Interaction with Embodied Media

by

David Jeffrey Merrill

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

[June]
May 2009

Author .....

Program in Media Arts and Sciences,
School of Architecture and Planning
May 1, 2009

Certified by .....

Pattie Maes
Associate Professor of Media Arts and Sciences
Thesis Supervisor

Accepted by .....

Deb Roy
Chair, Program in Media Arts and Sciences

Interaction with Embodied Media

by

David Jeffrey Merrill

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
on May 1, 2009, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The graphical user interface has become the de facto metaphor for the majority of our diverse activities using computers, yet the desktop environment provides a one size fits all user interface. This dissertation argues that for the computer to fully realize its potential to significantly extend our intellectual abilities, new interaction techniques must call upon our bodily abilities to manipulate objects, enable collaborative work, and be usable in our everyday physical environment.

In this dissertation I introduce a new human-computer interaction concept, *embodied media.* An embodied media system physically represents digital content such as files, variables, or other program constructs with a collection of self-contained, interactive electronic tokens that can display visual feedback and can be manipulated gesturally by users as a single, coordinated interface. Such a system relies minimally on external sensing infrastructure compared to tabletop or augmented reality systems, and is a more general-purpose platform than most tangible user interfaces.

I hypothesized that embodied media interfaces provide advantages for activities that require the user to efficiently arrange and adjust multiple digital content items. Siftables is the first instantiation of an embodied media interface. I built 180 Siftable devices in three design iterations, and developed a programming interface and various applications to explore the possibilities of embodied media. In a survey, outside developers reported that Siftables created new user interface possibilities, and that working with Siftables increased their interest in human-computer interaction and expanded their ideas about the field. I evaluated a content organization application with users, finding that Siftables offered an advantage over the mouse+graphical user interface (GUI) for task completion time that was amplified when participants worked in pairs, and a digital image manipulation application in which participants preferred Siftables to the GUI in terms of enjoyability, expressivity, domain learning, and for exploratory/quick arrangement of items.

Thesis Supervisor: Pattie Maes
Title: Associate Professor of Media Arts and Sciences

Thesis Readers:

Thesis Reader .....

Joseph A. Paradiso
Associate Professor of Media Arts and Sciences
MIT Program in Media Arts and Sciences

Thesis Reader .....

Scott R. Klemmer
Assistant Professor of Computer Science
Stanford University

Acknowledgments

I have many people to thank. My Ph.D. advisor Pattie Maes encouraged this work from the beginning, even when it was little more than an intuition that Siftables would create new and interesting interaction possibilities. My collaborator Jeevan Kalanithi contributed innumerable ideas, lines of code, and debugging-session help. I appreciate Pattie's thoughtful analysis and feedback that has steered my design in productive directions. Given the departure that Siftables represents from her earlier work, Pattie's support attests to her openness to new ideas and approaches.

I have also benefited greatly from communication with my other committee members, Joe Paradiso and Scott Klemmer. Joe can always be counted on to suggest numerous innovative approaches to tough technical challenges. I consider myself fortunate to have worked closely with Joe, not only because of his earnest *roll-up-the-sleeves* willingness to problem-solve on a moment's notice, but because of his insight about the future of mobile sensing and interaction technology; industry is only recently beginning to adopt innovations that he was prototyping 10-15 years ago. Scott Klemmer has a deep and up-to-the-minute knowledge of the field of human-computer interaction and a masterful ability to identify connections and articulate important distinctions. Scott's efforts toward the democratization of technology development have been a steadfast inspiration to me, and Siftables explores a vision that we share: that hardware might one day be as easy to author as software.

I would also like to acknowledge Ted Selker, my advisor and mentor during my first year at the Media Lab. I have never met Ted's equal in ideation. My work with Ted resulted not only in a number of creative projects; my understanding of, and my ability to brainstorm and prototype ideas rapidly and effectively is forever boosted.

Special thanks as well to key Siftables contributors Seth Hunter, Katya Popova, Evan Broder, Joshua Kopin, Tobe Nwanna, Rick Mancuso, and Laura Harris.

My colleagues in and out of the Context-Aware Computing, Responsive Environments, and Fluid Interfaces groups have been an enlightening community that has helped me define my present understanding of research, invention and Media

Art. I have not encountered another group of individuals that so successfully dovetail innovative research with practical technological *how-to* experience. Special gratitude to Andrea, Win, Josh, Ari, Mat, Mark, Brian, Tristan, Hayes, Amanda, Monzy, James, Chao-Ming, Gian, Rob, Nathan, Oren, Orit, Brandon, Jeff, Dan, Vincent, Jamie, Enrico, Sajid, Seth, Marcelo, Amit, Pranav, Doug, and all my other friends!

The staff of the Media Lab have been invaluable friends and supporters throughout the years, providing steadfast help and generously participating in my studies. Linda Peterson demonstrates time and again that she truly cares for the well being of students and for the cohesion of our community at large.

A number of other mentors outside of MIT have influenced my intellectual journey in important ways. Dominic Massaro at UC Santa Cruz took a chance by hiring me for the summer when I was an undergraduate at Stanford, and in the years since I have learned innumerable lessons about psychology, rationality and navigating academia from our continued friendship and collaborations. Bill Verplank and Max Matthews at Stanford's CCRMA provided the spark that has become my passion for blending technology and music, as well as my ability to work with sensing and electronics. Finally, Cliff Nass's optimism and insights into social science, and Terry Winograd's wise advice afford continuing inspiration and food for thought.

Mom and Dad's progressive parenting philosophy was: *offer love and encouragement and your children will do great things.* I hope my life path is proving them right. My father Jeff taught me to be curious, industrious, civically engaged, to always have a number of projects in motion, and to believe that when confronted with a problem the solution can be arrived at through research and industrious tinkering. My mom Jeanne taught me to be adventurous and curious about other cultures and languages, and to not worry about the small things in life. My siblings Amy and Dan are frequent conversation partners on philosophical matters and are my good friends.

Finally, I am fortunate for the love and companionship of Amy, who thought she was moving to Boston for 2 years, tops. Amy has already stood with me during some of the highest and lowest moments of my life, and our personal philosophies have co-evolved through spirited discussion, travel, and years of shared experiences.

Contents

List of Figures

Chapter 1

Introduction

Research in physical and multi-touch tabletop systems [55] [39], as well as tangible user interfaces [54], has expanded the expressive potential of the computer by enabling more natural interactions with digital content. These systems make our interactions with the computer more physical and gestural, representing physical constraints [93] and enabling us to utilize both hands to navigate spatial information.

However, despite their improvements over the desktop metaphor, tangible and tabletop systems tend to be burdened by one of two problems: Most are either special-purpose systems, built to support a particular type of task and user, or they have infrastructure requirements (i.e. sensing embedded into the work surface or graphical projection) that confine their use to a particular location.

In his seminal 1991 essay in Scientific American, Mark Weiser outlined a vision for the future of computing that predicted an increasing number of electronic devices supporting our everyday activities, including inch-scale computers that he called *tabs* [116]. Weiser discussed the role of these small future computers primarily through the lens of the already-developed Active Badge system [114], which made office environments more responsive to the location of the individual user. He also suggested that tabs with small screens could allow program windows from a computer to be transferred to a tab, so that these program windows could be scattered about a user's desk along with their papers or carried to a colleague's office where the program window would be transferred to the colleague's computer for collaborative work.

Weiser's reference to inch-scale computers with screens foreshadows Siftables. However, my work explores the *user interaction possibilities* with collections of inch-sized devices, a direction that Weiser left largely unexplored. The Active Badge that Weiser helped to develop is not particularly interactive from the wearer's point of view; each badge is a transponder that identifies them to the room, so that the environment can become responsive to their presence and can customize certain services based on their identity and location. Fishkin, Gujar and Want et al.'s work on *manipulative* [40] and *embodied* user interfaces [28] articulated the idea that a personal electronic device could be considered an embodiment of digital content that it displays, and they explored ways that the exterior of the device could afford manipulations of the embodied content. This work was an inspiration, but my research moves in a different direction by focusing on physical interactions with *collections of identical devices.*

Research in distributed cognition shows that physical objects help us think, by allowing cognitive processing to be externalized onto the tools that we use during problem-solving. Furthermore, our hands and bodies are skillful in ways that are under-utilized by computers; we can grasp, push, drag, and scoop individual or groups of objects with great dexterity, actions that are largely ignored by today's desktop and mobile interfaces. This background, discussed more at length in chapter 2 on page 25, was a key motivator for the work.

This thesis presents *embodied media*, a new model for distributed, physically embodied user interfaces. This class of interface comprises a collection of small, physical, stand-alone electronic manipulatives that can represent collections of digital items such as files, variables, or other program constructs. Embodied media devices present visual feedback to the user indicating their current role, and can be physically manipulated as a single, coordinated interface to alter the represented digital items. Unlike today's mobile devices, the model of use is not one device per person. Rather, *a single person interacts with a collection of devices.* Instead of pressing buttons on a mouse or moving their fingertips across a touch-screen, the user interacts with the system physically and spatially by arranging the manipulatives in relation to each other and gesturing with them in continuous, three-dimensional ways. The result is

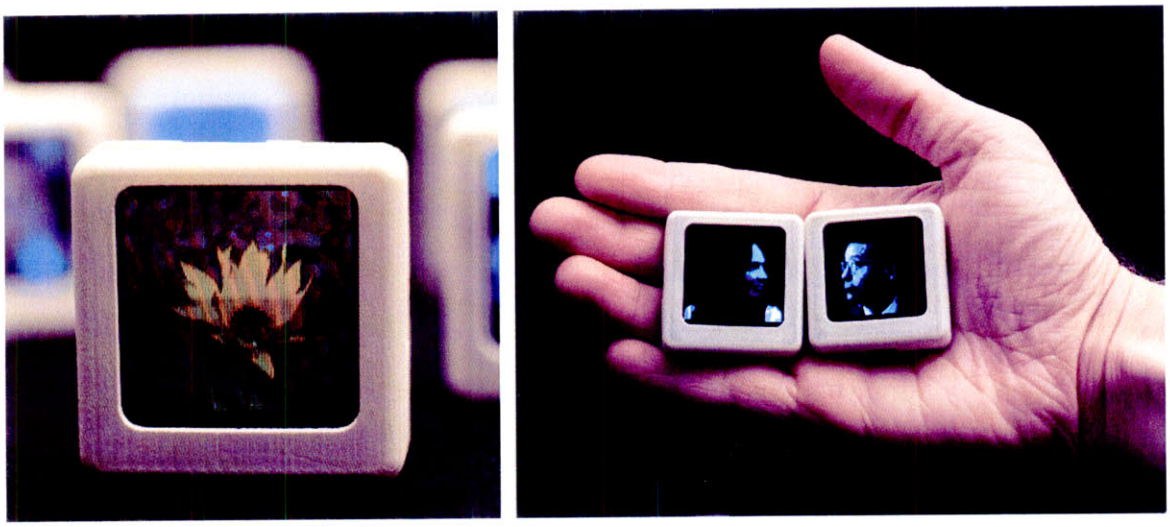


Figure 1-1: Siftables are the first instantiation of an embodied media user interface.

a mobile, distributed physical embodiment of digital information that the user can perceive visually, grasp physically, and manipulate by hand. The concept draws on previous work in tangible and graphical user interfaces, but it brings these pieces together in a novel way to investigate an unexplored point in the design space.

As part of this work I have created Siftables, the first instantiation of an embodied media interface. A Siftable device is a tiny mobile computer that does not require (but than can use) environmentally-installed sensing infrastructure. Each Siftable features a color display, a three-axis accelerometer, infrared-based neighbor detection, a rechargeable battery, flash memory and a Bluetooth radio for wireless communication. Siftables introduces a new class of multi-object interaction techniques that take advantage of our natural ability to quickly manipulate collections of physical items (see chapter 3 on page 47 for details). They can be grasped, shaken, tilted, arranged in a row or arbitrary two-dimensional topology, or moved in expressive gestural ways.

Developers can create applications for Siftables using two distinct application programming interfaces (APIs). The first is a C API for programming the firmware of the devices directly, which allows for completely stand-alone operation. The second is a Python API for controlling a set of Siftables wirelessly from a nearby computer over Bluetooth. The use of a host computer makes it possible to use a large display or other input-output resources in conjunction with the Siftables. I consider both

application forms to be instantiations of embodied media; while the Siftables-only model enables the greatest mobility, a wireless connection to a computer (or to the Internet) is important for many applications in order for manipulations to affect an underlying data model. The notebook computer that I use for development is portable compared to most surface-based interaction systems, however in the future the use of a mobile phone to host Siftables applications could enable even greater mobility.

Siftables combine the flexibility of graphical display with the tangibility of physical manipulatives, and they have enabled the creation of a number of applications. Through my reporting of my design process, feedback from the user studies that I conducted, and related discussion about interactive possibilities and design techniques, I hope that Siftables may act as a reference implementation that points the way to future embodied media implementations.

1.1 Embodied Media: Key Application Types

Applications that may benefit from an embodied media representation feature some or all of the following characteristics: spatial arrangement of items, iterative definition of relationships between content items, collaborative interaction, children or certain special-needs populations as users, non-precise gestural input, and a primary emphasis on the *manipulation* of content rather than its *capture*.

Educational interactions are well-suited to embodied media, for instance language, science or mathematics tools in which learners compose words, molecules, or equations from component parts and receive real-time feedback about the correctness or implications of their solution. Casual entertainment such as puzzle or narrative games can be implemented using embodied media in a manner that adopts traditional play patterns for collaborative or competitive engagement with other players in a uniquely face-to-face manner. Multi-person collaboration is an important style of work and play, so interfaces like embodied media that support parallel interaction are valuable. The manipulative-based interaction style is also useful for young children who are not yet proficient with the keyboard and/or mouse, or special-needs users who are not able

to effectively interact with the standard desktop interface. Expressive interactions for music or video performance may be compelling if the design of the gestures matches the input affordances to the user's precision.

An expanded discussion of these recommendations, including application characteristics that are not well-suited to an embodied media user interface, can be found in section 7.1 on page 162. See chapter 4 on page 85 for a full listing of applications that have been created for Siftables.

1.2 Thesis Contributions

The primary contribution of this thesis is the introduction and characterization of a new human-computer interaction concept, *embodied media.* An embodied media system physically represents digital content such as files, variables, or other program constructs with separate interactive electronic tokens that can display visual feedback and can be manipulated physically by the user as a single, coordinated interface. Unlike tabletop or augmented reality systems, an embodied media system relies minimally on external sensing infrastructure. Compared to most tangible user interfaces an embodied media system represents a more generalizable platform. My characterization includes both a listing of essential features that an embodied media system requires, and a summary of novel interaction possibilities and application types that embodied media systems are well suited to support.

Siftables is an interactive system of programmable electronic manipulatives that was constructed to explore the embodied media concept. Its combination of sensing, embedded computation, graphical display and wireless communication built into each device is a novel system design that allows Siftables to support a range of interactive application scenarios. The contribution of Siftables is the construction and critical discussion of a working embodied media instantiation.

Siftables also enables a number of novel multi-object interaction techniques that are not possible, or that require additional infrastructure in other systems. These techniques leverage the collective sensing, coordinated graphical display, and wireless

communication capabilities of the manipulatives. Siftables can sense adjacency of neighboring Siftables, tilting, shaking and other motion. Their feedback capabilities include on-manipulative color graphics and audio triggered on a wirelessly-connected computer. The ways that Siftables can be manipulated together in concert allows this work to explore a new point in the design space of human-computer interaction technologies, and I identify and discuss a number of of novel multi-object interaction opportunities.

Along with my colleagues I have created a number of applications using Siftables, including an image manipulation system, a word-finding game, an equation editor, a graph-topology creation tool, an interactive cartoon narrative system, and various other game and creativity support tools. These application examples primarily validate the utility and flexibility of Siftables. They also provide the background for my identification of application types that embodied media interfaces such as Siftables are particularly well-suited to support. Along with my discussion of novel multi-object interaction opportunities, this work offers guidance for other implementors of embodied media user interfaces..

Siftables was designed to enable other developers can create applications with the platform. More than twenty researchers in industry and academia have used Siftables' high-level Python API to explore embodied media. Siftables thus contributes a multi-purpose, reusable platform for exploring embodied media interaction possibilities, and the feedback I collected from these developers contributes additional insight into the interaction possibilities of embodied media.

Finally, I conducted studies to measure qualitative aspects of the user experience using Siftables as well as the task efficiency implications of an embodied media system compared to the mouse/GUI. The overall findings of these studies were that participants preferred Siftables to the mouse/GUI in terms of enjoyability, expressivity, domain learning, and for exploratory/quick arrangement of content items, and that Siftables offered an advantage over the mouse/GUI for task completion time (particularly when participants worked in pairs). These findings, and my subsequent discussion, are the final contribution of this thesis.

1.3 Dissertation Roadmap

Chapter 2 covers the background and motivation for this thesis, beginning with findings from cognitive psychology that suggest the advantages of physical tools for interaction. It then reviews an abbreviated history of gestural interaction with electronic media, with special consideration given to tangible interaction systems that consist of collections of physical manipulatives. The chapter continues with systems that enable rapid prototyping of physical interfaces, and closes with a discussion of ubiquitous computing and some user interface trends in contemporary mobile technology.

Chapters 3 through 5 cover the design process, applications, and the technical details of Siftables. Chapter 3 explains the initial design inspiration and the prototypes that were built on the way to the current system. It also contains a discussion of the gestural language possibilities of an embodied media system. Chapter 4 continues with a full listing of the applications that have been created for Siftables and a number of shorter "application sketches" that explored particular interaction ideas. Chapter 5 presents implementation details including Siftable hardware, firmware, and the high-level Python application programming interface.

Chapter 6 describes the methods, results and discussion of one pilot and two user studies that provide quantitative and qualitative feedback about Siftables compared to a mouse/GUI system. Chapter 7 presents takeaway lessons about the types of interactions that are well-supported by an embodied media user interface, along with a number of design suggestions for embodied media. The thesis ends with a look toward both the future of Siftables and embodied media and to our future interactions with computers as collections of networked "smart" objects.

Chapter 2

Background and Motivation

This chapter begins with a discussion of the typical tradeoff between physicality and flexibility in a user interface. From there it continues by presenting evidence from cognitive psychology for the advantages of physical tools for interaction. It then reviews an abbreviated history of gestural interaction with electronic media, beginning with the 1920's-era Theremin and moving to modern systems. Special emphasis is given to tangible interaction systems that utilize collections of physical manipulatives, since these interfaces provided inspiration and contrast during my design process. The chapter continues with a look at the progress in recent years toward enabling easy and rapid prototyping of physical interactive systems, and closes with a discussion of contemporary mobile, ubiquitous personal technology, putting forth both a philosophy for design of these systems going forward and an argument about why the present is an advantageous moment in history to be building them.

2.1 Introduction

Throughout most of human history, the tools we have used have been purely physical. People have developed deep skill in using tools, and utilizing the physicality of the tool has been a key advantage. From tools for manipulating physical matter such as stone adzes, augers, and looms, to tools for manipulating abstract items like the abacus or slide rule, a unifying theme has been that the shape of the tool largely

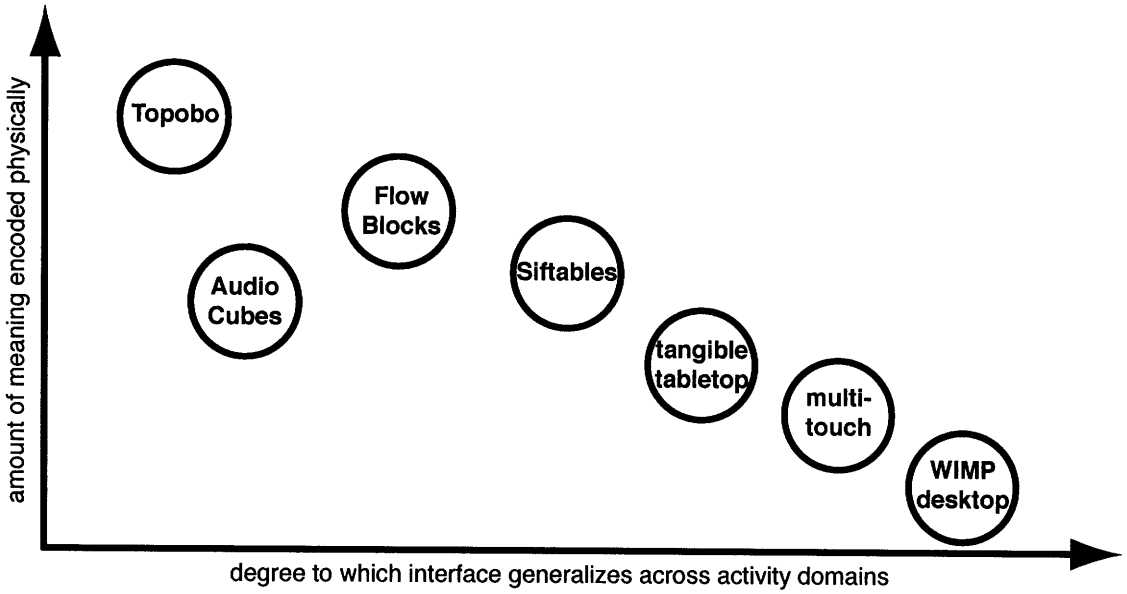


Figure 2-1: Physical representation versus ability to generalize. This plot situates selected human-computer interfaces in a two-dimensional space, at locations characterized by the amount of meaning that is encoded in the physical form of the interface (vertical axis) versus the degree to which the interface generalizes across activity domains (horizontal axis). Scale is intentionally omitted; the general positions are what I intend to communicate.

determines its possible uses. An auger does not make a very good adze, nor does it make a very good abacus, and vice versa for all of the aforementioned. At the risk of over-generalizing, I suggest that most physical tools on their own have limited versatility. A tool like the Swiss army knife is more versatile than most as it is an agglomeration of smaller individual tools. Other more generic tools, such as a hiking stick, are more versatile due to their unspecific form.

The introduction of the computer was a step forward in the versatility of tools for manipulating information. A software program can change the behavior of the underlying machine effortlessly, and graphical displays now provide inexpensive and flexible visual output. Pixels are cheap, and extremely malleable. The machines on our desks can at once be calculators, spreadsheets, audio recording workstations, word processors, Internet browsers, video editing systems and more. The tool seems nearly infinitely adaptable. However, compared to an adze, a hammer, or even an abacus, the computer is not a particularly physical tool. Typing on a keyboard and

moving a mouse uses our skills for manipulating physical objects with our hands to some degree, but this usage is limited compared to our rich heritage.

Tangible User Interfaces (TUIs) have been developed in the past fifteen years as a way to re-physicalize our usage of computers. From early work at Interval Research [109], to Hiroshi Ishii's research at the MIT Media Lab [54] and Scott Klemmer's research at UC Berkeley [60] and Stanford [41], many new physical interfaces to computation have been explored. Some of this work follows the old-tool pattern of designing specific form-factors for specific tasks, while other systems keep the tools generic but project graphics around them to give them context-specific meaning. I believe that there is an inherent tension between the amount of meaning that is encoded into the physical shape of an interface, and the degree to which the interface generalizes across activity domains. See figure 2-1 on the facing page for a depiction showing how I believe some selected human-computer interfaces are situated in a space defined by this tension. I this figure do not make a claim about how these properties can be measured or what the numerical value of each coordinate would be. Rather, the arrangement of the examples with respect to each other is intended to illustrate the general point that there is a tension between the amount of meaning that is encoded into the physical shape or affordances of an interface, and the degree to which the interface generalizes across activity domains.

