enough?: a study of the effects of latency in direct-touch pointing tasks. In *Proc. CHI '13*, 2291-2300.

[Junuzovic 2012] Junuzovic, S., Quinn, K.I., Blank, T. and Gupta, A. IllumiShare: Sharing Any Surface. In *Proc. CHI '12*, 1919-1928.

[Kane 2009] Kane, S.K., Avrahami, D., Wobbrock, J.O., Harrison, B., Rea, A.D., Philipose, M. and LaMarca, A. Bonfire: a nomadic system for hybrid laptop-tabletop interaction. In *Proc. UIST '09*, 129-138.

[Kao 2016] Kao, H-L., Holz, C., Roseway, A., Calvo, A. and Schmandt, C. DuoSkin: rapidly prototyping on-skin user interfaces using skin-friendly materials. In *Proc. ISWC '16*, 16-23.

[Khalilbeigi 2013] Khalilbeigi, M., Steimle, J., Riemann, J., Dezfuli, N., Mühlhäuser, M. and Hollan, J. D. ObjecTop: Occlusion Awareness of Physical Objects on Interactive Tabletops. In *Proc. ITS '13*, 255-264.

[Khoshelham 2012] Khoshelham, K. and Elberink, S.O. Accuracy and resolution of Kinect depth data for indoor mapping applications. *Sensors*, 12(2), 2012, 437-454.

[Kiang 1998] Kiang, M.-H., Solgaard, O., Lau K.Y. and Muller, R.S. Electrostatic combdrive-actuated micromirrors for laser-beam scanning and positioning. *J Microelectromech Syst*, 7(1), Mar 1998, 27-37.

[Kim 2014] Kim, J., Seo, J. and Han, T-D. AR Lamp: interactions on projection-based augmented reality for interactive learning. In *Proc. IUI '14*, 353-358.

[Knowlton 1977] Knowlton, K. C. Computer Displays Optically Superimposed on Input Devices. *Bell Systems Technical Journal*, 53(3), 1977, 367-383.

[Koike 2001] Koike, H., Sato, Y. and Kobayashi, Y. Integrating paper and digital information on EnhancedDesk: a method for realtime finger tracking on an augmented desk system. *ACM Trans. on Computer-Human Interaction*, 8 (4), 307-322.

[Kramer 2011] Kramer, R. K., Majidi, C. and Wood, R. J. Wearable tactile keypad with stretchable artificial skin. In *Proc. ICRA '11*, 1103-1107.

[Kratz 2009] Kratz, S. and Rohs, M. HoverFlow: expanding the design space of around-device interaction. In *Proc. MobileHCI '09*. Article 4, 8 pages.

[Kreylos 2009] Kreylos, O. Vrui VR Toolkit. http://idav.ucdavis.edu/~okreylos/ResDev/Vrui.

[Krueger 1985] Krueger, M. W., Gionfriddo, T. and Hinrichsen, K. VIDEOPLACE — an artificial reality. In *Proc. CHI '85*. 35-40.

[Landay 2001] Landay, J.A. and Myers, B.A. Sketching Interfaces: Toward More Human Interface Design. *Computer* 34, 3 (March 2001), 56-64.

[Laput 2014] Laput, G., Xiao, R., Chen, X. A., Hudson, S. E. and Harrison, C. Skin buttons: cheap, small, low-powered and clickable fixed-icon laser projectors. In *Proc. UIST '14*, 389-394.

[Laput 2015] Laput, G., Yang, C., Xiao, R., Sample, A. and Harrison, C. 2015. EM-Sense: Touch Recognition of Uninstrumented, Electrical and Electromechanical Objects. In *Proc. UIST '15*, 157-166.

[Laput 2016] Laput, G., Xiao, R. and Harrison, C. ViBand: High-Fidelity Bio-Acoustic Sensing Using Commodity Smartwatch Accelerometers. In *Proc. UIST '16*, 321-333.

[Leap Motion] Leap Motion, Inc. Leap Motion Mobile VR Platform. https://www.leapmotion.com/product/vr

[Lee 1985] Lee, S. K., Buxton, W. and Smith, K. C. A multi-touch three dimensional touch-sensitive tablet. In *Proc. CHI '85*. 21-25.