The point in the design space that my work explores is the generic physical tool with a graphical skin and sensing capabilities. This point represents a hybrid of the physical tool and the graphical user interface; a smart, physical-digital instrument that can sense various forms of user input and can display information on a built-in graphical display.

The rest of this chapter will examine the background and motivation for this design, and related work.

2.2 Brains, Hands, and Objects

This section examines some psychological literature that relates to problem solving using physical objects.

2.2.1 Distributed Cognition: How Artifacts Aid Thought

There has been a great deal of recent interest in physical user interfaces for computers. A key reason is that physical user interfaces have certain unique benefits, some that can be explained by theories of cognitive science. One such theory is *distributed cognition*, developed in the early 1980's by Edwin Hutchins [49]. The premise is that people can *externalize* working memory and cognitive processing onto the objects or tools that they use during problem solving activities. These objects, whether they are physical or virtual, *help us think* about problems [120], and support us in solving problems more effectively. Don Norman puts it nicely in his book "Things That Make Us Smart": *...the more information present in the environment, the less information needs to be maintained within the mind* [86].

For example, Kirsch and Maglio observed that expert players of the video game Tetris made *more rotations on average* of each piece before dropping it to the bottom of the game area, compared to novice players [70]. Although counterintuitive at first, the connection between greater expertise and a greater number of rotations suggests that in-game rotation, where the player can view the result, is less cognitively expensive than mental rotation of the pieces.

In a related study on a word-finding task using Scrabble tiles [71], Maglio et al. found that participants that were allowed to re-arrange the tiles had more success identifying possible words than those who were not allowed to move the tiles. This finding reinforces the idea that objects can be used to offload cognitive processing; rather than having to imagine possible letter sequences, participants re-arranged the tiles to spell out words, and this ability to re-arrange the solution space proved to be helpful. In this work and subsequently, actions that a problem-solver takes to re-arrange the environment to aid their problem-solving process are termed *epistemic*

actions, whereas actions that make a direct step towards the solution of a problem have been termed *pragmatic actions* [57]. For example, placing certain letters next to each other that seem likely to form a word fragment (without having a full word yet identified) would be considered an epistemic action, since some word-recognition effort would be offloaded into the completed "chunk." Assembling a complete word would be a pragmatic action.

These studies suggest that human-computer interfaces that allow a solution space to be re-arranged easily by the user provide a likely benefit for problem-solving activities. Recognizing this, Fitzmaurice posed an important question in his dissertation: *We have the potential to rapidly manipulate physical artifacts. The question is does the UI provide us with the affordances to utilize this potential?* [29] (section 2.1).

The speed of manipulation that a user interface affords can also impact the efficacy of an interaction. Even small differences in the amount of time that an interaction takes can have a profound impact on the type of strategy that is employed by a user, impacting the quality of the solutions [36]. Fitzmaurice made the same point about virtual objects used as cognitive aids: *...if the amount of effort and attention needed to manipulate these virtual objects is high, it may outweigh the value of using them as external cognitive aids* [29].

Another related cognitive science experiment [120] found that the form of objects used in the classic Tower of Hanoi problem impacted the speed of participants' problem-solving and the accuracy of their solution. Objects that encoded constraints or rules of the problem in their shape provided a time advantage and reduced errors. Mackay observed that air traffic controllers use paper strips to work together, checking their position and markings repeatedly and sometimes even annotating strips simultaneously [69]. These finding suggests that user interfaces to computers that can represent problem constraints in a manner that is perceptible to the user and relevant to the style of manipulation, and that permit collaborative manipulation, may have similar advantages. While Siftables feature a generic physical shape, the visual display capabilities of embodied media manipulatives allow them to visually encode some problem constraints.

2.2.2 The Rich Biomechanics of Human Hands

Our hands are skillful, allowing us to manipulate physical objects in ways that are not yet well-utilized by today's interfaces to computers. Guiard's Kinematic Chain Model provides an analysis of role differences between hands when both hands are used together. The summary of Guiard's framework is that for the majority of manual acts, the hands act in concert in an asymmetrical, complementary manner. The non-dominant hand is typically used to hold the manipulated object in place, creating a frame of reference that the dominant hand works in to take action on the object. The dominant hand tends to move more frequently and with greater precision than the non-dominant hand. Finally, the action of the non-preferred hand often precedes the action of the preferred hand. These findings should be familiar, for example from the everyday experience of writing on a sheet of paper, and they apply to many other manipulation activities (summarized from [44]).

The implication of Guiard's analysis on the current work is that by enabling both hands to take different roles in manipulating digital information, Siftables may support a greater degree of skillful interaction than systems that do not enable two-handed manipulation. Multi-touch systems typically support two-handed interaction, but they lack certain physical feedback advantages enjoyed by tangible interfaces.

An interesting possibility is that role differences in a Siftable-based activity may be distributed into different manipulatives. See the color-mixing interaction sketch as an example (section 4.12.4 on page 109), wherein several Siftables show colored paint buckets on their screens, and another Siftable starts out with an image displayed. When a paint-bucket Siftable is placed next to the image Siftable then tilted toward it, a *pouring* action occurs wherein the given color is added to the image Siftable for as long as the paint-bucket Siftable is tilted. To relate this example to Guiard's model, the image Siftable is the reference frame and the dominant hand holds the tilted Siftable as a way to take action on the object (the image). This example is inspired by the way we pour water from a jug held in the dominant hand into a cup held in the non-dominant hand.

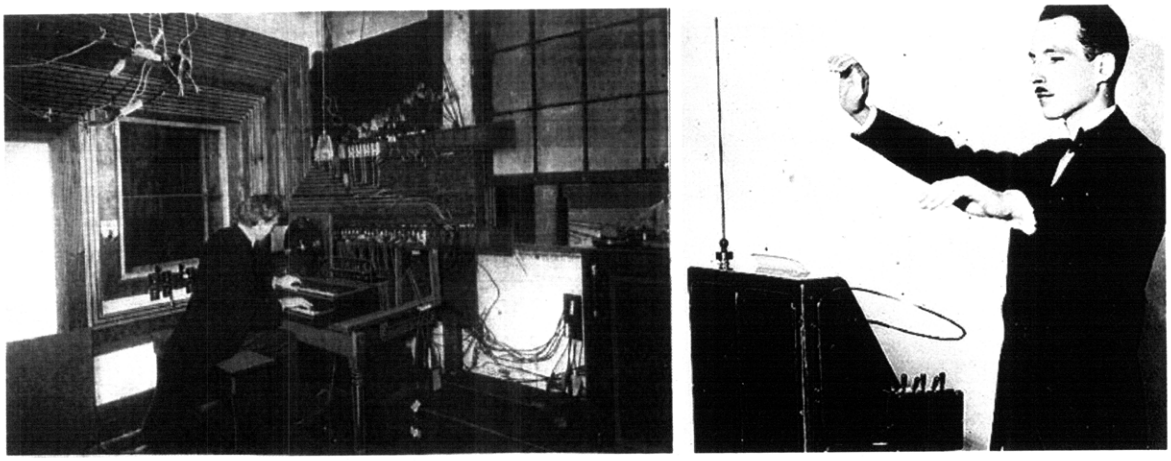


Figure 2-2: Early Gestural Interaction: The Telharmonium, developed in 1897 (left), and Theremin, developed in the 1920's (right) are examples of early electronic musical instruments that supported expressive gestural interaction.

Support for collaboration is the other key advantage of a multi-manipulative interactive system, since more than one person may simultaneously grasp, arrange, and otherwise manipulate elements of the interface in parallel. The ability to work collaboratively can be helpful in problem solving and creative work, but is not well-supported by typical desktop computers.

2.3 Gestural Interaction with Computers

Our ways of using the computer to manipulate information have changed significantly a number of times over the past fifty years. Early interactions with computers for 'number-crunching' tasks such as code cracking and missile trajectory calculation were defined by discrete inputs. The first widely available personal computers were programmed by a series of toggle switches on a front panel [81]. However, beginning with the light pen in the mid-1950's [111] and followed by the mouse in the late 1960's [25], the computer's ability to sense and utilize a user's continuous physical gesture has opened up possibilities to make it a much more expressive, nuanced tool.

2.3.1 Gestural Interaction with Electronic Media

Gestural interfaces that provide continuous input to electronic systems predate the digital computer. The keyboard-based Telharmonium [115] musical instrument from the late 1800's may have featured expressive foot-pedal controls, though little is known about the details. The first non-contact gestural interface to an electronic system was a musical instrument called the Theremin [34]. The Theremin allowed a player to modulate the pitch and volume of an auditory oscillation by varying the distance between their body and the instrument. Standing behind the instrument as behind tiny podium, the hands of the player carefully dance up and down, forward and back, never touching the instrument itself while creating an oscillation that can change as nimbly and expressively as the human voice.

It is interesting to note that technical advancements in *musical instruments* have foreshadowed later progress in human-computer interaction. For instance, the first "button" [1] was probably the hole of a flute, since the change in pitch when the hole is covered or uncovered is a mode switch. Considered in this way, the first button may have appeared 9000 years ago, as prehistoric flutes have been found from that era [119]! Other electronic music interfaces such as Hugh LeCaine's Electronic Sackbut (1948) [90] had pressure-sensitive panels that allowed a player continuously modulate volume, pitch and timbre. It may have been the continuous nature of analog electronics that encouraged the creators of these early musical instruments to support expressive gestural interaction before the arrival of the digital computer. As discussed in the next section it was not until the 1950's that digital computers learned to sense continuous gesture, and not until the Macintosh with mouse emerged in the 1980's that gestural interaction with computers became widely available. These examples of arts-oriented inventions making strides that predate and perhaps inspire advances in information technology may be instructive as we seek ways to transform the personal computer into an ever-more capable tool for personal expression.

[1] Here I will use a loose definition of a button, considered to be any mechanism by which a discontinuous state-change of a system may be affected by some physical pressure applied by the user.

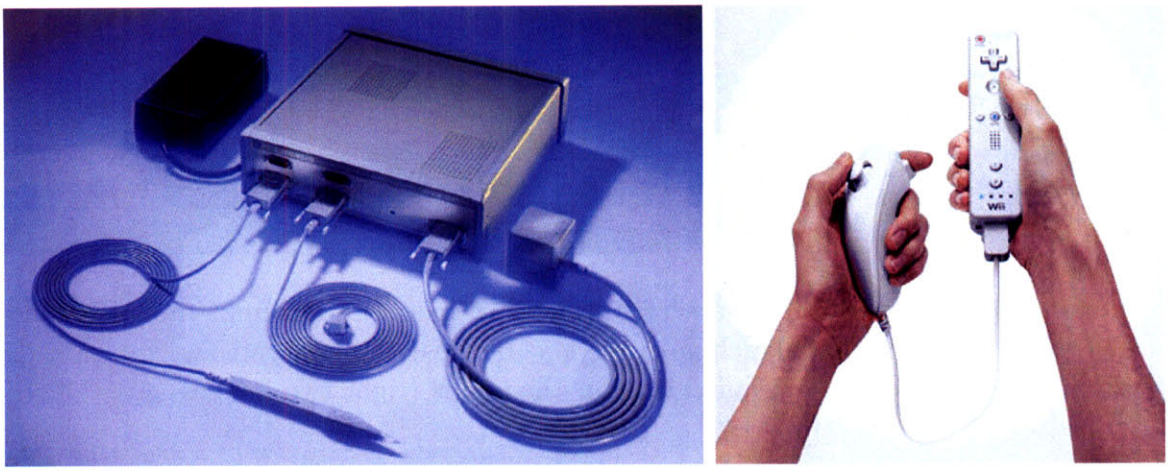


Figure 2-3: The Polhemus (left) is a system that permits three-dimensional position sensing for interactive applications. The Nintendo Wii (right) controller is an inertial motion sensing and pointing device.

2.3.2 Free Gesture Interfaces

The term "free gesture" is used here to describe gestural input to a computer that is unconstrained by physical contact with a fixed surface or object. The Theremin was the first example of free-gesture input to an electronic system, however it did not gain wide popularity. A possible reason is that free-gesture systems, although expressive, are known to be poor input devices for precise control [88]. In the years since the Theremin, a number of other systems have been created that feature free gesture input to a computer.

The Polhemus FASTRAK is a 6 degree of freedom (DOF) system for tracking the absolute position and orientation of a small sensor unit. It has been used in a number of human-computer interface systems such as [46] and [75]. A limitation of the FASTRAK is that the sensor requires a wired connection to the base unit; a subsequent product release [97] makes the sensor unit wireless, but requires nearby receiver modules that are connected by wires to a base station.

More recently the Nintendo Wii game console [51] has popularized a baton-like free gesture interface as an input for video games. During game play, up to four players use wireless game controllers featuring 3-DOF inertial (acceleration) sensing as well as a number of discrete buttons. Each wireless controller has an optional

Figure 2-4: Direct pointing at a screen or surface has been an input modality since early stylus systems like Sketchpad (left two images). Touch pad and multi-touch systems have removed the requirement for the stylus (right two images).

secondary inertial input connected by a cable that also provides 3-DOF acceleration input for use by the player's other hand. The Wii has been a commercial success, and a factor that may contribute to the popularity difference between the Wii and the Teremin is that games for the Wii are designed to largely obviate the requirement for precise control.

2.3.3 Gesture on a Surface

Two-dimensional, surface-oriented interactive systems are another class of gesture-based interfaces that are relevant to the current work. Stylus-based tools for position sensing across a surface are the oldest example of this category. The light pen of the 1950's was the first of these interfaces, and modern drawing tablet and tablet computer systems still feature a pen-like stylus that provides absolute position sensing of a single point of contact across the tablet or screen's surface. The mouse is also in this category, allowing for two-dimensional *relative* sensing of a single point of contact with a surface. Both mouse and tablet+stylus interfaces also feature discrete button-based input along with 2D gesture.

Fingertip-sensitive touch pads, such as those made by ALPS [68] and Synaptics [112], have become integrated into most modern laptop computers. These interfaces do not require a stylus; using a capacitive sensing grid, they track the position of one or more of the user's fingertips to control the mouse cursor or zooming and scrolling of on-screen content.

Multi-touch surfaces have also become popular in recent years. Although the first multi-touch interface appeared in 1982 [76], it was Han's large-surface multi-touch demos [39] and Apple's introduction of multi-touch on their iPhone device [53] that has recently brought this interaction technique to the public's attention. A drawback of surface-based interfaces is that they typically require visual attention, since the smooth planar surface itself does not provide any tactile feedback to the user [17].

2.3.4 Final Thoughts on Gesture

In summary, interfaces that permit continuous gestural input can support expressive control in ways that discrete button-based interfaces cannot. The tradeoffs are that free gesture has limited affordances for precision and thus may be most successful in gaming systems such as the Wii or in other scenarios where the task can be designed or adapted to avoid requirements for high precision input. Mice and tablet interfaces both offer single-point of contact interaction; a key difference between these two is that whereas the mouse-based interaction separates the locus of interaction (the desk) from the location of the visual feedback (the screen), tablets and touch screens feature visual feedback that is *co-located* with the input. This co-location of input and output can create an increased sense of connectedness between the user's actions and the corresponding manipulation of the content [28]. However, multi-touch interaction on large displays requires extra visual attention from the user, and the interaction is typically confined to a two-dimensional plane.

2.4 Reducing the Cost of Developing Interactive Physical Systems

A great deal of progress has been made in the domains of web and personal software towards enabling people to more easily author their own online content and software programs. Wiki and blog infrastructure allow people to create web pages without having to learn HTML, and toolkits such as d.mix [42] lower the barrier to entry

for crafting programmatic online behavior. Additionally, programming environments like Flash [50] and Processing [32] allow people to write interactive software that runs on the web or on their local machine with only an introductory-level background in programming.

A key challenge in technology design and development is that physically-embodied (i.e. hardware) systems are much more difficult to create and to author behavior for than software systems [59]. Building a new electronic device requires tools and supplies beyond the personal computer, and answers to questions and problems are not as easily found on the Internet, making troubleshooting more difficult. The result is that there are few tangible platforms that can be reused for multiple applications. Toolkits have emerged that begin to meet this need, such as Bug Labs (prototyping personal consumer devices) [62], Lego Mindstorms (modular robotics) [8], iStuff Mobile (physical user interfaces for mobile phones) [6], and Arduino (microcontroller and sensors prototyping) [77]. Each toolkit addresses a certain class of device or system, but none specifically addresses distributed, embodied media user interfaces. Siftables have been designed as a general-purpose tangible prototyping platform to support the creation of a range of user interface ideas, and as such they make a contribution in reducing the difficultly of exploring the design space of hybrid tangible-graphical user interfaces.

2.5 Tangible and Tabletop Interfaces

George Fitzmaurice's Ph.D. dissertation [29] introduced the idea of a "graspable" user interface, outlining the benefits of a system featuring multiple physical manipulatives[2] that can each be distinct both in visual appearance and function. Since this work, there have been a number of examples of what I will call *tangible tabletop interfaces*, systems that comprise physical "handles" on a display surface that provide a means to manipulate digital content, such as d-touch [19], reacTable [55] and Sensetable [94].

[2]In educational circles, the term 'manipulative' describes any physical object that is specifically designed to foster learning. Here I use a technology-oriented definition, wherein a manipulative is considered to be a physical object that affords some interaction with digital information.

Many of these systems have a similar structure and interaction style: they feature inert manipulatives whose positions (and in some cases, orientations) can be sensed by the system, and graphics superimposed onto the work surface. For instance, reacTable features cubes with fiducial markers (patterns that are visually distinct) on each face. Computer vision software operates on the video feed captured by a camera installed under the clear surface of the table to determine the identity and position of all blocks that are currently in the work area. Users interact by moving the cubes to create networks of inter-connectivity in a graph-like structure that is projected visually, and the system produces audio in response. In the case of reacTable the end result is an audio synthesizer.

Earlier variants on the tangible tabletop paradigm include Wellner's Digital Desk [117], a tabletop system that projected graphics onto a desktop work surface, allowing the user to interact by pointing with the fingers at real paper documents. Another was the wall-mounted Designers' Outpost [60], that featured post-it notes as physical manipulatives and allowed them to be hand-annotated. The focus of the Designers' Outpost was the seamless integration of physical and virtual editing capabilities, as individual notes could be edited or removed, whilst remaining in the digital representation of an ongoing design session. The Tern tangible programming language [48] comprises inert but physically and visually distinct interlocking wooden manipulatives. The user creates a program by locking a set of statements into a sequence, and when the sequence is photographed it is parsed from the single snapshot using computer vision software, then the associated instructions are executed.

These tangible tabletop interfaces all require *environmental infrastructure*: either sensing apparatus built into the table, cameras above or below the surface, or some combination of the two. These infrastructure requirements impose a tradeoff; they permit the systems to sense the absolute (workspace-relative) position of the manipulatives, and in some cases to display graphics surrounding the manipulatives. However, these features come at the cost of portability and directness. Since the sensing is required, the manipulatives must be used in the two-dimensional plane of the instrumented workspace and usually have no utility elsewhere; some systems

such as reacTable and Sensetable lose their ability to track the object at a height of only a few centimeters from the surface. Furthermore, in some of the aforementioned systems the digital content is projected *around* the manipulative, rather than the manipulative *displaying the content directly.* A system that can display graphical content *on-manipulative* can be more mobile since it does not require use near a projector for the graphics to be seen, and the feedback can take the form of an "information skin" directly on the object rather the manipulative being a "handle" [30] to a separate projected item.

2.5.1 Minimal-Infrastructure Distributed Tangible Interfaces

A few systems have emerged that attempt to free tangible tabletop manipulatives from environmental infrastructure like position-sensing and graphical projection. These systems build more functionality into the manipulatives themselves, or interface the manipulatives to a nearby computer in a more lightweight manner that allows them to more easily be used in different locations.

The Tangible Music Sequencer [10] and Flow Blocks [122] are examples of this class of minimal-infrastructure systems. The Tangible Sequencer [10] in particular is an interesting precursor to the current work because its blocks have both local (neighbor-to-neighbor) and longer-distance (radio) communication. However, the Tangible Music Sequencer has been applied only to a single domain: electronic melody sequencing. Sony's Block Jam [84] is a similar system, wherein cubes with low-resolution Light-Emitting Diode (LED) arrays on top can show iconic graphical feedback and can be arranged into two-dimensional patterns to create musical sequences. Like the Tangible Music Sequencer and Block Jam, Flow Blocks are an interface designed for a specific activity; the purpose of Flow Blocks is to allow children to explore complex causal relationships and to understand their analogical relationships to the dynamics of real world systems.

Zigelbaum's Tangible Video Editor [121] is another example of a minimal-infrastructure tangible system. Zigelbaum configured modified Compaq iPaq PDA devices to represent individual video clips, and users could create an edited movie sequence by

aligning the iPaqs end-to-end. Since the PDAs have screens, Zigelbaum was able to exploit the flexibility that comes from the introduction of graphics capabilities on the manipulatives, and to represent video clips in a manner consistent with their underlying graphical nature. However, the Tangible Video Editor was a single-task-domain system, with affordances designed specifically for video editing.

Exploring distributed topological interaction for non-block forms are interfaces like Triangles [35] and Glume [91]. These construction kits allow a user to connect pieces into three-dimensional shapes, and the system captures the topology of the interconnected elements. Topobo [100] is a similar system that does not capture the *topology* of the three-dimensional construction, but that records motions to the structure applied by the user, then can actuate to play back the recorded motions.

These minimal-infrastructure tangible interfaces are important steps towards true general purpose systems. They are more portable than their tangible tabletop predecessors yet they preserve the utility of physical graspability. However, their design for single-activity usage limits their ability to explore the wide range of possibilities permitted by a general-purpose distributed physical interface such as Siftables.

2.5.2 Concluding Tangible User Interface Thoughts

This section reviewed distributed user interfaces and a number of specific tangible and tabletop systems with varying degrees of infrastructure requirements. A key advantage of these systems' physical *graspability* is the implicit feedback that comes from manipulating a real object, as compared to a touch screen interface with purely graphical items. However, the typical *handles-on-a-surface* instantiation of tangible tabletop systems (see section 2.5 on page 36) is still not as direct as an embodied media interface like Siftables wherein the manipulative itself can both sense the user's manipulation *and* display graphical feedback. Furthermore, interaction with most tabletop interfaces is limited to the two-dimensional plane of the work surface, and they require environmentally installed sensing infrastructure to operate, limiting their mobility. Those systems that do not require significant infrastructure have so far explored relatively specific usage scenarios.

2.6 Distributed Media: Complex Behavior from Collections of Simple Pieces

The idea that interesting global behavior can emerge from a collection of relatively simple, locally interacting computational pieces dates back to cellular automata (CA) simulations [118]. Inspired by the behavior of crystals, ant colonies, beehives and other collective phenomena from nature, the authors of early CA software explored ways that simple rules for the behavior of individual nodes can form complex behavior in aggregate when these nodes are allowed to interact with each other. John Conway's game of life [18] popularized the concept and spawned thousands of software implementations by programmers worldwide.

Paintable computing [15] brings a theme similar to cellular automata into the physical domain. It posits the possibility that individual computational devices might be so small as to be suspended *en masse* in a viscous fluid and literally painted onto a surface by a user, where they would harvest energy from their environment, establish radio communication with their neighbors, and collaboratively become a dynamic ad-hoc computer and (in some cases) a graphical display. Paintable computing has not yet been realized at the desired size and scale, but larger (sensor-network sized) nodes have been built to prototype the behaviors that a paintable computer could exhibit, notably the Pushpin computing system [66]. Other distributed computing paradigms include object-oriented architectures, and systems that are distributed across the Internet.