[Lee 2004] Lee, J.C., Avrahami, D., Hudson, S.E., Forlizzi, J., Dietz, P., and Leigh, D. The calder toolkit: wired and wireless components for rapidly prototyping interactive devices. In *Proc. DIS '04*, 167-175.

[Lee 2005] Lee, J., Forlizzi, J. and Hudson, S.E. Studying the effectiveness of MOVE: a contextually optimized in-vehicle navigation system. In *Proc. CHI '05*, 571-580.

[Lee 2007] Lee, T. and Hollerer, T. Handy AR: Markerless Inspection of Augmented Reality Objects Using Fingertip Tracking. In *Proc. ISWC '07*,1-8, 2007.

[Leo 2002] Leo, C. K. Contact and Free-Gesture Tracking for Large Interactive Surfaces. MEng Thesis, MIT Dept. of EECS and MIT Media Lab, May 2002.

[Letessier 2004] Letessier, J. and F. Bérard. Visual tracking of bare fingers for interactive surfaces. In *Proc. UIST '04*, 119-122.

[Liang 2011] Liang, R-H., Lin, S-Y., Su, C-H., Cheng, K-Y., Chen, B-Y. and Yang, D-N. SonarWatch: appropriating the forearm as a slider bar. In *Proc. SIGGRAPH Asia '11 Emerging Technologies*, Article 5, 1 page.

[Lim 2015] Lim, S-C., Shin, J., Kim, S-C. and Park, J. Expansion of Smartwatch Touch Interface from Touchscreen to Around Device Interface Using Infrared Line Image Sensors. *Sensors* 15(7), 2015, 16642-16653.

[Lin 2011] Lin, S-Y., Su, C-H., Cheng, K-Y., Liang, R-H., Kuo, T-H. and Chen, B-Y. PUB - point upon body: exploring eyes-free interaction and methods on an arm. In *Proc. UIST '11*, 481-488.

[Lindeman 2004] Lindeman, R. W., Page, R., Yanagida, Y. and Sibert, J. L. Towards full-body haptic feedback: the design and deployment of a spatialized vibrotactile feedback system. In *Proc. VRST '04*, 146-149.

[Linder 2010] Linder, N. and Maes, P. LuminAR: portable robotic augmented reality interface design and prototype. In *Adj. Proc. UIST '10*, 395-396.

[Lyons 2004] Lyons, K., Starner, T., Plaisted, D., Fusia, J., Lyons, A., Drew, A. and Looney, E. W. Twiddler typing: one-handed chording text entry for mobile phones. In *Proc. CHI '04*, 671-678.

[MacKenzie 1997] MacKenzie, I. S. and Zhang, S. X. The immediate usability of graffiti. In *Proc. GI '97*, 129-137.

[Maeda 1998] Maeda, J., Iizawa, T., Ishizaka, T., Ishikawa, C. and Suzuki, Y. Segmentation of Natural Images Using Anisotropic Diffusion and Linking of Boundary Edges. Pattern Recognition, 31(12), 1998.

[Malone 1983] Malone, T. W. How do people organize their desks?: Implications for the design of office information systems. *ACM Trans. Inf. Syst.* 1(1), 1983, 99-112.

[Mann 1997] Mann, S. "Smart clothing": wearable multi-media computing and "personal imaging" to restore the technological balance between people and their environments. In *Proc. MULTIMEDIA '96*, 163-174.

[Massie 1994] Massie, T. H. and Salisbury, J. K. The PHANToM Haptic Interface: A Device for Probing Virtual Objects. *ASME Winter Annual Meeting, DSC* 55(1), 295−300, 1994.

[Matsushita 1997] Matsushita, N. and Rekimoto, J. HoloWall: designing a finger, hand, body, and object sensitive wall. In *Proc. UIST '97*. 209-210.

[Maynes-Aminzade 2007] Maynes-Aminzade, D., Winograd, T., and Igarashi, T. Eyepatch: prototyping camera-based interaction through examples. In *Proc. UIST '07*. 33-42.