Embodied media explores a different style of distributed system, with a fewer number of components than cellular automata or paintable computing, each typically configured to exhibit *different behavior*. Furthermore, neither cellular automata nor paintable computing were imagined to be particularly *tangible* systems for interaction. A few projects have explored diffusion of media using tangible, mobile devices. Kramer's master's thesis explored custom tangible manipulatives that could be used to interactively transmit mobile code to each other based on their adjacency [61]. The iBall experiment allowed children to create small interactive programs that were

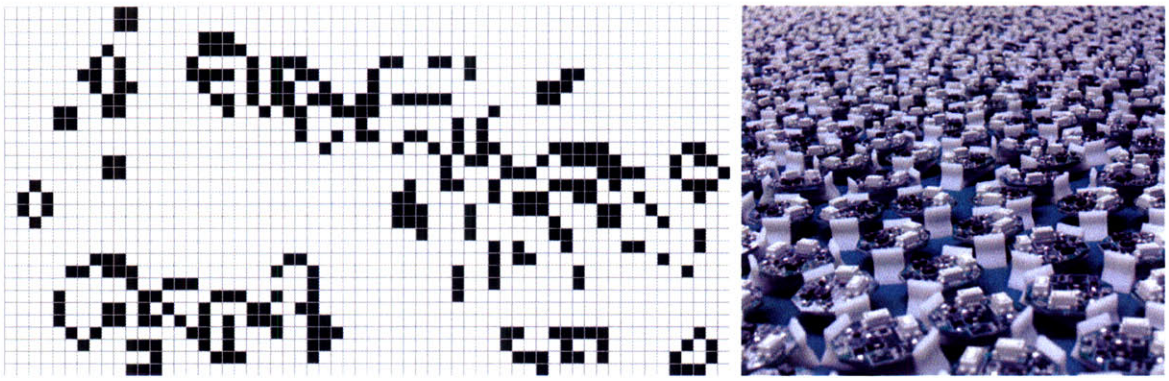


Figure 2-5: Conway's Game of Life (left) demonstrated how complex behavior can emerge from a distributed system of interacting nodes with simple rules. Butera's dissertation on *paintable computing* developed a working physical instantiation of a distributed physical system.

shared in a peer-to-peer fashion using a modified version of the SEGA DreamCast platform [12]. The following sections look towards human-computer interaction with distributed systems.

2.7 Mobile and Ubiquitous

Other research threads that feed into this dissertation are mobile systems that can comfortably operate away from the familiar desktop computer setting, and ubiquitous systems that introduce many computational devices into the environment. This section will examine the important ways that these systems differ from the desktop scenario, and the possibilities that they offer to my work.

2.7.1 Enabling Advances in Technology

The timely convergence of a number of technological advancements makes us now uniquely positioned to explore the implications of mobile and ubiquitous computing. The implication of Moore's Law [104] is an exponential increase in computational power that allows mobile phones and other portable devices to run sophisticated user interfaces driven by small processors whose clock speeds were characteristic of typical desktop computers just a decade ago.

In addition to processing power, other factors have converged to allow discontinuous advances in mobile interaction with information. Wireless communication has seen a number of changes, from more efficient transmit and receive circuitry that reduces power usage and extends battery life to algorithms for reliable mesh connectivity between sensor network nodes. Recent progress in sensing technology has also permitted interesting new possibilities. Particularly, the maturation of MEMS (Micro Electro-Mechanical Systems) based accelerometers, first made by Analog Devices [22], has allowed these sensors to become a standard way for mobile phones and digital cameras to sense orientation and motion during use.

These advances in sensing, processing and wireless communication have enabled the creation of mobile phones, sensor networks, and a number of other wirelessly-connected devices like pacemakers [110] and portable game systems [85] that enable health monitoring, access to information, and play away from the desktop environment. The next section will examine the current landscape and recent progress in mobile human-computer interaction.

2.7.2 Good Mobile UI is Not Just Mobile WIMP

Due to their limited processing and display capabilities, early mobile devices such as pagers and first-generation mobile phones featured minimal, text-based user interfaces. User interaction with these devices comprised pressing buttons to navigate simple menus and contact lists. As mobile processing and display capabilities improved, the interfaces on mobile devices began to more closely emulate the standard windows, icons, mouse and pointer (WIMP) paradigm from the desktop. However, the use of space in a WIMP system can become awkward on a system with no mouse such as today's mobile phones. With only a keypad and perhaps a "joystick" directional control, extra reliance on visual feedback for mobile phone interaction became the norm, for instance highlighting the background of the currently selected desktop icon.

Touch screens have been brought into service as a more graceful solution to the problem of navigating spatial information on small displays. Position-sensitive touch sensing on a graphical display is not a new technology. An early example is the

PLATO IV Touch Screen Terminal (1972), an instructional system that allowed students to answer questions by touching anywhere on the screen. In 1992, IBM and Bell South made a mobile phone with a touch screen, preceding Apple's iPhone by more than two decades [17]. However, the falling costs of sensing architecture supporting this technique and the maturation of the related algorithms, particularly for multi-touch, have resulted in an explosion in the number of recent mobile devices that use a touch screen. The influence of the interaction designer in this process should not be underestimated. In the case of multi-touch interaction, Han popularized the technique by demonstrating a series of compelling interaction sketches and applications in online videos and live presentations[39]. Apple has now made very similar interaction techniques standard on their iPod and iPhone devices [53].

The example of the touch screen can be a useful case study in how a new technology or sensing technique may provide an advantage in a mobile context even before it finds wide usage for the desktop computer user. The design challenges of mobile interaction (i.e. small devices, tiny screens) make the incorporation of new technologies and the development of specialized techniques critical to progress in usability. However the integration of touch screens into mobile devices may be a bandage for a wound that actually requires stitches, or even major surgery. The next section looks at some recent examples of more dramatic ways that mobile interaction his being reconsidered.

2.7.3 Mobile User Experience: Unique Challenges and New Directions

The reasons why human-computer interaction with mobile devices is fundamentally different than interaction with a desktop computer go beyond the surface differences of a smaller screen and more limited keypad. Important differences between the two scenarios relate not only to the device itself, but additionally to the contexts of use [89].

In addition to the spatial problems presented in the previous section, the graphical user interface also relies on the assumption that the user can devote undivided visual

attention to the screen of the device. This assumption may not be valid when we consider the unique constraints on the mobile user with respect to attention and device manipulation [13]. She may be driving a car or riding a bicycle, both of which will place restrictions on how much visual attention she can devote to the device itself (possibly none), and that will also restrict her interaction with the device to one hand or fewer! Even if both hands are free, she may be walking down a busy sidewalk or talking to a friend, reducing the amount of attention she can pay to the device.

A growing body of research in mobile HCI seeks to address the unique challenges posed by mobile technology use. New approaches have included the use of spatialized audio menus and head motion [96] as input, the sensing of wrist gestures [27] hand poses [101] and body-relative spatial motions [3], vibrotactile feedback [98], and improvements to common mobile tasks such as traversing a list of contacts [40] [87].

An even more radical response to the challenges of mobile interaction is a flexible device that responds to bending rather than buttons [105], largely dispensing with the WIMP paradigm. Another prototype input device discerns the user's scratching and other physical contact with its textured surfaces [83] by listening to the audio signature of these interactions, allowing the device to remain in the user's pocket during use as they operate it completely by feel.

These recent prototypes are only the beginning of a dramatic re-conceptualization of the mobile user interaction experience, and I believe the most interesting work is still yet to emerge. Much future work is technically possible today, but not yet imagined by interaction designers. Humankind is at a moment in history featuring a sea of new technological possibilities, we just need to develop more creative ways to imagine how we might leverage them. The next section examines sensor networks, one such area of technological development that I believe is ripe for application to problems of human-computer interaction.

2.7.4 Wireless Sensor Networks: Minimal and Distributed

There has been a great deal of research activity in academic and industrial settings around wireless sensor networks (WSNs). The typical WSN features a collection of physically separate devices with sensing, computation and wireless communication abilities that cooperate to perform a wide variety of tasks. They are capable of exhibiting coordinated behavior, forming a kind of "functional fabric" in the spaces that they inhabit. Pister's "Smart Dust" work at U.C. Berkeley predicts individual *motes* that will eventually be the size of a grain of sand, or even a dust particle, each with self-contained sensing, computation, communication and power [56].

WSN deployments have often been applied to problems of environmental monitoring, such as detecting the stresses on a structure like a bridge [103], or the movement of people through a building [26]. The key features that make these WSNs useful are their ability to sense phenomena that is distributed across space, and to aggregate the sensor data so that it can be pieced together (usually in an offline manner or on a separate dedicated server) into a coherent summary of the phenomena. Thus many deployments can build rich models of local interactions and their surroundings without requiring external sensing or power infrastructure.

Ad-hoc mesh networks and routing protocols push the boundaries of system flexibility by avoiding reliance on environmentally installed infrastructure such as cellular networks or WiFi. Most WSNs have a user interface *to* them; this allows a user, for example, to query the current state of the nodes or to upload new firmware, yet little research effort has been invested in understanding the possibilities of multi-node sensor networks *as* user interfaces. Human-computer interaction research for single-device-per-user scenarios (e.g. mobile phones and other ubiquitous computing) is discussed in section 2.7.3 on page 43. Mapping out the human-computer interaction possibilities realized by WSN-like system that features a collection of independent interactive nodes was a motivating inspiration behind the design of Siftables (for more discussion of this theme, see section 3.2 on page 53). The next section looks at one example of sensor network techniques applied to human-computer interaction.

2.7.5 Shared Synchronous Motion: An Example Application of a WSN Technique to HCI

A problem that is central to many WSN deployments is the coordinated detection of events such as sound or motion. For networks that feature body-worn or carried devices, for instance in gait monitoring applications [82], the collective detection of inertial events can be a key feature. It can be useful for a distributed system to know when certain subsets of nodes are moving together. From gestalt psychology the principle of *common fate* [113] explains our bias to interpret things that move in a synchronous manner to be part of the same object. Common fate is a heuristic that allows us to make sense of dynamic visual phenomena. At least in part inspired by this human capability, a number of distributed systems have attempted to detect synchronous inertial events sensed concurrently by multiple devices.

The Smart Its Friends [47] system detects when two personal devices are being held and shaken together as a criteria for establishing a trusted connection between them. The question addressed is: When should two mobile devices be allowed to communicate? The assumption is that if the same person holds the two devices in hand it indicates trust between their owners. Hinckley's related work uses the detection of an impact between tablet computers as a trigger for opening a communication channel for data, or for turning the two displays into a single larger display [45]. A related project detects when the same person is wearing two devices while walking [63], and another allows multiple sensor nodes to detect if they are attached to items that are being transported by the same vehicle [73] in a distributed real-time manner.

These research projects demonstrate the utility for human-computer interaction when groups of mobile devices can detect shared synchronous motion. Given their inertial sensing, wireless communication and graphical display capabilities, Siftables would be a useful platform to explore this technique further with larger numbers of small devices, and in conjunction with graphical on-object feedback. In the same manner, I expect that other developments in the field of WSNs will contribute to the design of distributed *interactive* systems.

Chapter 3

Design Process and Interaction Techniques

Siftables began as a brainstorm in 2006 with Jeevan Kalanithi; we imagined how people might interact with digital information by using their hands to manipulate a sea of tiny physical, active, computational objects. Though we were influenced by ideas from tangible interfaces, pervasive computing and sensor networks, only later would Siftables be contextualized against the backdrop of these ideas as a hybrid that blended these themes with the flexibility of pixels that defines graphical user interfaces. The beginning however, was pure inspiration, an uninhibited "what if" speculation about a system that would permit compelling new physical interactions.

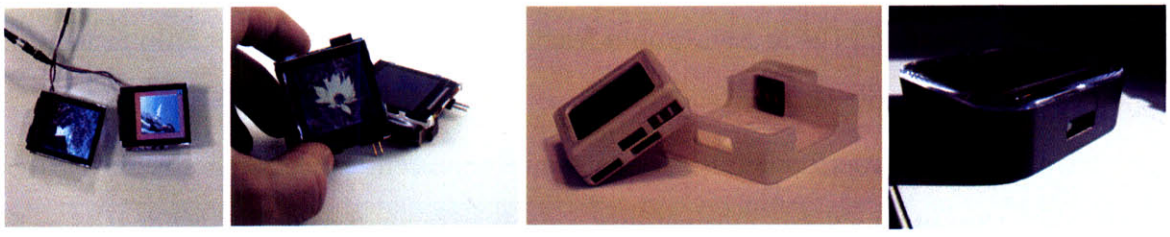


Figure 3-1: The four design iterations of the Siftables platform. From left to right: Version 1: not battery powered, used an LCD, and could sense accelerometer motion. Version 2: Bluetooth-enabled, battery powered and able to communicate with neighbors over infrared. Version 3: OLED display, 3D printed case and charging cradle. Version 4: injection-molded case, mature event-driven operating system and Python API for remote software control and application development.

The key idea motivating this brainstorm and leading to the development of Siftables was the conviction that current user interfaces are not yet utilizing our hands and bodies very well [58]. Tangible user interfaces engage our bodies and leverage our spatial understanding of physical objects, but they tend to be single-purpose systems designed for a particular task or user population. Tabletop and graphical user interfaces tend to be more general-purpose due to their use of pixels, but infrastructure requirements limit their mobility. We speculated that a system able to fuse the beneficial features of tangible and graphical user interfaces could enable a huge step forward in our ability to manipulate digital content in a physical, expressive, collaborative manner.

In the first brainstorm, Kalanithi and I posited the *Siftable Computer* composed of a collection of physical "beans", tiny battery-powered electronic devices with a screen, accelerometer and radio. Each bean would embody a digital information or media item to be arranged (i.e. a photograph, email, audio clip, etc.), and users could move the beans around by hand, sorting them in the same way they would sort any collection of physical items. Using their accelerometers and wireless communication, the system would infer groups based on shared synchronous motion. Group affiliation would be displayed visually via a colored bar or border on a portion of each small screen to provide feedback to the user. A user could shake a bean to erase its current group affiliation, or bang on the table to simultaneously erase all current group affiliations of the beans. All arrangements would be wirelessly synchronized with a nearby computer, and the system would feature a tight loop of user manipuation and on-device visual feedback.

We also posited "action beans", which would represent application-specific actions a user can take on the data. For instance a particular bean could be designated as an "email to Mom" bean. Individual beans or groups of beans could be bumped against this action bean, causing all photos to be emailed to the user's mother. Other possible action beans could be a "backup bean", "create a zip archive" bean, or "delete" bean.

The current working system reflects a number of hardware and software iterations, each version pushing closer to realizing the ideas from the original brainstorm.

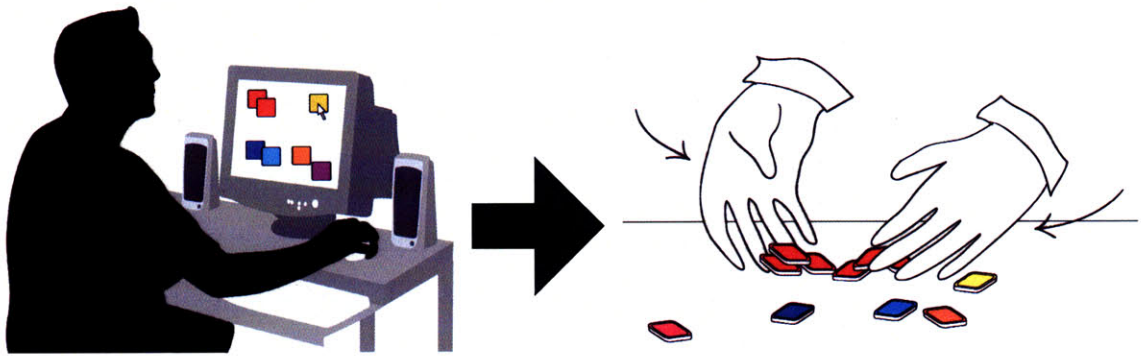


Figure 3-2: A typical desktop computer interaction scenario (left), in which a mouse is used as a virtual pointer into a graphical representation of the virtual desktop space. In an embodied media user interface (right), the manipulatives give physical embodiment to digital media content. They can both display a dynamic visual representation of the media or action that they represent, and sense the user's input, without requiring environmentally-installed sensing or display infrastructure.

3.1 Embodied Media: Hybrid Tangible-Graphical Distributed User Interfaces

This thesis introduces a new human-computer interaction concept, *embodied media.* An embodied media system physically represents collections of digital items such as files, variables, or other program constructs with a set of self-contained, interactive tokens. Embodied media tokens can present visual feedback to the user indicating their current role, and can be physically manipulated by the user as a single, coordinated interface as a means of altering the digital items represented. An embodied media system relies minimally on external sensing infrastructure, in contrast to tabletop or augmented reality systems.

Siftables is an embodied media system that combines the flexible graphical display capabilities of the GUI with the physicality of a TUI, while incorporating some capabilities of a sensor network. In contrast to tabletop TUIs that provide *handles* to a projected digital representation of data, a Siftable can display a graphical representation of the data *on its exterior* that can be viewed by the user and altered by their manipulations of the device. Since each Siftable is both a physical manipulation

interface and a display, it can tightly couple input and output to *embody* the digital media that it represents.

The concept of embodiment [28] is an important difference between Siftables and GUI or tangible tabletop interfaces. When our hand moves and clicks a mouse button, or when we manipulate a collection of graspable pucks, these interfaces are tools by which we navigate an interaction grammar. These physical manipulation interfaces have become 'physical cursors' into a digital interaction space, but they are still "handles."

Siftables attempt to offer the user a mental model where the manipulative itself is the target of the action. The goal is to enable an increase in directness (the aforementioned coupling between input and output), compared to physical handles. The design philosophy is that to the user, the Siftable *is* the media (the noun), or the Siftable *is* the action (the verb), rather than being just a tool for navigating an interaction grammar expressed on a larger display. This mental model makes the most sense for Siftables applications that do not include a large screen. In these applications the manipulatives can be seen as a medium, as the material for expressing the user's intentions in the digital realm, rather than as a reference or handle to a separate virtual representation.

3.1.1 Essential Properties of an Embodied Media System

To articulate a crisp definition of embodied media, I have identified the following properties that I consider essential to any embodied media system. These properties enable the physical, two-handed, collaborative interaction with collections of digital content items imagined in the original brainstorm. The properties outlined are characteristic of Siftables, but they are not a full description of the Siftables platform. The goal of this section is to give the reader an ability to identify what features are essential, and what features are incidental, to any specific embodied media instantiation.

- *Multiple Physical Manipulatives Used Together:* Multiple manipulatives enable a one-to-one correspondence between a manipulative and a digital content item

or control, even if this is not always the manner in which they are used. Importantly, embodied media systems permit the embodiment of *collections* of media. The requirement of being used together entails some form of real-time communication among the group of manipulatives. Applications for Siftables have used two to thirteen devices at a time.

- *On-Manipulative Feedback:* On-manipulative feedback is essential to make the mapping between a manipulative and the digital entity that it embodies legible to the user, and to establish the impression that the manipulative *embodies* the entity. This feedback could be as simple as glowing with a unique color, as flexible and reconfigurable as a color bitmap display as with Siftables, or even physical such as a shape-changing surface.

- *On-Manipulative Sensing of Other Manipulatives:* The ability for a manipulative to be aware of its interactions with other manipulatives allows the user to establish and manipulate symbolic relationships between the digital entities that they embody. Siftables senses the proximity of other devices in four directions; sensing shared synchronous motion (see section 2.7.5 on page 46) is another possibility.

- *On-Manipulative Sensing of User Input:* Each manipulative must permit direct user interaction. This interaction could be inertial (as with Siftables), or touch-based, or pressure, breath, or any number of other modalities. The key reason that on-device manipulation is important is that this directness contributes to the impression of *embodiment* of the entity by the manipulative, compared to a mouse/GUI or tangible "handles" system.

- *Reprogrammability:* Embodied media entails flexibility, and a single system of embodied media manipulatives should support many different application types. As such, it is important that the behavior and visual feedback of the manipulatives can be easily altered. Siftables features two high-level APIs, enabling applications to be developed quickly and easily.

- *Minimal External Sensing Infrastructure:* The final part of the embodied media vision is that the system should be *mobile*, avoiding strong dependencies on bulky/fixed-location sensing infrastructures such as cameras or special-purpose tabletop surfaces. This allows the user experience with an embodied media system to adhere more closely to the experience of working with non-electronic tools, which by nature do not have dependencies on their environment. Siftables can wirelessly connect to a laptop computer, or can operate in a standalone mode, either of which permits greater mobility than most interactive "surface" installations.

3.1.2 Incidental Properties of Siftables

Siftables is just one instantiation of the embodied media concept, and its particular features are not the only possible set. The following properties are characteristic of Siftables, but are not strictly essential to an embodied media user interface.

- *Un-Tethered Operation:* Battery power and wireless communication allow Siftables to operate in an un-tethered manner, without requiring a cable for communication or power. However, a system could conceivably satisfy the essential requirements of embodied media while requiring such tethering, if the tethering was to a portable device such as a laptop computer or mobile phone.

- *Square Shape:* Siftables are square tiles, which allows them to be tessellated and to communicate laterally in four directions. However, they could be a different shape, such as a triangle, pentagon, hexagon, etc. while still realizing the essential properties.

- *Generic Physical Shape:* The physicality of Siftables makes them *graspable*, and their generic square shape does not bind them to represent a particular function or media type. However, an embodied media system could feature physically differentiated manipulatives, for instance to make a durable distinction between devices that represent items versus operators, optionally with physical con-

straints on how such pieces can interact or interlock. The specific item or operator assigned to each manipulative could still be changeable at run time.

- *Each Manipulative Has Identical Capabilities:* Siftables each have the same capabilities to sense each other, to sense their own motion, and to communicate wirelessly, store data, and display graphics. The design philosophy for Siftables is that each is interchangeable with any other. However, related to the discussion about generic physical shape, an embodied media system could feature some manipulatives with different capabilities than others, or even a system that featured a combination of embodied media manipulatives and non-embodied-media, but still interactive, items, working together.

Technological capabilities are now available to explore other embodied media instantiations, including sensor networks, which are discussed in the next section.

3.2 A Sensor Network User Interface (SNUI)

The tradeoff between tangibility and flexibility discussed in chapter 2 on page 25 led us to consider how we could expand the design space of interactive systems to enable combine directness and flexibility in a user's interaction with digital information or media. One element of how Siftables combines these design goals is its incorporation of certain sensor network characteristics.

As discussed in [2], most sensor network systems today implement monitoring scenarios [72] [103] [26] [67]. These deployments typically feature a user interface *to the network*, for instance visualization and management software on a PC that can query and summarize data from the nodes, or that can update their behavior. Lifton et al. first coined the acronym SNUI (Sensor Network User Interface) [65] to describe the management software that they developed for such a deployment. Other work in "participatory sensing" [14] has posited networks of people, each with a mobile device such as a smart-phone, as a kind of interactive sensor network.

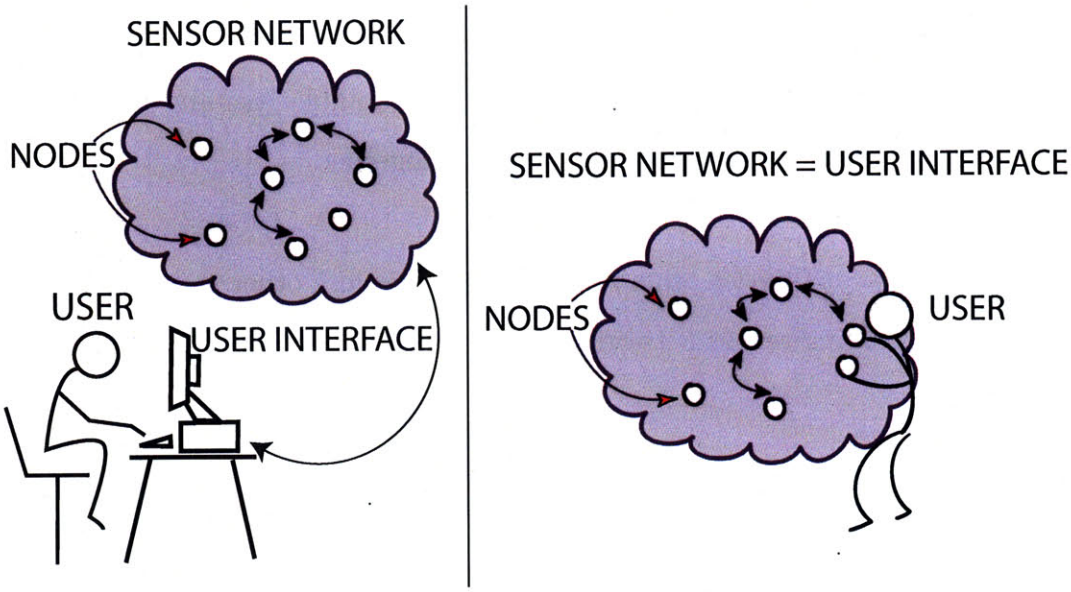


Figure 3-3: In a Sensor Network User Interface (SNUI) as defined by Merrill, Kalanithi and Maes [79], the sensor network *is* the user interface. On the left is a SNUI as described by Lifton et al. [65], in which the nodes are "out there" in the world, accessed by a user interface program on a standard computer. On the right a SNUI like Siftables, a user interface where the nodes themselves are interactive and are directly manipulated by the user.

With Kalanithi and Maes I defined a new sense of SNUI [79]. Rather than describing an interface *to* a sensor network, or a group of individuals with mobile devices as the nodes of an interactive sensor network, we proposed the use of nodes of a wireless sensor network *as* a user interface that can be directly manipulated by an individual or a group (co-located or not). Siftables is a SNUI in this sense of the term. Each node has sensing, wireless communication, *and user-directed output capabilities*, specifically graphical representations.

3.3 Prototyping

Prototyping is an important part of many design processes. I must clarify the distinction between the capabilities of Siftables as currently implemented and the possibilities that the implemented system allow us to explore. The underlying assumption of my research is articulated by Bradley Rhodes as he summarizes the attitude towards prototyping generally found at the MIT Media Lab. It comes from an explanation of why the Media Lab is situated within the School of Architecture at MIT.

> *...it's not just the research topics that draw from architecture, it's also the methodology. Engineers draw designs. Scientists run experiments. Architects, we build models. At the Media Lab we called them "demos" and they tended to look more like computers and electronics than miniature foam-core houses, but they were based on the same basic idea that you can't truly understand something new until you build one and play with it a while. - Bradley Rhodes*

By building a functioning model that can be tested and shared with other people, we can better understand the possibilities and limitations of a new technology or design idea. Therefore Siftables is a high-resolution prototype. In order to understand the implications of an interface that comprises a collection of small physical manipulatives that can be easily handled *en-masse*, I could not have used existing platforms such as mobile phones, because their form factor is too large and certain

capabilities (i.e. neighbor sensing) are absent. I wanted to be able to display dynamic information on the devices, so Siftables were built to have functional color screens. Furthermore, the vision was that they be untethered, so Siftables are battery-powered and have wireless communication.

Despite the system's relatively high fidelity, some details that were part of the original idea but that were not essential to understand the interaction possibilities were omitted for pragmatic reasons. For instance, mesh networking. To draw on many of the exciting advances in sensor network research, future embodied media manipulatives should be able to communicate with any other nearby manipulative directly. However, I decided that this capability could be simulated with a star network topology for data communication in which each Siftable has a wireless connection to the same host computer wherein resides the controlling program. Furthermore, a real-world Siftables deployment may not require the nearby computer to run the program. It could either be distributed entirely among the devices, or a single Siftable could be elected dynamically to control the others. Again, this architecture was not necessary in order to answer the interesting research questions, so it was not implemented.

After the initial brainstorm described at the outset of this chapter, we prototyped the Siftables platform at various levels of realism in order to better understand the implications of different possibilities for physical form factor and interaction. The next few sections discuss the prototypes we created and what we learned from each.

3.3.1 Choosing Features

Determining the set a capabilities that a Siftable would require was not a straightforward process. From the kernel of the original idea of interactive "beans" that could show graphical feedback and be manipulated by hand it was clear that Siftables would require a display and to be able to sense how they were being handled. Wireless communication would be required to immediately sync their state with an on-computer representation of the embodied data. However, a description at this level is grossly under-specified. A display could be a bitmap screen capable of millions of colors, or it could be black and white, or a segmented liquid crystal display (LCD), a series of

LEDs of different colors, or even an actuated "skin" that could deform dynamically. Likewise, the ability for the devices to sense how they were being handled might imply inertial sensing with an accelerometer or gyros (or both), but it could also include sensitivity to touch, proximity, eye gaze, sound, breath, directional heading, absolute position relative to a workspace, or any number of other quantities.

The feature set that was implemented was chosen after several iterative brainstorms regarding the core interaction ideas and applications that might be built. Recognizing that Siftables could be a platform supporting a number of different applications, we decided to choose a set of features that enabled a reasonably large flexibility of use contexts. The physical size of electronic components placed a lower bound on the size and shape of the device, pushing it a bit larger than we had originally imagined. Finally, accessibility of components and ease of integration was a factor: components that could be obtained easily and utilized without undue difficulty were selected given the time-sensitive nature of getting the platform to a usable state. See figure 3-4 on the next page for an overview of the space of feature possibilities that were considered and those that were selected.

3.3.2 Paper, Wood and Acrylic Prototypes

The exact size and shape of the Siftable manipulatives was left unspecified in the initial brainstorm. The basic interaction idea called for devices no larger than a small mobile phone, since the user manipulates a collection of devices during a typical interaction and might each need to hold several of them in one hand at times. This observation provided a rough upper bound on the size of the individual devices slightly smaller than a typical mobile phone. At the other end, each Siftable would need to be easily manipulated by a wide range of users of different ages and degrees of manual ability, so they could not be too small or difficult to grasp. Picking a Siftable up from a flat surface should be easy for most users, implying that the devices would need to be at least as wide as a small coin, and probably taller, given the difficulty that coins can present when they are lying flush against a smooth surface.

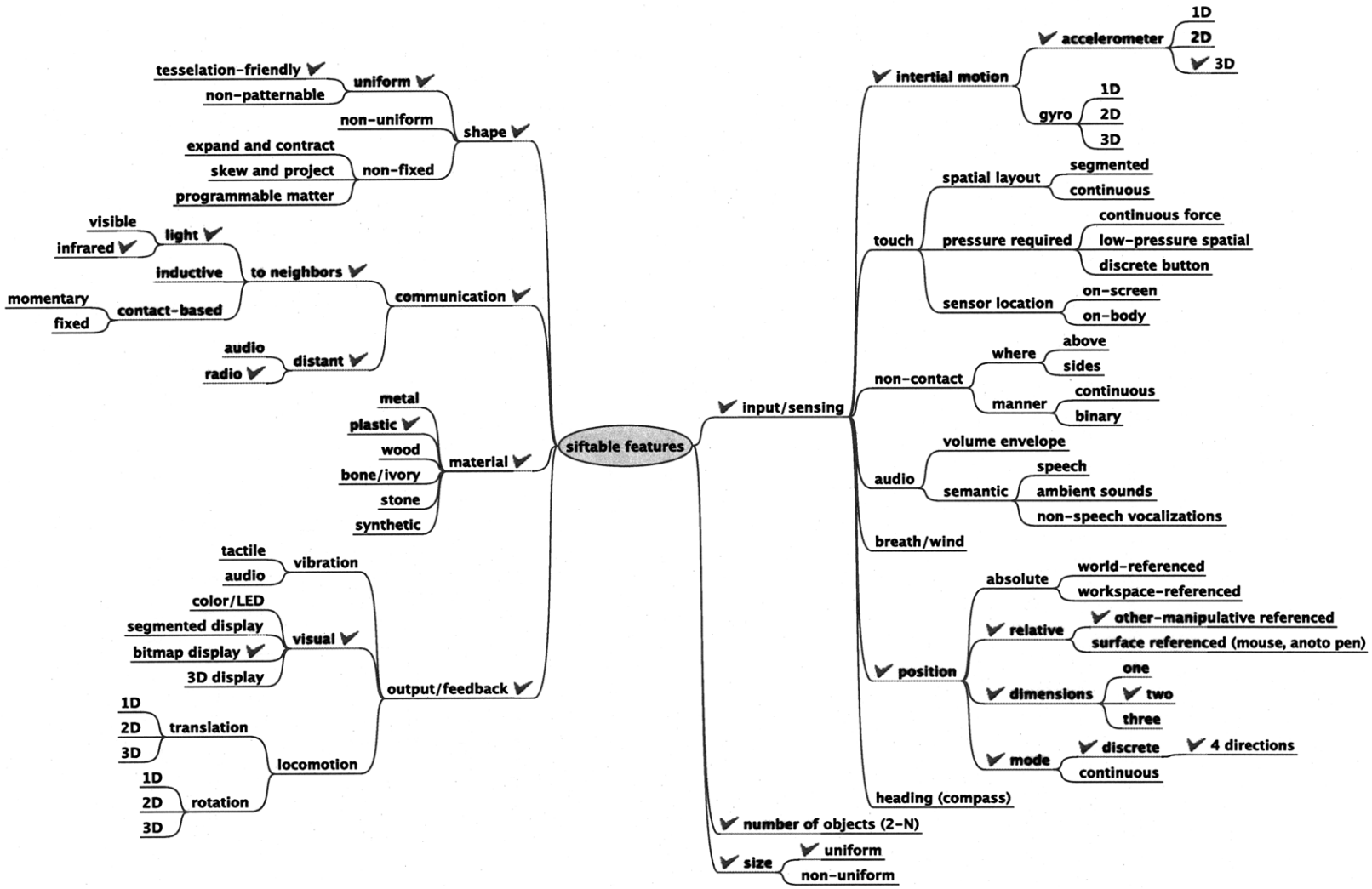


Figure 3-4: Design space of Siftables as an embodied media implementation. The check marks show features of Siftables, and the entire figure shows an expanded design space of options that would be compatible with embodied media. The non-checked items were not explored in the current instantiation.

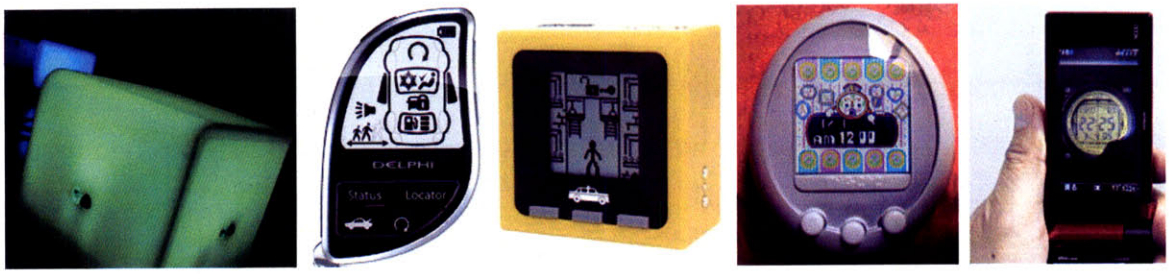


Figure 3-5: A subset of different display possibilities that were considered for Siftables. From left to right, they are single-color (essentially 1 pixel of information), black and white segmented, black and white bitmap, color bitmap, and three-dimensional. Depicted products are (from left to right) AudioCubes [95], Delphi key fob [21], Cube World [33], Tamagotchi [7], and the Hitachi Wooo H001 mobile phone [24].

Figure 3-6: Paper, wood and acrylic mockups. The given size and thickness were selected after considering both the visibility of thumbnail images and how many Siftables could comfortably be used on a tabletop at a time.

To explore the size and shape that Siftables should be, we built non-functional prototypes from wood, acrylic and paper. Beginning with differently-sized paper cutouts of images, we quickly observed that the Siftable's screen should be large enough to comfortably discern the content of an image thumbnail. Given that many imagined uses involved tabletop interaction, easy viewing would need to be possible even at a distance of up to a meter away. After examining images of different sizes, we determined that the size of the screen should be at least 1 inch on a side.

We also realized that the impression of a "sea of small active manipulatives" quickly diminished with larger cutout images. Our desire to retain this design idea suggested an upper bound on the size of each manipulative. Pictures the size of mobile phones or larger began to take up too much desk space in aggregate. Furthermore, larger images invited deeper inspection, causing attention to focus on individual items rather than the collection. We also noted during this phase that a perfectly square device would be advantageous, since it permitted regular tiling and other 2-dimentional topologies to be created. Thus we narrowed our focus to square screens at most 2 inches on a side.

Wood and acrylic cases were made to hold images that were our favorite size after the paper prototyping phase. After a bit of experimentation with the thickness of the acrylic, we made these cases to be roughly 1.5 x 1.5 inches square, and 0.3 inches in height. See effig:sift-nonelectronic-mockups for more details.

The takeaway from our paper, wood and acrylic prototypes was that the primary tension in screen size came from our desire to see the images easily (the larger the better), while being able to handle multiple devices easily at the same time (the smaller the better, to a certain point). Also, we realized that the size of the image impacted the amount of visual inspection that a Siftable would invite. In a sense analogous to hand-sketching in which the level of detail must be finely tuned to avoid non-important features [16], we observed that the level of visual detail shown on a Siftable should be considered carefully when displaying photographs or other images.

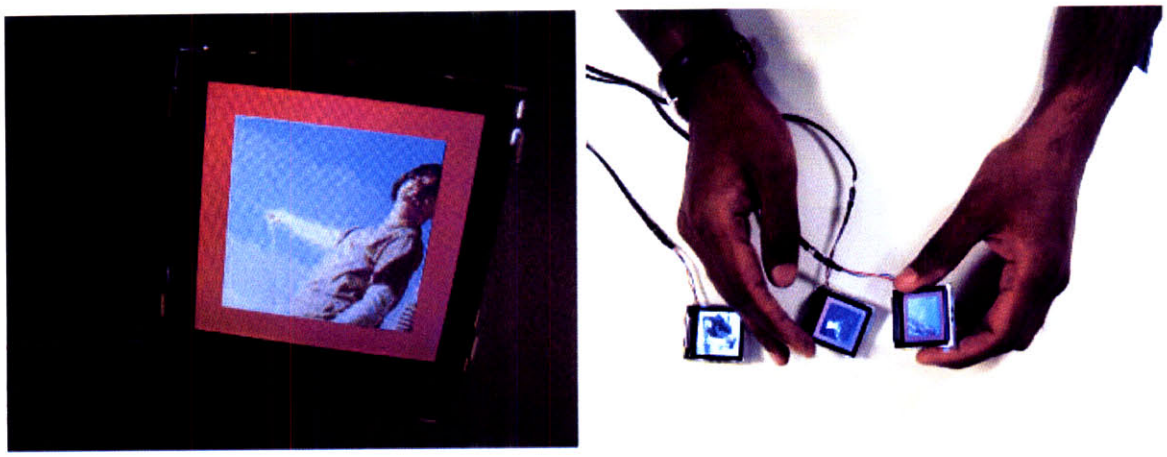


Figure 3-7: The first electronic prototype. Early feedback was that the interaction ideas were compelling, but the proposed photograph-sorting application was not, since many people felt that their current GUI-based tools were sufficient.

3.3.3 First Electronic Prototype

The first electronic prototype was built between August and October in 2006. It featured a 128x128-pixel color LCD and 3-axis accelerometer. It was used to implement a simple interaction sketch to explore photograph grouping. Images stored in the program memory of the microcontroller could be loaded into the display, and when the Siftable was shaken, a border would appear around the edge of the screen.

A problem with the circuit board layout prevented us from implementing neighbor detection in this first prototype. This capability was explored in the next prototype.

I built four devices of this version, and visitors to our laboratory were exposed to the following interaction sketch: They were told that Siftables was an interface that could be used to manipulate a personal collection of digital photographs or other digital media, and that in a true deployment, thumbnails of the images would be transmitted wirelessly to the devices. Pushing a collection of Siftables together into a pile would result in the original photographs being put into a folder together on the computer. I would present two Siftables showing different images on their screens to the visitor, and bump the devices together. The impact would cause a border to appear around the edge of each photograph. I would explain that at this point the two photographs would either be put into a folder together or labeled with the same

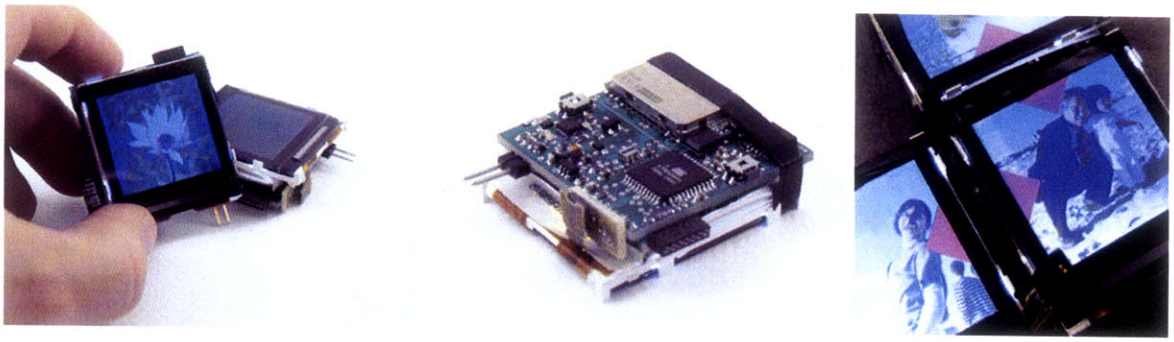


Figure 3-8: The second electronic prototype, with working infrared communication, flash memory and Bluetooth radio, allowed me to experiment with neighbor detection and wireless image uploading from the computer. In the application depicted in the rightmost image, a detected neighbor triggers drawing of a red triangle on the affected side.

tag, so that they would be found together when a person searched their photograph collection at a later time, Turning the Siftables upside-down briefly would clear the border.

I learned two lessons from this sketch. People found the interaction possibilities of the *platform* compelling, but they were not particularly interested in the proposed application of photograph sorting. Many felt that their current GUI-based tools were sufficient, or that they would rather have ways to enter textual annotations for photographs more easily. They also wondered how the system could accommodate the thousands of photographs in their collections. While it is possible that a more realistic photo-sorting application would have been more compelling, my intuition was to look elsewhere for applications that uniquely benefited from the possibilities of embodied media.

3.3.4 Second Electronic Prototype

The second electronic prototype was finished in early February 2007. It improved technically over the earlier prototype, with infrared communication in all four directions, Bluetooth radio, flash memory and a rechargeable battery.

The interaction sketches I implemented with this prototype explored peer-to-peer communication and detecting impacts to the table surface. Images could be loaded

Figure 3-9: The "Attentionables" application was programmed for the Second Electronic Prototype by Evan Broder. Siftables detect each other using infrared when placed side-by-side, and each face looks towards the other. Attentionables adapts an art piece by Zuckerman and Sadi, and it was the first real application that extended Siftables' behavior beyond the level of a simple application sketch.

into the flash memory of a Siftable wirelessly over Bluetooth, and the presence of a neighbor could be detected using infrared communication.

To explore user interface implications of peer-to-peer communication, the Siftables were programmed to show visual feedback in the form of a red triangle at the edge of the screen when the presence of a neighbor was detected on the given side. This neighbor-detection demo illustrated the possibility that Siftables could be arranged on a table in arbitrary two-dimensional topologies to build interconnected structures such as flowcharts, or they could just as easily pass information and media such as business cards or photographs from person to person.

To explore inertial interaction with the environment, several Siftables were programmed to sense sharp perturbations in the Z (up-down) direction. On detection of perturbation, each Siftable would toggle its display between showing a photograph and showing a blank screen. This allowed us to prototype a solution for the problem of having more digital media items than the number of available Siftables to display them; the user could pound or slap the table surface to swap in the next "batch" of

media. The gesture would cause each Siftable to change its assignment and update its graphical display to show the next piece of available media.

The Second Electronic Prototype was the first version to have all major features working. As a result, it began to interest other researchers who saw it as a platform that they might use to implement user interface ideas. Ivan Poupyrev (Sony Computer Science Lab) suggested that DataTiles [102] could have been implemented using Siftables. Zigelbaum (Tufts, MIT Media Lab) expressed interest in creating a more compelling version of his Tangible Video Editor [121] using Siftables. In the months that followed, more colleagues inquired about when they could work with Siftables to implement human-computer interaction research ideas.

Feedback from other researchers began to suggest that Siftables would be a useful platform to implement a wide range of application ideas. During the summer of 2007 an undergraduate named Evan Broder became the first application developer, programming a Siftable-based version of the art piece "Spotlight" [123] wherein an animated face on each Siftable would look towards neighboring Siftables when they were placed side-by-side. In addition to neighbor-detection, Broder's "Attentionables" application exercised Siftables' ability to animate through sequences of images stored in the flash memory.

Siftables needed an API for communication between software on a computer and a Siftable. Broder developed a small command set that could be typed into a serial-over-Bluetooth terminal on the computer when connected to a Siftable for his own use. Although this early command set was subsequently discarded, it was an important first step towards what is now a full API.

3.3.5 Current Siftables Design

The technical specifications of the current Siftables design are described in chapter 5 on page 113. All of the intended features are now usable: graphical display, flash memory, inertial sensing, neighbor identity+orientation detection, rechargeable battery and Bluetooth wireless communication. Furthermore, the exterior case is now extremely robust. Whereas a proto-generation of the current version had relatively

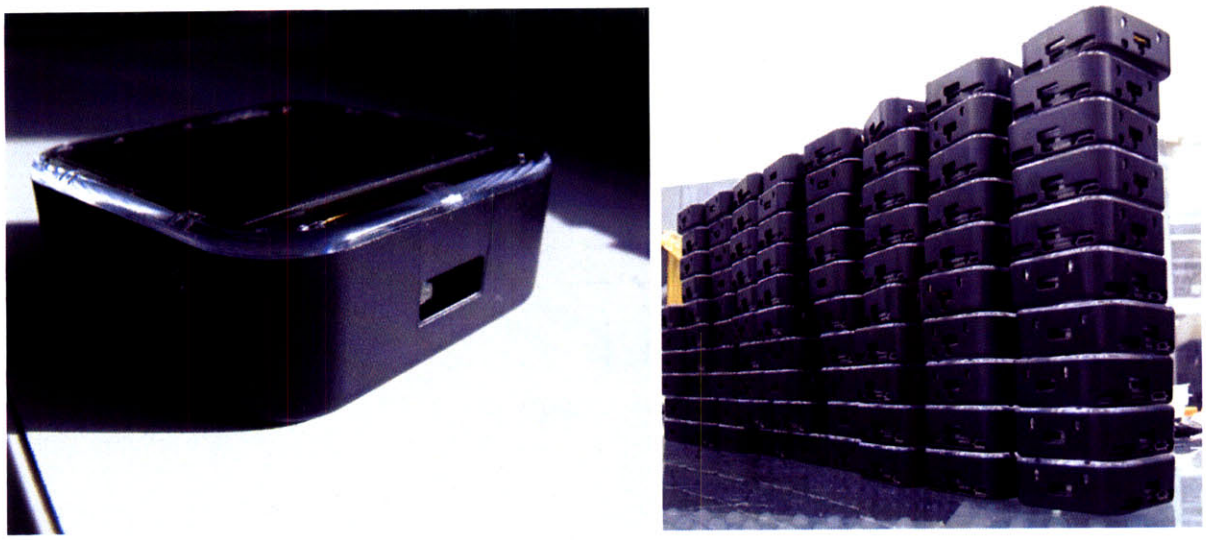


Figure 3-10: The current Siftable device. I built 140 units in the spring of 2008, some to support collaborations with other researchers and some for my own experiments.

fragile 3D printed cases, the current design has an injection-molded plastic exterior.