[McFarlane 2009] McFarlane, D.C. and Wilder, S.M. Interactive dirt: increasing mobile work performance with a wearable projector-camera system. In *Proc. UbiComp '09*, 205-214.

[Medeiros 2013] Medeiros, D., Teixeira, L., Carvalho, F., Santos, I. and Raposo, A. A tablet-based 3D interaction tool for virtual engineering environments. In *Proc. VRCAI '13*, 211-218.

[Microsoft Kinect] Microsoft Corporation. Kinect hardware. https://developer.microsoft.com/en-us/windows/kinect/hardware

[Microsoft Hololens] Microsoft Corporation. Microsoft HoloLens. https://www.microsoft.com/microsoft-hololens/

[Mistry 2009] Mistry, P., Maes, P. and Chang, L. WUW - wear Ur world: a wearable gestural interface. In *Proc. CHI EA '09*, 4111-4116.

[Mujibiya 2013] Mujibiya, A. Cao, X., Tan, D.S., Morris, D., Patel, S.N. and Rekimoto, J. The sound of touch: on-body touch and gesture sensing based on transdermal ultrasound propagation. In *Proc. ITS '13*, 189-198.

[Mulder 2003] Mulder, J., Jansen, J. and Rhijn, V. An affordable optical head tracking system for desktop VR/AR systems. In *Proc. EGVE '03*, 215-223.

[Mulloni 2011] Mulloni, A, Seichter, H. and Schmalstieg, D. Handheld augmented reality indoor navigation with activity-based instructions. In *Proc. MobileHCI '11*. 211-220.

[Nacenta 2008] Nacenta, M., Mandryk, R., and Gutwin, C, Targeting across display-less space. In *Proc. CHI 2008*, 777-786.

[Nacenta 2009] Nacenta, M., Gutwin, C., Aliakseyeu, D., and Subramanian, S., There and Back again: Cross-Display Object Movement in Multi-Display Environments, *JHCI*, 24, 1, 2009, 170-229.

[NASA 1995] NASA. Anthropometry and Biomechanics. NASA-STD-3000: Man-Systems Integration Standards, Volume 1, Section 3. Revision B, July 1995.

[Newman 1992] Newman, W. and Wellner, P. A desk supporting computer-based interaction with paper documents. In *Proc. CHI '92*, 587-592.

[Ng 2012] Ng, A., Lepinski, J., Wigdor, D., Sanders, S. and Diet, P. Designing for low-latency direct-touch input. In *Proc. UIST '12*, 453-464.

[Nielsen 2004] Nielsen, M., Störring, M., Moeslund, T.B. and Granum, E. (2004) A procedure for developing intuitive and ergonomic gesture interfaces for HCI. *Int'l Gesture Workshop 2003*, LNCS vol. 2915. Heidelberg: SpringerVerlag, 409-420.

[NTT 2010] NTT IT Corp. 2010. TenoriPop: http://tenoripop.com.

[Ogata 2013] Ogata, M., Sugiura, Y., Makino, Y., Inami, M. and Imai, M. SenSkin: adapting skin as a soft interface. In *Proc. UIST '13*, 539-544.

[Ogata 2015] Ogata, M., Totsuka, R. and Imai, M. SkinWatch: adapting skin as a gesture surface. In *SIGGRAPH Asia 2015 Emerging Technologies*, Article 22, 2 pages.

[Olsen 2008] Olsen, D. 2008. Interactive viscosity. In *Proceedings of the 21st annual ACM symposium on User interface software and technology* (UIST '08). ACM, New York, NY, USA, 1-2. http://dx.doi.org/10.1145/1449715.1449717

[Paradiso 2000] Paradiso, J., Hsiao, K., Strickon, J., Lifton, J. and Adler, A. Sensor Systems for Interactive Surfaces. IBM Systems Journal, Volume 39, Nos. 3 & 4, October 2000, pp. 892-914.

[Paradiso 2002] Paradiso, J., Leo, C., Checka, N. and Hsiao, K. Passive acoustic sensing for tracking knocks atop large interactive displays. In *Proc. IEEE Sensors '02*. 521-527.