Possessing a large quantity of manufactured prototype units has enabled more developers to get involved in creating applications for Siftables. In the summer of 2008 four undergraduates and two graduate students at MIT worked with Siftables. Additionally, nearly 100 units have been sent out to researchers in industry and elsewhere in academia. My ability to share the platform has resulted in an acceleration of progress on the firmware and Python API. More details about these collaborations and new applications can be found in chapter 4 on page 85.

3.4 Designing a Gestural Language

Designing a gestural language that enabled novel possibilities for interaction with digital content was a motivating factor behind the development of Siftables. We began with the idea of a single gesture: grouping (see figure 3-11 on the following page). Inspired by research on shared synchronous motion [45] [47] [63], we imagined that collections of content, each item represented by a Siftable, could be grouped by pushing the given Siftables together into a pile. The devices, noticing that they were being jostled, would each report this motion to a server. The server would notice

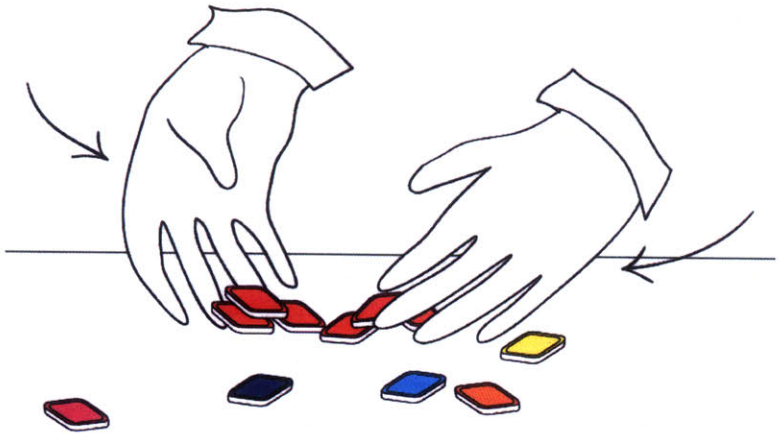


Figure 3-11: An early design idea for a gestural language element. Inspired by related work on user interface uses of shared, synchronous motion, we imagined that digital content could be grouped by pushing a collection of Siftables representing content items into a pile.

which devices were being moved at the same time and would group the content by placing it into a shared directory, or assigning a common tag to all elements. Visual feedback commands would be transmitted to each affected Siftable for display.

As the platform was developed, we identified a more complete set of gestural language actions that could be implemented with Siftables. Some of these actions are shared with other tangible-tabletop style systems, while others are not possible in those systems. The actions that are unique to Siftables take advantage of their ability to sense motion and/or to show graphics on the manipulatives themselves, capabilities that other multi-manipulative systems do not typically have.

3.4.1 Actions in the Gestural Language

This section will outline the actions (basic interactive primitives) that are possible in the interaction language of Siftables. I do not present a complete enumeration, since many custom actions can be created by leveraging continuous gesture. However, these primitives capture the current actions that applications have been designed around, as well as a few others that are not yet implemented.

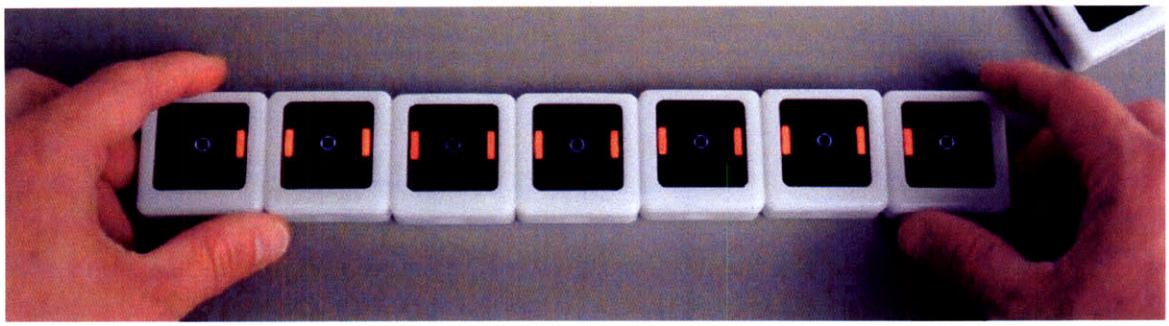


Figure 3-12: Topology: row/column. Siftables can be arranged into linear sequences.

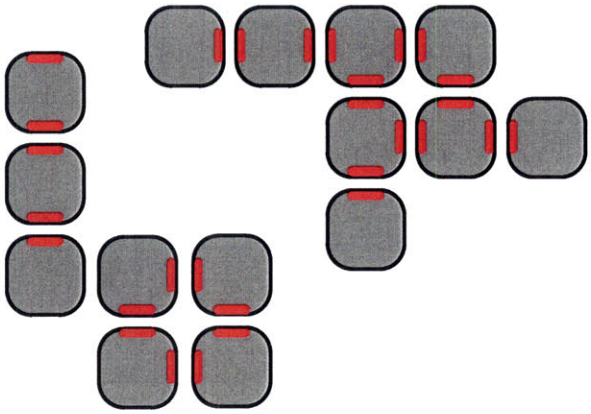


Figure 3-13: Topology: arbitrary 2D pattern. Siftables can sense each other in any contiguous two-dimensional pattern, provided that the devices are at right angles with respect to each other and close enough to communicate by infrared. In this image, the red rectangles indicate the awareness that each Siftable in the shown topology would have of its neighbor in the direction of the given edge.

Topology: Row/Column

Siftables can be arranged into linear topologies, sensing their adjacency to neighbors as an input (see figure 3-12). This language element can be used in applications that involve sequencing of content items, such as letters, numbers, video clips, or Boolean variables, for instance model parameters or database query elements.

Topology: Arbitrary 2D Pattern

Since they are able to sense their neighbors in four directions, Siftables can be arranged into arbitrary two-dimensional topologies (see figure 3-13). This can be used

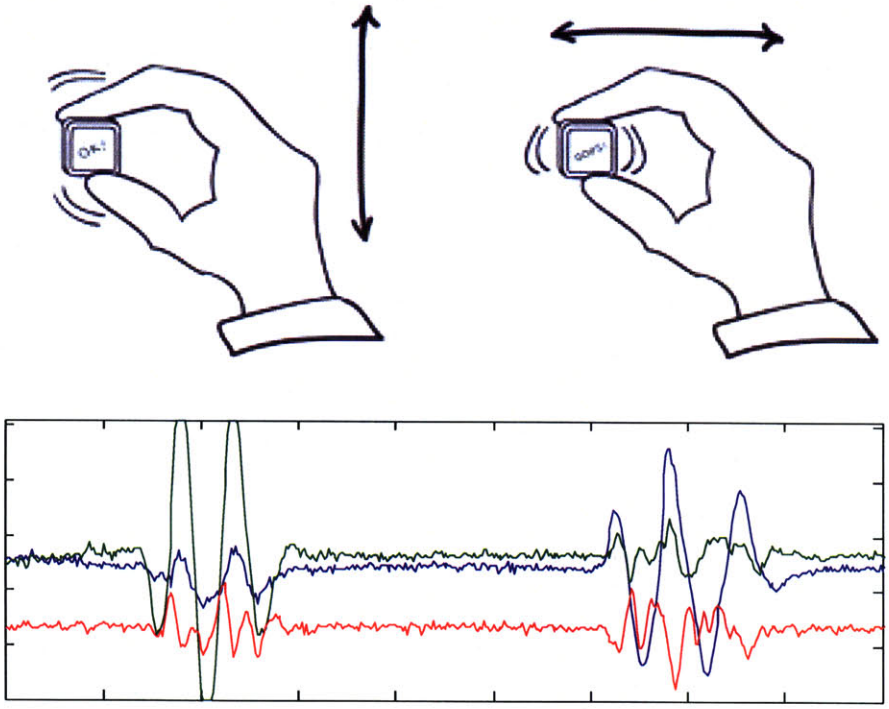


Figure 3-14: Gesture: shake. At the bottom you can see data collected from a Siftable being shaken up-down (bottom left), versus side-to-side (bottom right).

to implement applications that involve spatial arrangement of content items.

The limitations of the 2-dimensional arrangement possibilities are that Siftables must be close enough to detect each other, and they must be arranged at right angles to each other in a grid pattern.

Gesture: Shake

Siftables can detect the user's shaking gesture on each axis separately using a running windowed calculation that sums the absolute deviation from the mean of the last 32 samples. A mapping of this gesture could be an on-screen confirmation that asks for a "yes" or "no" shake. Another possibility is that a shake may clear the association between the shaken Siftable and a digital content item (see figure 3-14).

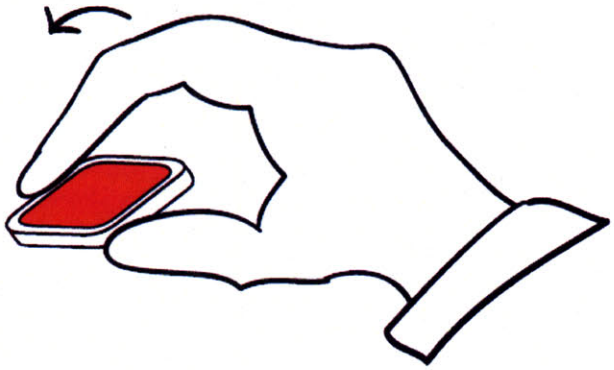


Figure 3-15: Gesture: tilt.

Gesture: Tilt

Since acceleration and gravity both exert a force, a Siftable can compute its tilt with respect to the direction of gravity. It can generate an event upon detecting that this tilt exceeds a threshold (see figure 3-15).

Gesture: Upside-Down

An extension of tilt-detection, the property of being upside-down can also be detected by a Siftable. The gesture of turning a Siftable upside-down then back has been used in one application sketch to advance the content being displayed on-screen.

Gesture: Nudge

A small shove of the Siftable along the X or Y axis in the horizontal plane can be detected. This gesture was prototyped by undergraduate Laura Harris, and could be used to provide discrete directional input to an application. For instance, in a map exploration application a nudge could trigger the current view shown on the Siftable's screen to traverse to the adjacent tile of a map.

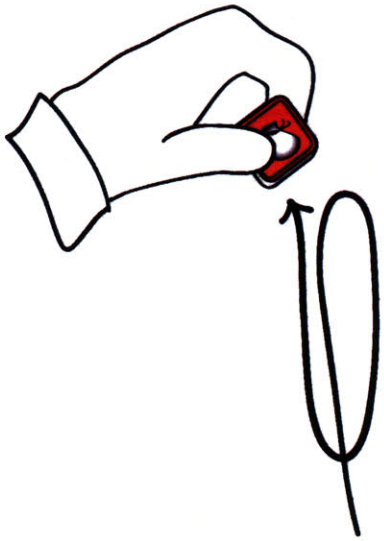


Figure 3-16: Gesture: arbitrary motion.

Gesture: Arbitrary Motion (application-specific)

Using its three-axis accelerometer a Siftable can sense arbitrary inertial motion. The ability for a manipulative or controller to recognize arbitrary user-created or developer-created motions has been explored both in research [80] [78] [9] and commercial contexts [51] so this possibility was not implemented for Siftables (see figure 3-16).

Environment: Lean

Lean is a variation on *Tilt*. Being sensitive to its angle of tilt means that a Siftable can be leaned up against a physical object such as a wedge or other object. In a manner akin to Patten's mechanical constraints, [92] this allows physical objects in the environment to be used in conjunction with Siftables, for instance as a way to quickly set the value of a parameter in a simulation or musical performance application (see figure 3-17 on the facing page).

Environment: Thump

The table itself can be used as an input. If their threshold for shake detection is set to high sensitivity, Siftables can detect impacts to the surface, for instance to clear

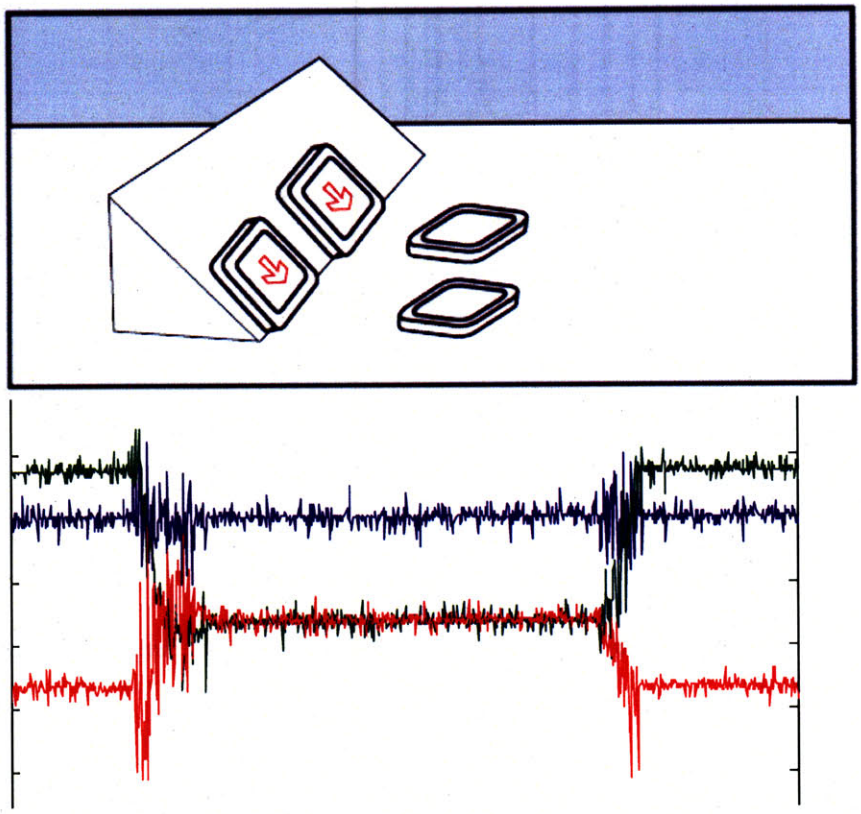


Figure 3-17: Environment: lean (top). Siftables can interact with physical objects in the environment, for instance to quickly set a parameter that is keyed to tilt. The bottom image shows the accelerometer data that is produced by leaning a Siftable against a wedge. Each signal is from one axis of the accelerometer. From top to bottom (left-hand edge of plot) the signals are: Z, X, Y. As the Siftable is tilted up from a flat resting position to an angled orientation, the component of gravity sensed by the Z axis decreases, while the component of gravity sensed by the Y axis increases.

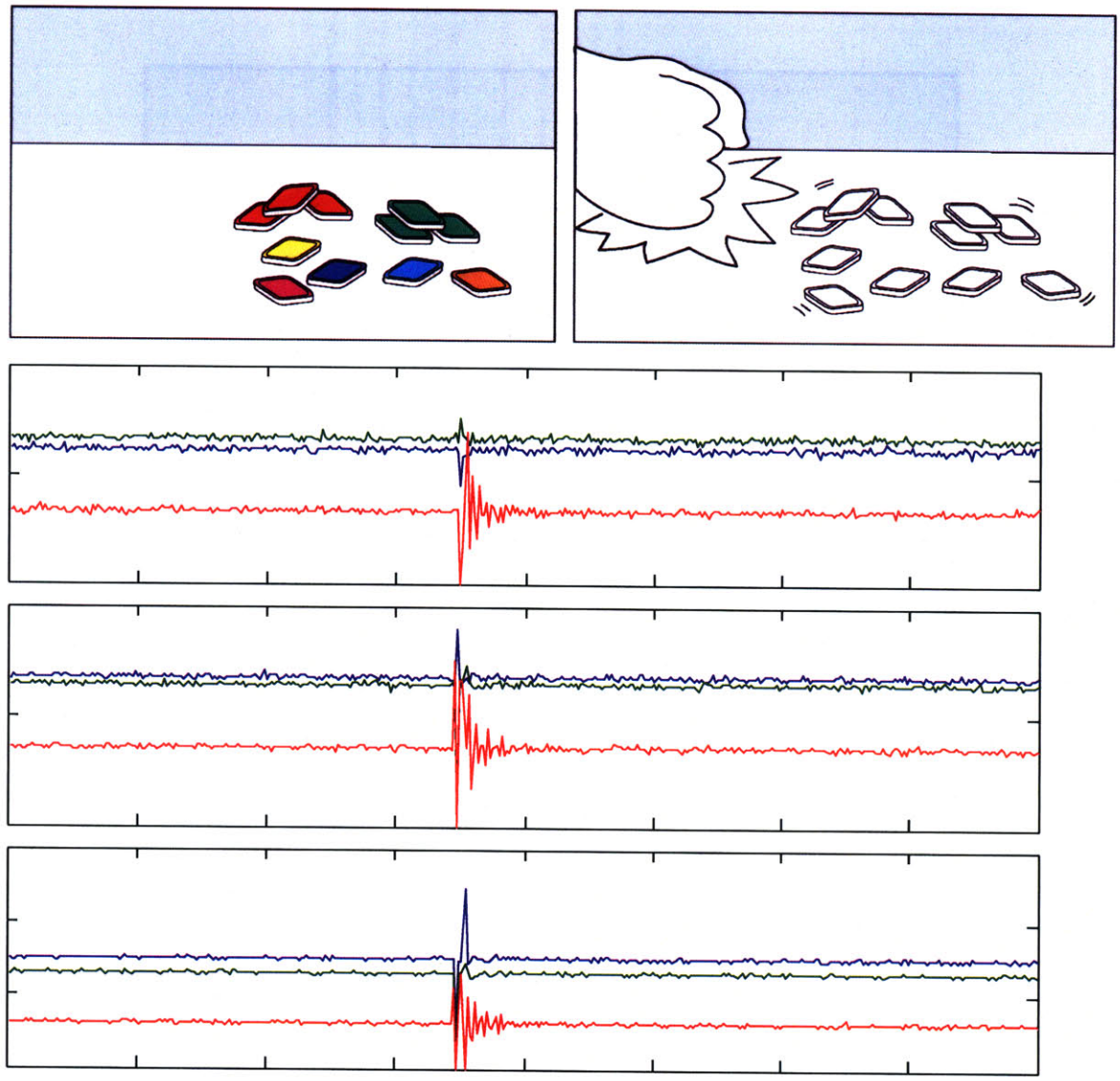


Figure 3-18: Thump gesture: A user thumps their fist on the table, bumping all Siftables at once to clear their current data associations (top). Inertial data from three Siftables manipulated in this way is shown in the graphs (bottom).

the current associations between Siftables and media items. In the future, custom signal-processing code could be written that would separate this gesture even more cleanly from other manipulations (see figure 3-18 on the preceding page).

3.4.2 Compound Gestures

Just as words in a spoken language are strung together to form sentences, actions in the Siftables interaction language can be detected in a sequence. An example is the interaction sketch created for pouring color. The user interaction consists of placing a Siftable showing a paint-bucket next to another Siftable showing an image. Then, as the paint-bucket Siftable is tilted on its edge towards the image Siftable, the image Siftable begins to accumulate the hue shown by the paint-bucket Siftable. When the paint-bucket Siftable is returned to a flat position, the accumulation of color ends.

The sequence of atomic gestures in the aforementioned example is the following: First, two neighbor proximity events are generated (one by each Siftable upon noticing the other). Then, a tilt event is generated by the paint-bucket Siftable, indicating that it is on edge. Until the second tilt event arrives to indicate that the paint-bucket Siftable has been laid flat, color is added at a fixed interval to the image on the image Siftable. The rate of the addition of color may depend on the degree of tilt of the paint-bucket Siftable.

The combinatorics of the existing gestures thus creates a large set of possibilities for the implementation of compound gestures.

3.4.3 Mapping the Gestural Language to Applications

Mapping the possibilities of the gestural language to the needs of particular applications is an application-specific activity, and a general-purpose recipe cannot be provided. However, in this section I will discuss some useful and re-usable mappings that have been developed in service of the various applications that I and others have created. Note that more than one of these mappings may be used in conjunction to create a maximally usable application.

Adjacency-Based Entity-Connection

Placing and leaving two Siftables adjacent to each other, close enough to be mutually detected as neighbors, can be a way to logically or spatially arrange their associated information entities. An example is the Scraboggle application, in which Siftables representing letters are placed adjacent to each other to form complete words. Visual feedback indicating the detection of neighbors may be helpful, but is not required.

Adjacency-Based Property-Toggle

Placing two Siftables adjacent to each other momentarily can be a way to toggle a property on one or both of the Siftables. An example of this mapping is the node edge application, in which Siftables can be "bumped" against each other to create an edge between them, and "bumped" again to remove the same edge. Visual feedback indicating the state of the property is recommended.

Entity Dump

When two Siftables are next to each other, one can be tilted toward the other as a means to transfer an information entity from the tilted Siftable to the flat Siftable. An example of this mapping is the maze exploration game, in which the player's character is transferred from one maze location to an adjacent location by "dumping". Visual feedback showing the entity transfer to the flat Siftable is recommended.

Attribute Pour

When two Siftables are next to each other, one can be tilted up toward the other as a means to transfer a continuous amount of a property from the tilted Siftable to the flat Siftable. The amount increases as long as the tilted Siftable is held in a tilted state. An example of this mapping is the color pouring application, in which three colors can be mixed in a "container" Siftable by pouring. The rate of pour may be fixed, or may be linked to the degree of tilt. Visual feedback indicating the continually-updating amount of the property that has been transferred is recommended.

Shake to Signal

Shaking a Siftable can be used as a way to provide a discrete (i.e. button-like) input, without a button. An example of this mapping is the node edge application, in which tilt-to-adjust mode can be entered and subsequently exited by shaking the Siftable. In that example, the steady-state value is measured from 0.5 seconds *before* the second shake, to capture the tilt value before it is perturbed by the shaking gesture. See section 7.1.2 on page 166 for a longer discussion of this example.

Tilt to Adjust (direct)

The instantaneous tilt of a Siftable can be linked to a continuously-valued variable in an application, allowing the user to change the value of the variable by tilting the Siftable. An example of this mapping is the effect adjustment in the music sequencer application, used to manipulate Lead, Bass, and Drum. The advantage of the direct approach is an expressivity that derives from the value changing seemingly instantly as the user adjusts the tilt of the Siftable. The disadvantage is that in order to fix the value at a given point, the user must either leave the Siftable tilted at the desired angle, or have some way to start and stop the tilt-based adjustment (a button could have solved this problem, also see "Shake to Signal"). Auditory or visual feedback is recommended, but not required. See section 4.12.2 on page 105 and section 7.1.2 on page 166 for longer discussions of using tilt to adjust a value.

Tilt to Adjust (temporal)

The state of a Siftable being tilted at an angle that falls outside a "deadband" around flat can be used to increment or decrement the value of a variable for as long as the Siftable is tilted at an angle falling outside the deadband. An example of this mapping is the effect adjustment in the image manipulation application. The advantage of the temporal approach is that the user can adjust the value to a given point, then leave the Siftable flat on the table and the value will remain at the desired point. The disadvantage of the temporal approach is limited expressivity, since it takes time to

reach a given value. The rate of adjustment may either be fixed, or may be linked to the degree of tilt. See section 4.12.2 on page 105 and section 7.1.2 on page 166 for longer discussions of using tilt to adjust a value.

Discrete Tilt to Select (one-axis)

Tilting a Siftable away from flat and back can be used as a means to provide a single discrete directional input to an application for linear traversal of a list of items or for selection from two options. An example of this mapping is the color menu in the tilt-based color etch-a-sketch drawing application, in which tilting up or down allows the user to navigate the menu, and shaking the Siftable activates the current selection. Visual feedback showing the result of the selection or navigation, and keeping the list short are recommended.

Discrete Tilt to Select (two-axis)

Tilting a Siftable away from flat and back can be used as a means to provide a single discrete directional input to an application for two-dimensional traversal, or for selection of one of four options. One example of this mapping are the Simon game, where tilting the Siftable towards a given side is a means to select one of four color regions. Another example is Telestory, in which tilting the Siftable toward one of its corners is a means to select the object in the given quadrant. Visual feedback showing the result of the selection or navigation is recommended.

Upend to Switch

Turning a Siftable upside-down and back can provide a single discrete input to an application. An example of this mapping is an application sketch where the image displayed on the Siftable's screen changes each time the Siftable is upended. Visual feedback showing the result of the upending is recommended.

Thump to Advance

Thumping the tabletop upon which one or more Siftables is resting can be a means to provide a single discrete parallel input to all Siftables simultaneously. An example of this mapping is an application sketch where the images displayed on the Siftables' screens change each time the tabletop is thumped. Other mappings could include advancing the content displayed on each Siftable by one, an "undo" gesture, or a "clear" gesture that would remove any existing Siftable-to-content mapping. Sensing a tabletop thump requires either fine-tuning the Siftables' shake-detection threshold to be highly-sensitive while avoiding spurious detections, or developing a custom signal-processing routine to detect such impacts. Visual feedback showing the result of the thumping is recommended.

Cluster to Group

Pushing a group of Siftables together into a cluster can be a means to logically group their content. One way to accomplish this is to require all grouped Siftables to be positioned in such a way that they recognize each other as neighbors, and the contiguous topology of Siftables with neighbor relationships would be considered a group. This method places constraints on the interaction, requiring the user to carefully align the Siftables. Another way would be to detect a synchronous shake or impact on the grouped Siftables, then to infer which devices were part of the grouping by the temporal alignment of such shakes or impacts. I did not implement this mapping, but the theoretical background for it was discussed in section 2.7.5 on page 46.

Nudge to Traverse

Nudging a Siftable along its X or Y axis can be a means to provide a single discrete directional input to an application for two-dimensional traversal, or for selection from one of four options. For instance, an underlying image or map could be traversed by a single Siftable, moving one grid location at a time when the user nudges the Siftable

in the X or Y directions. I did not utilize this mapping in application development, but the ability to sense the gesture was prototyped by Laura Harris in her exploration of position estimation from integrated acceleration (see section 4.12.3 on page 108 for details).

Motion Shape to Anything

Moving a Siftable in an arbitrary three-dimensional shape can provide continuous or discrete input to an application. Inertial data such as the windowed sum of the absolute deviation from the mean or the raw accelerometer data can be linked to a program variable such as a pitch or a modulation frequency in a music application, or could be used as an input to an accumulative process such as "bowing" a physical model of a tensioned string. To utilize motion data in a symbolic manner, complete gesture shapes could be recognized as a means to trigger application events. I did not implement recognition of gesture shapes in the Siftables API, but I explored similar user-created "gestural bookmarks" in my master's thesis [78] [80].

3.4.4 Limitations of the Current Design

Although the possibilities of the gestural language are many, certain limitations exist when compared to other interactive systems. This section discusses such limitations, breaking them down into three rough categories: topology limitations, gestural limitations, and architecture limitations.

Topology limitations

Siftables can sense the presence of neighboring Siftables in the four directions facing their sides, but there is currently no provision for sensing a neighbor above or below. Such a capability would allow for *stacking* Siftables to be a meaningful gesture, for instance to group data items. Stacking could optionally impose an ordering on the grouped tokens, but this would not be necessary. Inertial data from Siftables being stacked may permit the detection of the stacking gesture (see figure 3-20 on page 80),

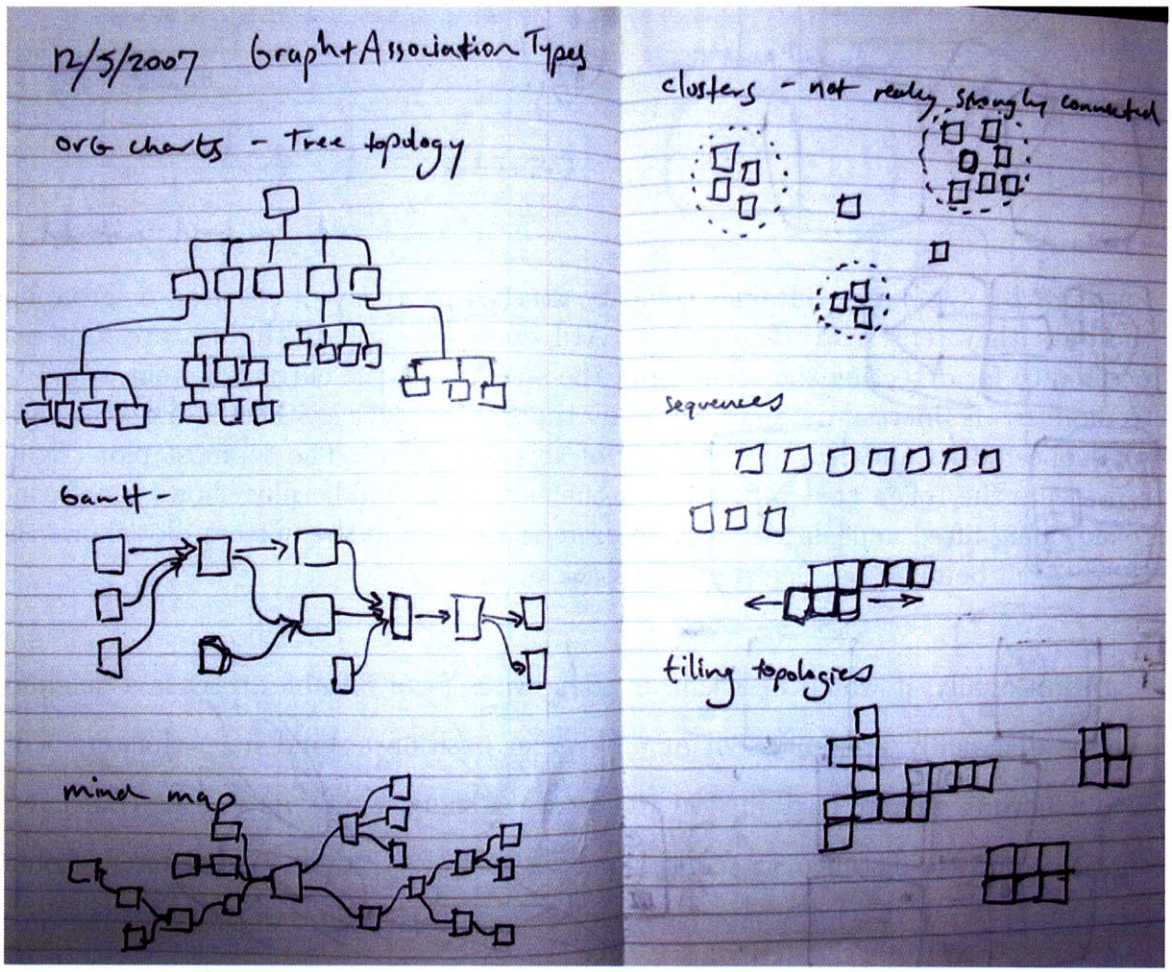


Figure 3-19: Sketches showing two-dimensional topologies that Siftables support. Topologies include tree (top left, bottom left), directed graph (middle left), clusters (top right), sequences (middle right) and 2D tiling topologies (bottom right).

but this possibility has not been investigated thoroughly.

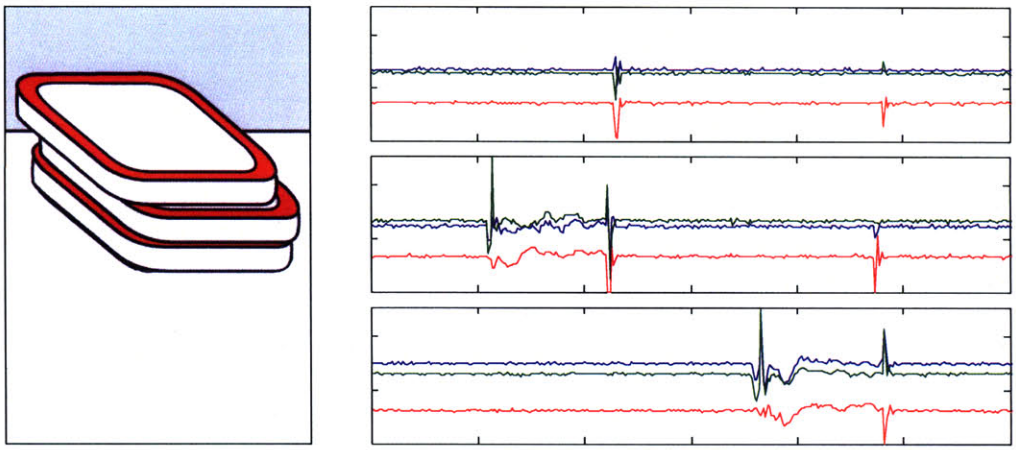


Figure 3-20: Stacking: Siftables could be stacked as a way of ordering or grouping the data they represent. The current API does not support this feature, but one possibility for detecting the ordering of the stack using the current system would be to monitor the inertial data generated by the stacked devices. The data on the right was collected to understand feasibility of this technique. The topmost plot (right) represents the token that remained on the table, the middle plot shows the second token being lifted and placed on the aforementioned, and the bottom plot shows the third token being lifted placed on the stack of two.

Additionally, it would be useful in some cases for a Siftable to sense a neighbor that is diagonally proximate, or at any radial position around its perimeter. This would permit greater flexibility in topological arrangements, as well as continuous spatial input; for instance, the continuous position of a Siftable along a circle around the perimeter of another could be used to tune a parameter such as volume in a musical application.

Taking the previous thought to its logical conclusion, Siftables would benefit from *absolute position sensing.* If a group of Siftables could determine their two-or-three-dimensional position in space with respect to each other to an accuracy of about a centimeter, they would claim a significant amount of the capability that tabletop interfaces possess, namely the ability to utilize the workspace as a coordinate system (see section 3.4.4 on page 82 for a discussion of graphics limitations that would remain). Absolute position sensing in three dimensions would extend this coordinate system in ways that would be useful for navigating three-dimensional data. For instance, each

Siftable could be used as a small "window" into a virtual world that could be moved and oriented to view the world from different viewpoints and angles.

Gesture Sensing Limitations

The current sampling rate of the accelerometer is 100Hz. The implication of Shannon's sampling theorem [108] is that by sampling at 100Hz, signals with a frequency less than 50Hz can be detected. An upper limit of 50Hz is acceptable for sensing human gesture, since any muscle-driven motion that a user could apply to a Siftable would not exceed 50Hz. However, some manipulations produce higher-frequency signals that could be useful to detect. For instance, scraping a Siftable along a rugged texture would produce vibrations both above and below 50Hz, as would a sharp impact to the surface that a Siftable rests upon.

Another gesture-related limitation is the inability to sense absolute position. True position tracking is not possible with only one accelerometer, since rotations cannot be sensed. Laura Harris and I have experimented with integrating the accelerometer data to derive a continuous estimate of position. This works to some degree when rotation is not applied. However the inclusion of additional sensors such as a gyro or an additional accelerometer would improve the ability to track absolute position. This could open up greater possibilities for gesture, as discussed above.

Bluetooth Limitations

The current design uses Bluetooth radios for wireless data communication between a Siftable and a computer. Bluetooth is a convenient standard for prototyping wireless devices since it functions as a "cable replacement" wireless data channel between two devices. However, it was not designed to support large numbers of intercommunicating devices, as it only permits seven simultaneous wireless connections at a time to a single radio.

One approach to mitigating the seven-device limitation is to use a Bluetooth router connected to the host computer. These routers typically contain several separate Bluetooth radios; for instance a router with 3 radios would permit simultaneous

wireless connections to up to 21 devices. However, these routers are expensive and would only be a bandage on the device-number limitation, since they still permit a relatively low number of connections. A scaleable solution would be the incorporation of a more flexible radio, for instance a ZigBee-compatible radio or another 2.4GHz RF radio capable of mesh networking. This would allow for greater flexibility both in the number of Siftables participating in an interaction at a given moment, and would permit dynamic inclusion of new Siftables as they become available. These capabilities would make a Siftables user interface more robust, since new Siftables could be engaged if existing ones are low on battery power, and would allow groups of people to use their personal Siftables together in an ad-hoc manner, in collaborative applications such as games or schedule planning.

Display Limitations

The graphics capabilities of Siftables are limited compared to a tabletop interface with graphics integrated into the work surface. The key difference is that Siftables can display graphics on their built-in screens, but *not in the spaces between manipulatives.* The design trade-off, as mentioned in chapter 2 on page 25, is increased mobility resulting from minimal reliance on bulky infrastructure. However, this limitation presents certain challenges; for instance, in an application with an underlying graph representation featuring nodes and edges, Siftables can show icons or dynamic depictions indicating the contents of the nodes, but edges (the connections between the nodes) are more difficult to represent. One approach to addressing this limitation would be to include more nuanced graphics on the Siftables' screens, as I have done in the graph application described in chapter 4 on page 85. However, this approach is not ideal; compared to tabletop interfaces the amount of pixel real-estate available per manipulative is small. One possible future approach to solve this problem is horizontal projection of graphics by each Siftable, as discussed in chapter 7 on page 161.

Power Limitations

Each Siftable is powered by a rechargeable Lithium-Polymer battery, which permits between 4-10 hours of operation before requiring recharging (see chapter 5 on page 113 for details). This duration has been acceptable for the purposes of my research, however for a real-world deployment it would be preferable to have a longer time between charges, for greater flexibility of applications, particularly mobile interaction contexts. Optimizing the design of graphics and the sleep states of the display and microcontrollers is future work that could extend the battery life significantly.

Chapter 4

Applications

The original idea for Siftables was that they would permit manipulations of existing collections of media. We did not specify whether Siftables should be interfaced to desktop or laptop computers or should operate as a stand-alone interactive platform; the characteristics of the interactive experience were the key concern. A collection of Siftables can provide distributed, physical manipulation affordances to a software application running on a laptop or desktop computer, allowing for other computational resources such as audio or Internet connectivity to be leveraged by an application. As a stand-alone platform, Siftables can provide both input and output without requiring the use of a laptop or desktop computer entirely for certain applications. In practice, I have found it more convenient to develop applications that run on a computer and communicate wirelessly with Siftables. As mentioned in chapter 1 on page 17, I consider both application forms to be instantiations of embodied media since the laptop is quite portable compared to most surface or tangible pucks interaction systems.

The following sections discuss a number of applications that were created by myself and collaborators to explore the possibilities of the Siftables platform. The discussion begins with applications that utilize only the displays of the Siftables for visual feedback, then describes applications that utilize a large display as well. The chapter concludes with a number of application sketches that were implemented as prototypes to investigate various interaction possibilities. For a compact listing of applications and sketches, along with the features that each uses, see figure 4-19 on page 112.

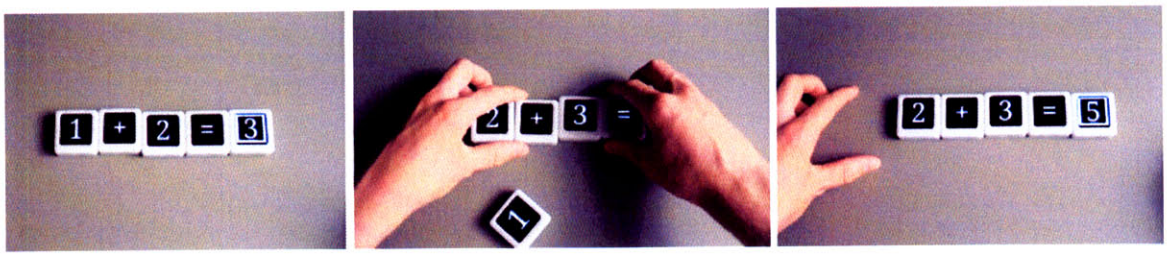


Figure 4-1: Equation Maker is an interactive equation editor. On the left is a complete equation. In the center, the user re-arranges the equation. On the right, the display has updated to show the correct value to the right of the equals sign.

4.1 Equation Maker

Jeevan Kalanithi created an application that allows simple mathematical equations to be created and computed. In the Equation Maker application, two Siftables show operators and a variable number of others display numbers. Arranging the Siftables into a valid ordering (for instance, [1][+][2][=][X], where X is a number Siftable) triggers the calculation and display of the value to the right of the equal sign. Whenever the Siftables are re-arranged, the computation updates immediately and the result is displayed (see figure 4-1).

The application so far features addition and subtraction. To change the operator from a plus to a minus, the user can tilt the operator Siftable to an angle greater than 45 degrees and back. Tilting the Siftable again change the operator back.

Equation Maker is a strong example of a "pure" embodied media application, since all interaction takes place with the Siftables, and no external display or other interface resources are required. Additionally, the ability for a user to see the result of their constructed equation immediately on the Siftable the right of the equal sign takes advantage of the tight communication loop between the computational resource (the laptop computer running the application) and the manipulatives themselves. In informal usage settings the Equation Maker application has been received enthusiastically by users.

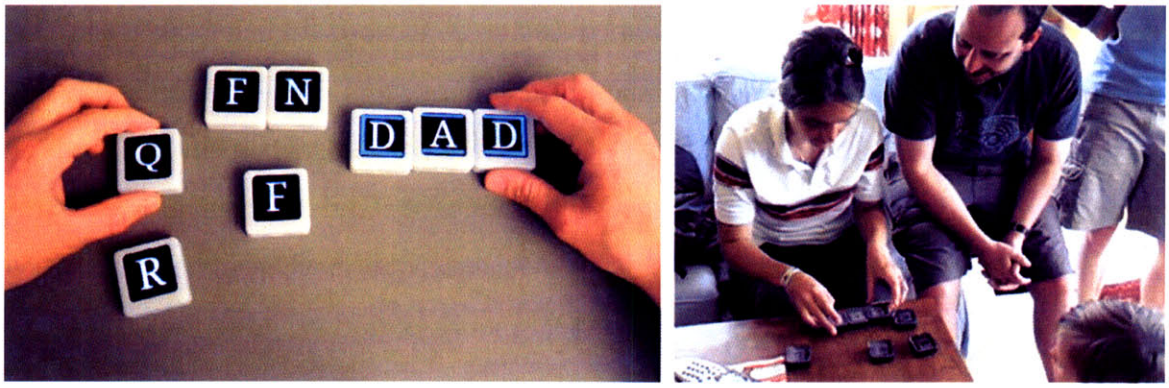


Figure 4-2: Scraboggle is a word-finding game similar to the popular tabletop games Scrabble and Boggle. It utilizes the display, neighbor-detection, and wireless radio communication capabilities of Siftables, and triggers sound effects on the host computer.

4.2 Scraboggle

Scraboggle is a word-finding game in which alphabetic characters are displayed on the screens of a set of Siftables (see figure 4-2). The name derives from popular word-making games Scrabble and Boggle. During each round of the game, each Siftable's screen displays a randomly-selected letter of the alphabet. Users create as many words as they can in a given period of time by placing Siftables into contiguous rows or columns to spell words. As the user attempts to create words, the sequences that they create are checked against a dictionary. Whenever the user constructs a valid word, the screens of the involved Siftables draw a colored border to encircle the word and an audio sample is played from the computer. After one second, the screens are redrawn without the border. As each round progresses, the brightness of the Siftables' screens decreases linearly until all of the screens are completely dark, signaling the end of the round. At the conclusion of each round, a new letter is randomly assigned to each Siftable, the screens return to full brightness, and the round timer begins again.

Scraboggle enabled the exploration of how Siftables could implement a constraint-satisfaction task with a requirement for domain knowledge, in which rapid re-arrangement and feedback played an important role. In a study using Scrabble tiles, Kirsch and

Maglio [71] found that people performed better at a word-finding task when they could use their hands to arrange the tiles, as compared to a version of the task where they were only allowed to visually inspect the available tiles. Their finding supports the utility of physically arranging letters into sequences, and Siftables augments this interaction with the capability of real-time visual feedback when a word is spelled. Informal usage by tens of users over a several month period suggests that the ability to physically create word sequences was compelling.

As the first application written with the current Python API, Scraboggle was a useful test bed for application development. All application-specific logic relies entirely on the Python API and it utilizes bitmap images stored in the Siftables' on-board flash memory. Scraboggle does not require any modifications to the standard firmware, and in this sense is a complimentary example to Attentionables (see section 4.4 on page 92). After completing Scraboggle I abstracted a standard application template that includes solutions to problems that arose during my authoring of the application, that has been a useful starting-point for a number of subsequent applications. The abstracted features include an application loop that waits for asynchronous events and generally handled concurrency, topology determination when Siftables are placed in a row or column and program exit that gracefully disconnects the wireless Bluetooth links to the Siftables.

4.3 Music Sequencer

I built a music sequencer to explore the use of Siftables for a relatively sophisticated and expressive application (see figures figure 4-3 on the facing page and figure 4-4 on the next page). During play, the application allows the user to create multi-layered sequences of samples, to order and re-order a sequence, to attach an effect to a particular voice and gesturally manipulate the effect, and to manipulate global effects that apply to the entire sequence.