[Paradiso 2005] Paradiso, J. and Leo, C. Tracking and Characterizing Knocks Atop Large Interactive Displays. *Sensor Review*, vol. 25, no. 2, pp. 134-143, 2005.

[Park 2008] Park, H. M., Lee, S. H. and Choi, J. S. Wearable augmented reality system using gaze interaction. In *Proc. ISMAR '08*, pp. 175-176.

[Patten 2007] Patten, D. 2007. What lies beneath: the use of three-dimensional projection in living anatomy teaching. *The Clinical Teacher*, 4: 10–14.

[Pinhanez 2001] Pinhanez, C.S. The Everywhere Displays Projector: A Device to Create Ubiquitous Graphical Interfaces. In *Proc. UbiComp '01*, 315-331.

[Post 1997] Post, E. R. and Orth, M. Smart Fabric, or "Wearable Clothing". In *Proc. ISWC '97*, 167-168.

[Processing] Processing. http://www.processing.org

[Raskar 1998] Raskar, R., Welch, G., Cutts, M., Lake, A., Stesin, L. and Fuchs, H. The office of the future: a unified approach to image-based modeling and spatially immersive displays. In *Proc. SIGGRAPH '98*, 179-188.

[Raskar 2003] Raskar, R., Baar, J., Beardsley, P., Willwacher, T., Rao, S. and Forlines, C. iLamps: Geometrically Aware and Self-Configuring Projectors. In *Proc. SIGGRAPH '03*, 809-818.

[Rekimoto 1997] Rekimoto, J. Pick-and-drop: a direct manipulation technique for multiple computer environments. In *Proc. UIST '97*. 31-39.

[Robertson 1999] Robertson, C. and Robinson, J. Live paper: video augmentation to simulate interactive paper. In *Proc. MULTIMEDIA '99*, 167-170.

[Saba 2012] Saba, E.N., Larson, E.C. and Patel, S.N. Dante vision: In-air and touch gesture sensing for natural surface interaction with combined depth and thermal cameras. In *Proc. IEEE ESPA '12*. 167-170.

[Sakata 2009] Sakata N., Konishi, T. and Nishida, S. Mobile Interfaces Using Body Worn Projector and Camera. In *Proc. VMR '09*, 106-113.

[Saponas 2009] Saponas T.S., Tan, D.S., Morris, D., Balakrishnan, R., Turner, J. and Landay, J.A. Enabling always-available input with muscle-computer interfaces. In *Proc. UIST '09*, 167-176.

[Saponas 2011] Saponas T.S., Harrison, C. and Benko, H. PocketTouch: through-fabric capacitive touch input. In *Proc. UIST '11*, 303-308.

[Sato 2012] Sato, M., Poupyrev, I. and Harrison, C. Touché: enhancing touch interaction on humans, screens, liquids, and everyday objects. In *Proc. CHI '12*, 483-492.

[Schmidt 2010] Schmidt, D., Chong, M. K. and Gellersen, H. HandsDown: hand-contour-based user identification for interactive surfaces. In *Proc. NordiCHI '10*. 432-441.

[Schmidt 2012] Schmidt, D., Molyneaux, D. and Cao, X. PICOntrol: using a handheld projector for direct control of physical devices through visible light. In *Proc. UIST '12*, 379-388.

[Sears 1993] Sears, A. Layout Appropriateness: A Metric for Evaluating User Interface Widget Layout. *IEEE Trans. Softw. Eng.* 19(7), 1993, pp. 707-719.

[Seewoonauth 2009] Seewoonauth, K., Rukzio, E., Hardy, R. and Holleis, P. Touch & connect and touch & select: interacting with a computer by touching it with a mobile phone. In *Proc. MobileHCI '09*. Article 36 , 9 pages.

[Sellen 2003] Sellen, A. and Harper, R. The myth of the paperless office. The MIT Press, Cambridge, London, 2003.

[Sodhi 2012] Sodhi, R., Benko, H. and Wilson, A. LightGuide: projected visualizations for hand movement guidance. In *Proc. CHI '12*, 179-188.