I wrote the Siftable-control and on-screen feedback behavior using the Python API, and Josh Kopin built the music engine with a combination of Pure Data [99] and

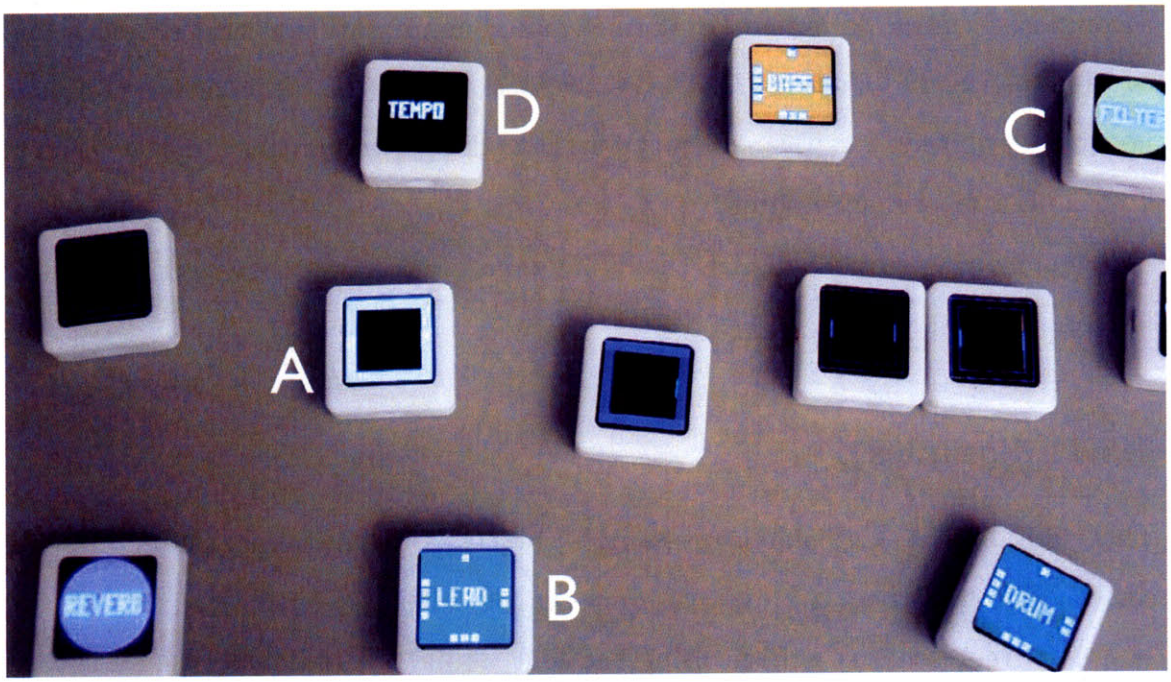


Figure 4-3: The music sequencer application. Siftables are configured as (A) initially blank *sequence* tiles, (B) *voices* such as lead, bass, and drums, (C) *voice effects* that can apply to a single voice, such as filter and reverberation, and (D) *global effects* that apply to the entire piece such as tempo and volume.

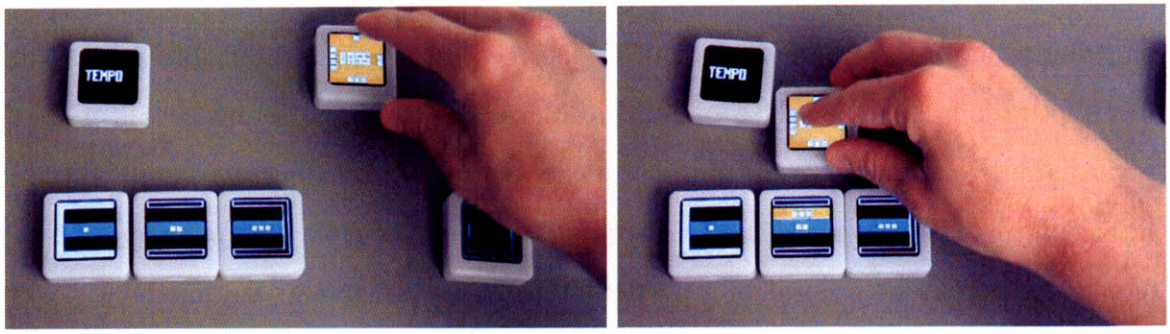


Figure 4-4: The music sequencer application in use. The user grasps the Siftable configured as the Bass voice (left), and adds Bass to the middle sequence by bumping it against to the sequence Siftable (right). A bar appears on the screen of the sequence Siftable, drawn with the same color as the Bass Siftable's background, and with the same number of square markings as the side that was bumped against it.

Ableton Live [1]. Communication between Pure Data and the Python interpreter was accomplished using datagram (UDP) network communication, formatted according to the Open Sound Control (OSC) [74] protocol. Since the Siftables themselves to not include speakers, it was necessary for the audio portion of this application to be hosted on a separate computer.

The following sections discuss the interactions of the music sequencer application in more detail.

4.3.1 Samples

Three Siftables are designated as "sample" objects. When the program starts, the screen of each of these Siftables is drawn with a unique color and the name of the sample it represents is written using the font library. Additionally, small squares are drawn along each of the four edges of the screen, visually marking each edge to represent a different variation of the sample. See the "Sequences" section below for an explanation of how samples can be inserted into sequence Siftables.

4.3.2 Sequences

A variable number of Siftables are designated as "sequence" objects that begin empty, showing a blank screen. Samples can be inserted into sequence Siftables by placing any of the four edges of the sample Siftable adjacent to the sequence Siftable. A given sample can be removed from a sequence Siftable by placing the sample Siftable adjacent to the sequence Siftable a second time. During play, zero or one of each sample class may be present in a particular sequence Siftable at any moment.

When the application begins, one sequence Siftable is designated as the "playback head" or "index" Siftable, and a bright border is drawn around its perimeter. At this point, the "active sequence" comprises only this single sequence Siftable, and the program loops continuously, playing any samples that have been inserted into it.

The active sequence can be extended, reduced, and rearranged while the program plays. Sequence Siftables that come in contact with the active sequence become

"active", extending the length of the active sequence. The active sequence is looped continuously. This means that the sequencer iterates across the Siftables in the active sequence, and at each step the samples present in the given sequence Siftable are played in unison. The border of the sequence Siftable being played is highlighted to provide visual feedback to the user. The border is present but not highlighted for sequence Siftables that are not in the active sequence, or that are not currently being played.

4.3.3 Sample Effects

Some Siftables are designated as "sample effects" meaning that they represent effects that can be *attached to a particular sample, then manipulated.* These Siftables show a colored circle that fills the screen, and the name of the effect is drawn on top with white lettering. In the current design, these effects include reverberation and a sweeping band-pass filter that acts like a "wah" pedal. A sample effect can be attached to a given sample by placing the sample effect Siftable adjacent to the sample Siftable. It can be removed by placing the two Siftables adjacent to each other a second time.

When a sample effect is attached to a sample, the sample Siftable displays a small circle drawn with the same color as the circle on the sample effect Siftable. This visual feedback is designed to help the user remember which effects are enabled for a given sample. When a sample effect is removed from a sample, the small circle disappears.

When a sample effect is attached to a sample, the user can manipulate the effect by tilting the effect Siftable along its X (left/right) axis. A direct mapping from instantaneous tilt to parameter value is used, and the tilt data is scaled to match the range of the effect. This means that tilting the sample 90 degrees to the left (such that the effect Siftable is standing on edge) will produce an effect value of zero, while tilting the sample Siftable 90 degrees to the right will produce the maximum effect value.

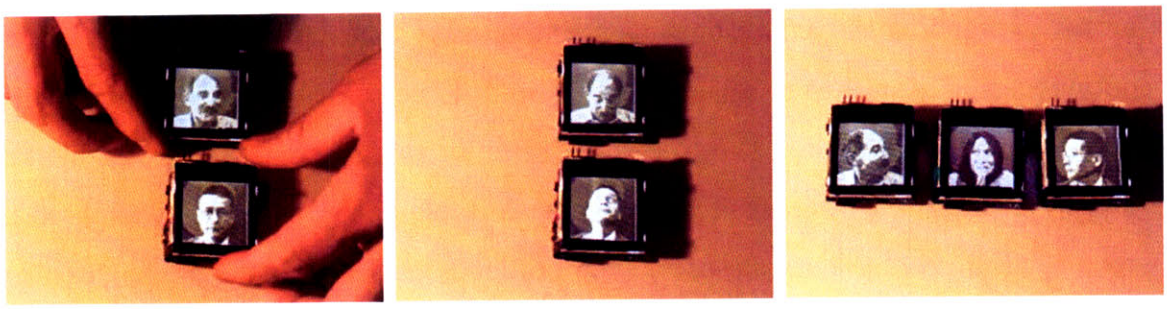


Figure 4-5: Attentionables was the first complete application created for Siftables. It utilizes the display, neighbor-detection and accelerometer and is an adaptation of Zuckerman and Sadi's Spotlight installation (see figure 4-6 on the facing page).

4.3.4 Global Effects

There are two "global effects" that affect the active sequence as a whole, and that do not require attachment to any particular sample. In the current version, these effects are volume and tempo. The Siftables representing these effects show a black screen with white text drawn to indicate the effect name. When a global effect Siftable is resting on the table, its value does not change. However, when it is lifted from the surface and tilted past 45 degrees, it exits a tilt "deadband" and the value begins to change at a regular interval until it returns within 45 degrees of flat again. To increase the tempo of the active sequence, for example, the tempo global effect Siftable is lifted from the table surface and tilted to the right. For as long as the Siftable remains tilted, the tempo increases, and when the Siftable is held flat or laid to rest on the table again, the tempo remains at the value set by the manipulation.

See section 4.12.2 on page 105 for a thorough discussion of tilt input.

4.4 Attentionables

Attentionables was the first application with any significant complexity that was built for Siftables. It is an adaptation of a wall-sized interactive art installation called Spotlight [123], by Orit Zuckerman and Sajid Sadi (see figure 4-6 on the next page). The original Spotlight piece features a matrix of "active/social portraits" affixed to a wall. The portraits are actually composed of a sequence of pre-recorded

Figure 4-6: Spotlight is an interactive art installation by Orit Zuckerman and Sajid Sadi that expands on the techniques of classic portraiture [123]. On the left, a visitor is pressing a button on a keypad on which each key shows a face from the installation. On the right, the selected face reacts while the others turn their attention in its direction.

video clips, and they can interact with each other autonomously or as a result of input from the viewer. The default state of the Attention piece features each face looking forward towards the viewer. Every 15 or 20 seconds, a pair of nearby portraits looks towards each other for a few seconds, then reverts to looking forward. Using a specially designed remote control, a viewer can cause all of the portraits to look towards a single portrait, and that portrait reacts in a manner that demonstrates the individual's personality, as if they suddenly found themselves the center of attention in a crowded room. Spotlight explores the boundaries of portraiture, giving the viewer a more intimate and dynamic understanding of the depicted individuals than a traditional static, single-snapshot portrait would allow.

Attentionables incorporated the video content and the core behavior patterns of Attention, but it enabled new interaction possibilities that were not available in the wall-mounted installation. In Attentionables, each portrait is displayed on the screen of a Siftable. When two Siftables are placed near each other, the faces look

in the direction of each other. When a Siftable acquires more than one neighbor, the newly-surrounded portrait looks at each neighbor in turn, then displays the "center-of-attention" sequence. The portraits also react to gravity, looking in the direction of tilt if they are not resting flat in the X/Y plane. Finally, shaking a portrait will also trigger the "center-of-attention" sequence. Evan Broder wrote the first version of Attentionables during the summer of 2007. I subsequently re-wrote Broder's version to utilize the C API for attaching behavior to events generated by the Siftables' operating system. The video clips are stored as sequences of image frames in the flash memory, and callback handlers are registered for neighbor, tilt, and shake events.

The implementation of Attentionables enabled three extensions to the behavior that were not possible in the original Attention piece. The first was that the layout portraits could be re-arranged. Any combination of neighbors could be created, and the topological arrangement of any two Siftables (i.e. which side of A was facing which side of B) could be easily manipulated. Second, the center-of-attention gesture can be triggered by the user in a more interactive manner: by surrounding a portrait with others or shaking it. Finally, by responding to tilt, the portraits acquire a proprioceptive element, aware of their own "body" as well as interactions with the environment. In informal usage settings, users have reported that this ability enhances the lifelike quality of the portraits.

4.5 Maze Exploration

Tobe Nwanna wrote a maze exploration application. The conceptual model of the game is that the user can explore the corridors of a maze, but only a limited amount of the territory can be seen at a given time. The application utilizes two Siftables, and the interaction model is that the user's character, represented by a black circle, exists in a single grid-square of the maze at any given moment. The Siftable showing the user's character is drawn with a bitmap loaded from the flash memory that shows the local passageway. A given grid-square can be a straight-through, right-angle, or a "T" junction. By placing the second Siftable next to the Siftable showing the user's

Figure 4-7: A maze exploration game illustrates some methods for navigating structures that are too large to be completely represented on the screen of a collection of Siftables at a single instant. The user can "dump" their character from one position to an adjacent position and can view a summary of the already-explored territory.

character, the adjacent passageway can be viewed. If the map permits it, the user can transfer their character's position to the adjacent square by tilting the primary Siftable in the direction of the adjacent square, "dumping" the character into the adjacent square (see figure 4-7).

Shaking the secondary Siftable triggers a map mode. In map mode, the secondary Siftable shows a small map of the squares that have been already explored by the user, as well as a dot indicating the character's current position. Shaking the secondary Siftable again brings the game back to play mode.

The game also features dangers. Certain locations have pits that the user's character can fall into, ending the game. Also, an enemy "ghost" wanders the passageways, moving continuously from square to square. If the ghost reaches the user's character, the game ends.

This application explores how Siftables can be used to navigate a space or a piece of digital content that is too large to be fully represented on their screens at a single moment. It also provides an example means of transferring the user's point of view from Siftable to Siftable, using the physical metaphor of "dumping" an item by tilting one device towards an adjacent device. Finally, the "map mode" provides a compact way to summarize the territory that has been already explored.

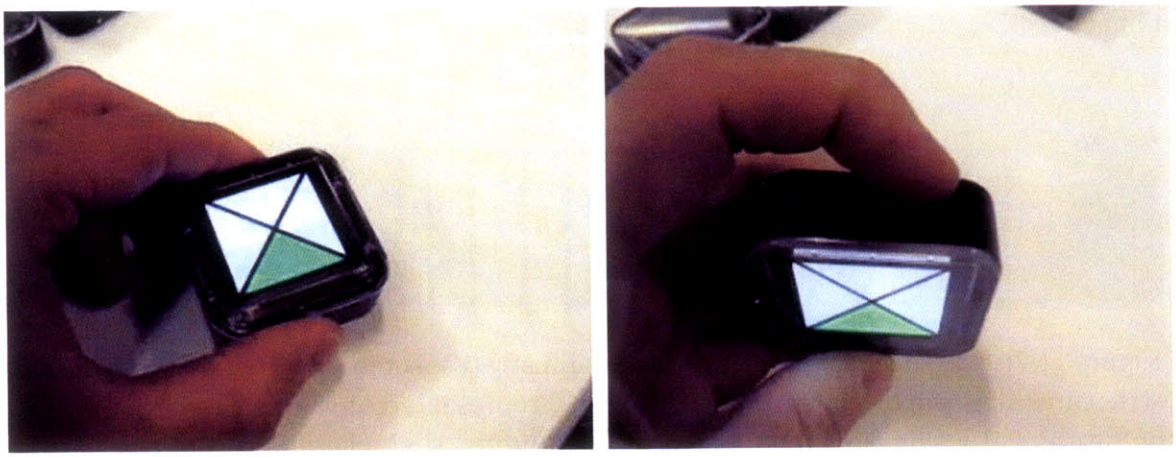


Figure 4-8: Simon: An adaptation to Siftables of the classic sequence-memory game. In the single-Siftable version, the user tilts a Siftable in the direction of the triangles shown on the screen to input the sequence.

4.6 Single-Siftable Simon

Rick Mancuso created an application similar to the classic memory puzzle game "Simon." The original Simon game features a device with four large buttons covering its top surface. Each button is a different color, and can be illuminated. The game is a memory activity in which a sequence of ever-increasing length is presented by the system by lighting up buttons and playing an auditory tone each time a button lights up. In each round, the user presses the buttons in the order presented by the system, which becomes increasingly difficult as the sequence gets longer. In the Siftables version, the buttons from the original Simon game are represented by colored triangles displayed on the screen of the Siftable. Rather than pressing a button, the user tilts the Siftable in the direction of the desired triangle (see figure 4-8).

4.7 Multi-Siftable Simon

This application is similar to Single-Siftable Simon, but it uses four Siftables. Each represents a button from the original Simon game, and the sequence is expressed by drawing a border around the Siftables from the sequence in order. To input their recollection of the sequence, the user shakes the Siftables in order, one at a time.

4.8 Tilt-Based Color Etch-A-Sketch Drawing

Tobe Nwanna also created a single-point drawing application. It features a drawing mode in which the tilt of the Siftable causes a single-pixel "tip" to draw in the direction of gravity at a fixed rate, and a color-selection menu that allows the user to change the active drawing color. The color menu is invoked and dismissed by shaking the Siftable, and while the menu is open tilting the Siftable towards the top or bottom can traverse it. Noting that tilt-to-scroll can be a problematic user interface paradigm for handheld devices, the decision was made to discretize menu-navigation. Each time the Siftable is tilted in a direction, the highlighted item advances by one menu item in the given direction. The Siftable must be tilted back to a flat orientation before the marker can be advanced again. See section 3.4.3 on page 76 for more about this style of menu navigation.

A unique of this application is the large amount of state that is kept in the Python environment: the entire drawing is represented as a collection of points in an array in the Python interpreter, so that when the user returns from menu mode, their drawing is re-created by sending the sequence of points to be drawn to the Siftable. This exposes a limitation of the current Siftables design, the speed of drawing graphics to the screen. Re-creating the user's drawing can take up to several seconds, depending on the complexity of the drawing. Nwanna generalized his menu architecture, creating a library that allows other developers to create their own menus.

4.9 Fiddle Diddle Make a Riddle

Seth Hunter, a colleague at MIT, consulted with a first grade teacher in the Boston Public School system to design and author a language-learning application using Siftables. The application is targeted at children aged 4-7, and it is based loosely on the children's book *Hop on Pop*, by Dr. Seuss [107].

The focus of the application is to teach children basic sentence-construction skills through creative play. The application allows children to explore creative analogies

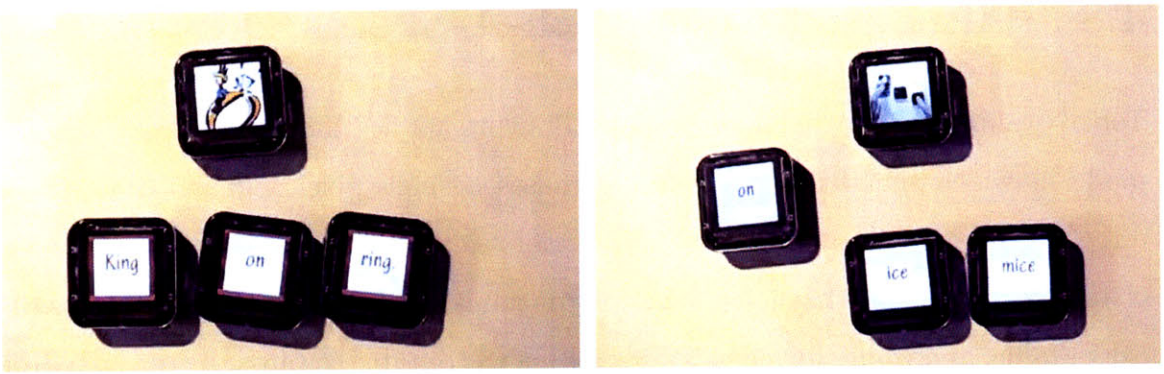


Figure 4-9: Fiddle Diddle Make a Riddle is a language-learning application for children that teaches prepositions and word order in simple sentences. On the left is the sentence "king on ring" spelled with three Siftables, and a picture of a small king sitting on a ring on the fourth Siftable. On the right is an example of an on-Siftable instruction, showing the user that they need to re-arrange the Siftables in order to move forward in the application.

by introducing humorous relationships between words and allowing them to structure those relationships by arranging Siftables in multiple sequences.

The application aims to teach phonemic awareness (rhyme endings with different spellings), semantic sentence structures (what image does the sentence imply?), creativity through control of language (experimenting with different arrangements), and vocabularies and fluency through image/word association, grapheme awareness through word families (the -ice of mice and lice), and basic punctuation (when blocks are switched the capital letters and periods are switched).

As an example, one run of the application features the words "ice", "on", and "mice" each on a separate Siftable. When the user arranges the three Siftables to spell "mice on ice", the fourth Siftable shows a picture of two cartoon mice wearing ice skates on a frozen rink. Re-arranging the three words so that they spell "ice on mice" causes the fourth Siftable to show an image of a mouse with icicles hanging from its whiskers. At the instant that the image on the fourth Siftable changes to display the result of the sentence, the sentence is also spoken aloud from the computer with a pre-recorded audio sample, reinforcing the spelling task with auditory spoken language (see figure 4-9).

A feature that sets this application apart from others is its inclusion of just-in-

time graphical hints that are displayed on the Siftable. A hint appears if a certain amount of time has elapsed in a single "round" of the application and the user has not yet spelled out a valid sentence. When these conditions are met, the fourth Siftable shows a short video clip of two hands re-arranging a collection of Siftables, to suggest to the user that they should re-arrange the manipulatives. Another similar hint happens after a user has completed both possible sentences at least once, but has not changed the order of the Siftables for several seconds. In this condition, a short video clip is shown of a hand reaching out, picking up a Siftable, and shaking it. These on-Siftable hints are an interesting example of how future applications could be self-teaching, helping a user become "un-stuck" when they have run out of ideas for how to proceed. As an extension of this concept, hints could coordinate visual information across several Siftables, optionally in conjunction with audio.

4.10 Telestory

Seth Hunter and Katya Popova created Telestory, a language-learning application designed to teach vocabulary to children that are not yet old enough to read. This application comprises an interactive cartoon on a large display featuring characters that are represented both on a Siftable and on a large computer or television display. A primary activity in Telestory is introducing props and characters into the scene by picking up the Siftable displaying the given item. A character and a prop can also be introduced to each other by placing the Siftables that represent them adjacent to each other (see figure 4-10 on the following page). The introduction of a character to any other object triggers an animation in the cartoon that involves the two entities, as well as auditory speech playback that introduces the new object or interaction.

Hunter created an interactive tilt-based menu for Telestory to allow a child to select props. The menu divides the screen of the Siftable into four quadrants, and shows a thumbnail image of each prop in each quadrant. One of the four quadrants starts out in full color, while the other three are faded. Tilting the Siftable toward a particular corner causes the thumbnail in that corner to become the selected item,

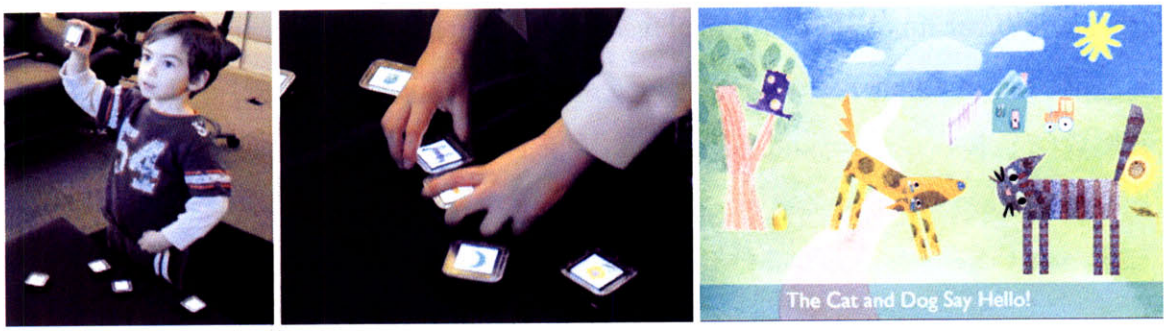


Figure 4-10: Telestory is a language-learning application designed to teach vocabulary to children that are not yet old enough to read. On the left, the child lifts a Siftable containing a prop to bring that prop onto the large screen. In the center, he places the Siftable showing the dog next to the Siftable showing the cat. On the right, the dog and cat meet as a result of the adjacency.

and to be shown in full color, while the others become faded. If a given prop remains selected for two seconds the selected item enters the cartoon on the large screen. This menu allows each prop-Siftable to represent four different items, and is described further in section 3.4.3 on page 76.