[Sodhi 2013] Sodhi, R., Poupyrev, I., Glisson, M. and Israr, A. AIREAL: interactive tactile experiences in free air. *ACM Trans. Graph.* 32(4), Article 134 (July 2013), 10 pages.

[Sridhar 2017] Sridhar S., Markussen, A., Oulasvirta, A., Thobalt, C. and Boring, C. WatchSense: On- and Above-Skin Input Sensing through a Wearable Depth Sensor. In *Proc. CHI '17*, 3891-3902.

[Steimle 2010] Steimle, J., Khalilbeigi, M., Mühlhäuser, M. and Hollan, J.D. Physical and digital media usage patterns on interactive tabletop surfaces. In *Proc. ITS '10*, 167-176.

[Steimle 2013] Steimle, J., Jordt, A. and Maes, P. Flexpad: highly flexible bending interactions for projected handheld displays. In *Proc. CHI '13*. 237-246.

[Sugita 2008] Sugita, N., Iwai, D. and Sato K. Touch Sensing by Image Analysis of Fingernail. In *Proc. SICE Annual Conference '08*. 1520-1525.

[Sugrue 2007] Sugrue, C. 2007. "Delicate Boundaries"

[Tan 2010] Tan, D., Morris, D. and Saponas, T. S. Interfaces on the go. *XRDS* 16, 4 (June 2010), 30-34.

[Tang 2008] Tang, A., Greenberg, S., and Fels, S. Exploring video streams using slit-tear visualizations. In *Proc. AVI '08*. 191-198.

[Thomas 2002] Thomas, B. H., Grimmer, K., Zucco, J. and Milanese, S. Where Does the Mouse Go? An Investigation into the Placement of a Body-Attached TouchPad Mouse for Wearable Computers. *Personal Ubiquitous Comput.* 6, 2 (January 2002), 97-112.

[Underkoffler 1998] Underkoffler, J., Ishii, H. Illuminating light: an optical design tool with a luminous-tangible interface. In *Proc. CHI '98*, 542-549.

[Underkoffler 1999] Underkoffler, J., Ullmer, B., Ishii, H. Emancipated pixels: real-world graphics in the luminous room. In *Proc. SIGGRAPH '99*, 385-392.

[Vyas 2012] Vyas, D. and Nijholt, A. Artful surfaces: an ethnographic study exploring the use of space in design studios. *Digital Creativity*, 23(1), 2012, 1-20.

[Wagner 2003] Wagner, D. and Schmalstieg, D. 2003. First Steps Towards Handheld Augmented Reality. In *Proc. ISWC '03*.

[Wagner 2005] Wagner D., Pintaric T., Ledermann F. and Schmalstieg D. Towards Massively Multi-user Augmented Reality on Handheld Devices. In *Pervasive 2005*.

[Walsh 2014] Walsh, J. A., von Itzstein, S. and Thomas, B. H. Ephemeral Interaction using Everyday Objects. In *Proc. AUIC '14*, 29-37.

[Wang 2009] Wang, F. and Ren, X. Empirical evaluation for finger input properties in multi-touch interaction. In *Proc. CHI '09*. 1063-1072.

[Wei 2010] Wei, D., Zhou, S. Z. and Xie, D. MTMR: A conceptual interior design framework integrating Mixed Reality with the Multi-Touch tabletop interface. In *Proc. ISMAR '10*, 279-280.

[Weigel 2015] Weigel, M., Lu, T., Bailly, G., Oulasvirta, A., Majidi, C. and Steimle, J. iSkin: Flexible, Stretchable and Visually Customizable On-Body Touch Sensors for Mobile Computing. In *Proc. CHI '15*, 2991-3000.

[Weiser 1999] Weiser, M. The Computer for the 21st Century. *SIGMOBILE Mob. Comput. Commun*. Rev. 3, 3 (July 1999), 3-11.

[Welch 2000] Welch, G., Fuchs, H., Raskar, R., Towles, H., and Brown, M., Projected Imagery in Your Office in the Future. *IEEE Computer Graphics and Applications*, Jul-Aug 2000, 20, 4, 62-67.

[Wellner 1991] Wellner. P. The DigitalDesk calculator: tangible manipulation on a desk top display. In *Proc. UIST '91*, 27-33.

[Wellner 1993] Wellner, P. Interacting with paper on the DigitalDesk. *Communications of the ACM*, 36 (7), 87-96.

[White 1980] White, R. M. Comparative anthropometry of the hand. No. NATICK/CEMEL-229. Army Natrick Research and Development Labs, MA. Clothing Equipment and Materials Engineering Lab, 1980.

[Wilson 2004] Wilson, A. TouchLight: An Imaging Touch Screen and Display for Gesture-Based Interaction. In *Proc. ICMI '04*. 69-76.

[Wilson 2005] Wilson, A. PlayAnywhere: A Compact Interactive Tabletop Projection-Vision System. In *Proc. UIST '05*. 83-92.

[Wilson 2007] Wilson, A.D., Depth-Sensing Video Cameras for 3D Tangible Tabletop Interaction. In *Proc. Tabletop '07*, 201-204.

[Wilson 2010a] Wilson, A.D. Using a depth camera as a touch sensor. In *Proc. ITS '10*. 69-72.

[Wilson 2010b] Wilson, A.D. and Benko, H. Combining multiple depth cameras and projectors for interactions on, above and between surfaces. In *Proc. UIST '10*. 273-282.

[Wimmer 2010] Wimmer, R., Hennecke, F., Schulz, F., Boring, S., Butz, A. and Hußmann, H. Curve: revisiting the digital desk. In *Proc. NordiCHI '10*, 561-570.

[Wobbrock 2005] Wobbrock, J.O., Aung, H.H., Rothrock, B., and Myers, B.A. 2005. Maximizing the guessability of symbolic input. In *CHI '05 EA*. 1869-1872.

[Wobbrock 2009] Wobbrock, J.O., Morris, M.R., and Wilson, A.D. User-defined gestures for surface computing. In *Proc. CHI '09*. 1083-1092.

[Xiao 2013] Xiao, R., Harrison, C., and Hudson, S.E. WorldKit: Rapid and Easy Creation of Ad-hoc Interactive Applications on Everyday Surfaces. In *Proc. CHI '13*, 879-888.

[Xiao 2014] Xiao, R., Lew, G., Marsanico, J., Hariharan, D., Hudson, S.E. and Harrison, C. Toffee: enabling ad hoc, around-device interaction with acoustic time-of-arrival correlation. In *Proc. MobileHCI '14*. 67-76.

[Xiao 2015] Xiao, R., Schwarz, J. and Harrison, C. Estimating 3D Finger Angle on Commodity Touchscreens. In *Proc. ITS '15*. 47-50.

[Yamamoto 2007] Yamamoto, G. and Sato, K. PALMbit: A PALM Interface with Projector-Camera System. In *Adj. Proc. UbiComp '07*, 276-279.

[Zeidler 2013] Zeidler, C., Lutteroth, C., Sturzlinger, W. and Weber, G. The Auckland Layout Editor: An Improved GUI Layout Specification Process. In *Proc. UIST '13*, 343-352.

[C. Zhang 2016] Zhang, C., Bedri, A., Reyes, G., Bercik, B., Inan, O. T., Starner, T. E. and Abowd, G. D. TapSkin: Recognizing On-Skin Input for Smartwatches. In *Proc. ISS '16*, 13-22.

[Y. Zhang 2016] Zhang, Y., Zhou, J., Laput, G. and Harrison, C. SkinTrack: Using the Body as an Electrical Waveguide for Continuous Finger Tracking on the Skin. In *Proc. CHI '16*, 1491-1503.

[Zhou 2008] Zhou, F., Duh, H. B-L. and Billinghurst, M. Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR. In *Proc. ISMAR '08*, 193-202.

[Zhou 2016] Zhou, J., Zhang, Y., Laput, G. and Harrison, C. AuraSense: Enabling Expressive Around-Smartwatch Interactions with Electric Field Sensing. In *Proc. UIST '16*, 81-86.