Hunter observed two children aged 4 and 5 interacting with Telestory in separate sessions. He found that both children seemed to easily understand the connection between the Siftables and the corresponding on-screen characters. Direct correspondences, such as raising the Siftable displaying the sun to make the sun rise in the cartoon, or shaking the Siftable showing the dog to make the on-screen dog shake, were the most popular. The four year old took great pleasure in making the cat character sneeze after sniffing a sunflower by placing the Siftables side-by-side, and he triggered this interaction more than twenty times.

Interactions that required the child to shift their visual attention from the screen to a Siftable did not work as well, as they tended to keep their eyes on the screen except when finding a new Siftable to grasp. For this reason, the tilt-based menu took longer for the children to understand and use.

Telestory uses only pre-loaded graphics and the Python API. A takeaway observation from Telestory is that more effective Siftables applications for children may be ones that either create a direct connection between manipulation of a Siftable

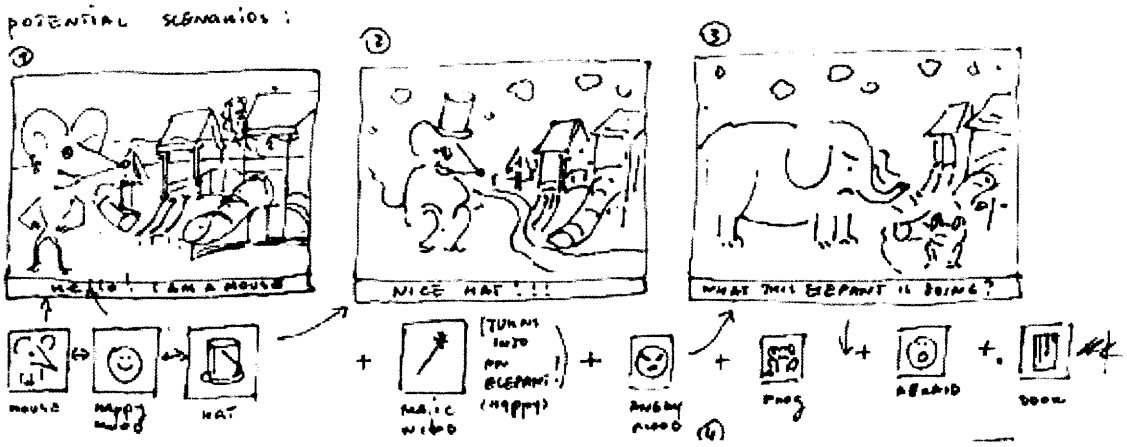


Figure 4-11: An early storyboard for the Telestory application, sketched by Katya Popova.

and a corresponding reaction from an on-screen entity, or that keep the content and interaction exclusively on the Siftables themselves. This lesson is corroborated by open-ended responses collected after the pilot trials for the image manipulation application.

4.11 Image Manipulation

I built an image manipulation application with Siftables that allows users to apply image-processing filters to digital images displayed on a computer screen. In the interaction, a single Siftable represents the original image, and it shows a thumbnail-sized version of the unmodified source image on its screen. The remaining Siftables represent image-processing filters, and in its default resting state, each filter-Siftable displays the name of the filter on its screen (for example, "Blur", "Threshold", etc.). A full listing of filters, and descriptions of their effects can be found in figure 4-13 on page 103.

A user can create an ordered sequence of filters by placing the filter Siftables adjacent to each other into a row. To apply the effects to the source image, the sequence of adjacent effect Siftables is placed adjacent to the source image Siftable, on its right side. The sequence of filters becomes "active" and is applied to the source

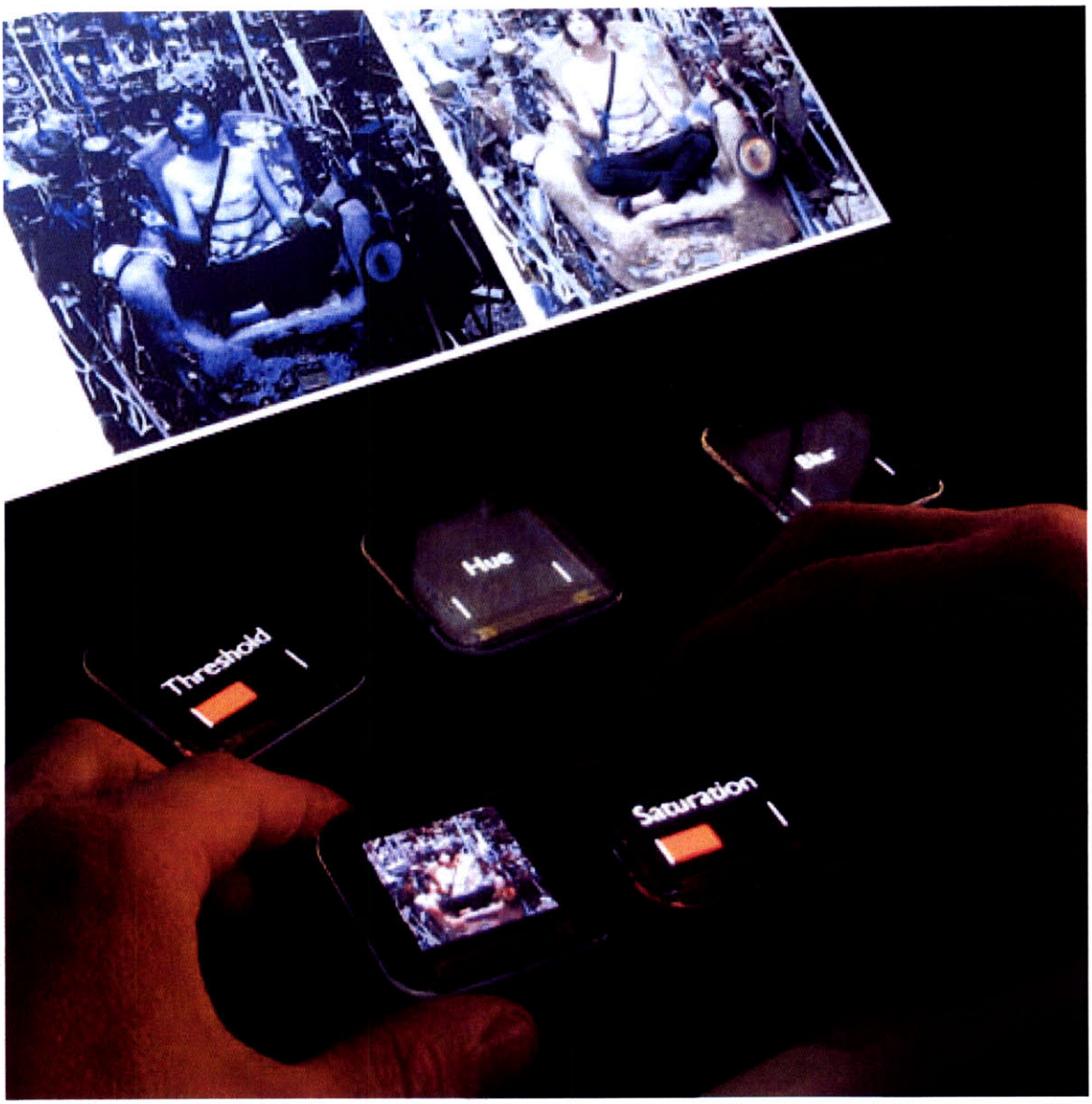


Figure 4-12: The *image manipulation* application in which individual image-processing effects can be engaged or disengaged by placing them into a contiguous sequence to the right of the "image" Siftable that shows a thumbnail of the original image. The "effect" Siftables display the effect's name when they are at rest, and show a live preview of the effect during adjustment.

Blur	Application of Blur reduces the image's high-frequency content, producing a perceived effect similar to de-focusing an image. At the extreme low end of the manipulation range, no change is applied to the image. At the extreme high end, the image looks very unfocused. The neutral point for this effect is at zero.
Saturation	Saturation controls the perceived intensity of the colors in the image. Application of this effect with a value of zero will produce a grayscale image. At the extreme high end, colors look more vibrant than the original. The range of this effect was set such that the neutral point is at fifty percent.
Brightness	Brightness controls the overall luminosity of the image. Application of this effect with a value of zero will produce an entirely black image. At the extreme high end, the image will be completely white. The range for this effect was scaled such that the neutral point is at fifty percent.
Threshold	Threshold converts the source image into a 2-color image containing only black and white pixels. The criteria for whether a given pixel from the source becomes white or black in the result is related to the source pixel's original brightness; if the brightness is over the user-defined threshold, the resulting pixel in the output will be white, and if it is below the threshold the resulting pixel will be black. This effect always changes the resulting image, unless the original image consists of only black and/or white pixels. The starting-point for this effect was set to fifty percent.
Hue	Hue determines the color mapping from input to output pixels, following a circular color-wheel pattern. Adjustment of the effect will rotate the colors of the image through a full 360-degree cycle such that at either the low or high end of the scale the image colors are unchanged. At intermediate values, the perceptual effect is that the salient colors of the image progress through variations, becoming more yellow, green, blue, purple and red. The scale for this effect was designed such that the neutral point is at 0 (same as the maximum value).

Figure 4-13: A full listing of image processing effects used in the image manipulation application, and descriptions of each effect.

Figure 4-14: Image Manipulation application: Adjusting the brightness parameter by lifting and tilting the "brightness" Siftable, the placing it back into the filter chain. While the adjustment is in progress, the Siftable's screen shows a live preview of the effect, then once it is placed down again it reverts to showing the name of the effect and red value-bar.

image in left-to-right order. The full-sized image on the computer screen updates accordingly. Every change to the active sequence to the right of the source image Siftable triggers an update of the result image on the computer screen.

The user adjusts the magnitude of each filter Siftable by tilting the Siftable on the X (left/right) axis. In this way, an effect can be lifted off the table by the user, adjusted, and then placed back onto the table, either into or out of the active sequence (see figure 4-14).

Each of the effects that are used is described in the following subsections. All effects except Threshold have a neutral point, a setting at which they produce no perceived change to the resulting image. In the application, the value of each effect is set initially to its neutral point, and the value of Threshold is set to fifty percent.

The image manipulation application investigated how Siftables can be used in exploratory manipulation of digital images. It also allowed for exploration of the use of Siftables in conjunction with a computer screen. See chapter 6 on page 135 for feedback from a study that was run using the image manipulation application.

4.12 Application Sketches, Interaction ideas

This section documents application sketches. These sketches were implemented as prototypes to investigate various interaction possibilities, but were not developed to the level of complete applications.

4.12.1 Grouping and Ordering

I created a simple application to evaluate the efficiency of content grouping and ordering activities using Siftables (see section 6.2 on page 136). The application used the alphabetic character images that were created for Scraboggle, a set of numeric digit images between 0-9, and rectangles of different colors that were drawn to the screen. The application featured three different states, one for each visual content type. In the alphabetic character and numeric digit states, the user's goal was to place the Siftables into a linear ordering, either alphabetic or numeric, both increasing to the right. When colored rectangles were drawn to the screen, the user's goal was to separate the Siftables into two groups, putting alike colors together. For information about the experiment, see section 6.2 on page 136.

4.12.2 Node Edge Graph Creation

I wrote a graph-creation framework to explore how graph topologies can be represented and manipulated using Siftables (see figure 4-15 on the next page). In this framework, a node can be represented by each Siftable, or by each side of each Siftable. Placing two Siftables adjacent to each other creates an edge connecting the two nodes. When an edge is created, each of the newly-connected Siftables shows a colored dot on its screen, along the edge that was placed adjacent to the other Siftable. The pair of dots is assigned a color that is unique to that particular edge. Placing the same side of the two Siftables adjacent to each other a second time removes the edge and causes the associated dots to disappear.

The node-edge creation tool was a proof-of-concept, to illustrate how Siftables could be used in process modeling applications such as supply-and-demand or predator-prey models. Along with their topological layout, another important feature of many process models is that continuous values can be set at a particular nodes. For instance, if a node represents a rate of change or production, adjusting this rate is important to understand and control the dynamics of the model. I created a tilt-based value-adjustment interaction that enables a continuous value to be set at a given node.

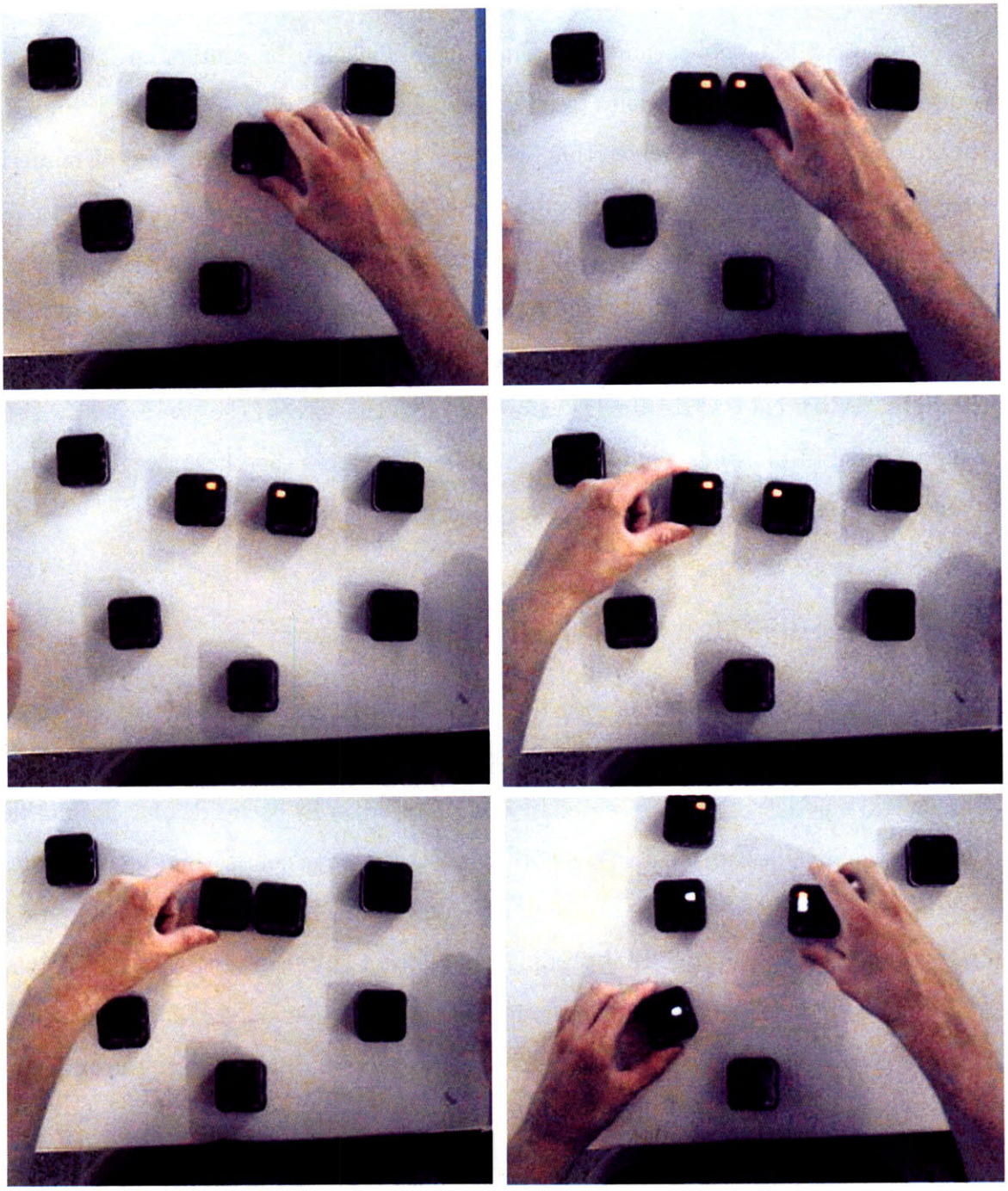


Figure 4-15: The node edge application sketch allows graph topologies to be created. Colored visual feedback is drawn on the Siftable's screen to allow the user to view edges; each edge is assigned a unique color so that the user can distinguish them visually. From left to right, top to bottom: creating an edge (1-3), removing the edge (4-5), and a later state in the interaction where three edges have been created (6).

There are two basic ways to map a tilt gesture onto a continuous value, instantaneous and time-based, which are explained in the following sections.

Tilt-to-value approach: instantaneous

The instantaneous tilt of the Siftable can be sensed and mapped onto the range of possible values. The problem with this method is that placing the Siftable down on the table will result in a "flat" tilt measurement, which may not be the desired value. One solution is to introduce physical props that allow the Siftable to be kept at a particular angle, thus "setting" the tilt to a given value in a way that persists until the Siftable is moved. The introduction of props, although interesting, seemed like an awkward solution.

Tilt-to-value approach: time-based

Rather than linking the Siftable's instantaneous tilt to a continuous value, the value can be set initially to 0.5. When the Siftable is lying flat on the table, this value remains constant. However, whenever the Siftable is tilted more than 45 degrees away from a flat state, the variable begins to change at a fixed rate (or optionally at a rate related to the degree of tilt) for as long as the Siftable is tilted. Thus, a 90-degree "dead-band" is imposed around the resting position.

The drawback of the "dead-band" approach is that it wastes the user's time. If they want to adjust the value by a large amount, they must *tilt, then wait* for the time-based updates to change the variable to the desired value. Even using an update rate that is related to the degree of tilt, this can take some time. A non-linear mapping from tilt to update rate may be the best time-based solution, however as other researchers have found [87], overshoot is common using this method.

Tilt-to-value solution: modified instantaneous

The solution I designed is a modification of the direct angle-to-value mechanism. In order to allow a value to be "locked in" without requiring a physical prop, I created a shake-to-lock/unlock scheme. To begin, shaking the Siftable puts it into

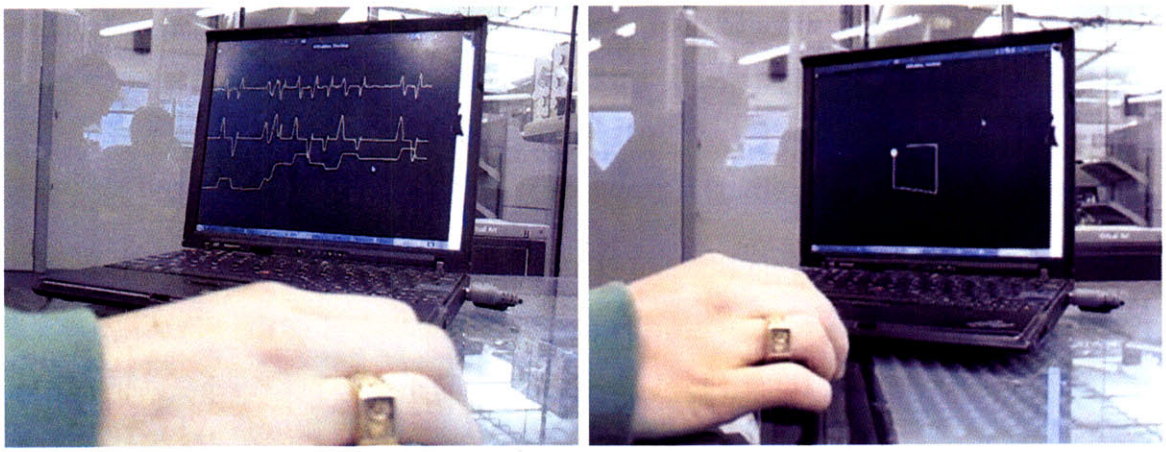


Figure 4-16: A position tracking application sketch integrated the accelerometer data to continuously estimate the two-dimensional position of the Siftable while the user moved it across a tabletop. On the left is the filtered accelerometer output, and on the right is estimated position.

value adjustment mode. While in this mode the direct angle-to-value mechanism is engaged and visual feedback is shown on the screen. A second shake exits adjustment mode. To mitigate the problem that the second shake inadvertently changes the instantaneous tilt value, the value is taken from 0.5 seconds *before* the detection of the shake event. This value was determined experimentally, and allowed the steady-state tilt value to be determined accurately.

The node-edge program was a proof-of-concept that Siftables could support system dynamics scenarios, and future work will explore these possibilities.

4.12.3 Position Estimation From Integrated Acceleration

Laura Harris was interested in the possibilities for using Siftables to navigate and manipulate three-dimensional computer-aided design (CAD) models. As a building-block, she wrote an application that processes the raw live accelerometer data to continuously estimate the two-dimensional position of a Siftable as it is moved by a user, and to display a representation of the estimated position on the computer screen (see figure 4-16). We did not formally evaluate the accuracy of this application sketch, but found that for short movements in the X/Y directions (where each movement

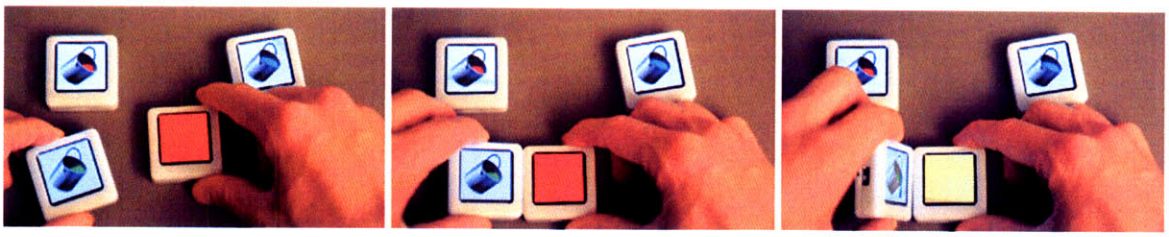


Figure 4-17: Color pouring application sketch. First, the user grasps the paint bucket and receptacle Siftables (left). Then, the two Siftables are placed adjacent to each other (center). Then, the paint bucket Siftable is tilted toward the receptacle Siftable, and its color is "poured" into the receptacle, mixing with the existing color (right).

consisted of mostly X or mostly Y motion, but not both), the on-screen position representation tracked the Siftable in a satisfactory manner. When movements were longer, when they featured rotation of the Siftable, or featured significant motion on both the X and Y axes, the on-screen position estimate diverged noticeably.

4.12.4 Pouring Colors

Jeevan Kalanithi created an application sketch that permits "pouring" a color from one Siftable to another (see figure 4-17). In the simple version of this sketch, one Siftable shows a photograph on its screen, while another Siftable shows a red paint bucket. Placing the paint bucket Siftable adjacent to the image Siftable causes the application to enter "pouring mode", which is visually indicated to the user by an animation showing a drop of paint emerging from the bucket. In this mode, tilting the paint bucket Siftable towards the image Siftable triggers a steady increase in the amount of color "tinting" applied to the image, for as long at the paint bucket Siftables is tilted. The tint is applied to the image in a pixel-by-pixel manner as the image is sent to the screen for display, and as the tint value increases the image is continually redrawn, giving the the user the impression that the color is being *poured into the image.* To reverse the process, the user can tilt the image Siftable towards the color Siftable, *pouring the color back* into the paint bucket Siftable.

The more complex version of the Pouring Colors application features three paint buckets, each showing a different color. Each paint bucket Siftable can be used

Figure 4-18: The tilt-to-roll video sketch utilizes Muybridge's classic series of photographs titled "The Horse In Motion." When the user tilts the Siftable to the right (right image), the frames advance in the forward direction and a right-pointing red arrow is superimposed. When tilted to the left (left image), they proceed in reverse with a left-pointing arrow. When the Siftable is flat (center image), the "at rest" image is displayed.

separately (as in the simple example) to apply its individual "tint" to a receptacle Siftable, and the contribution of the poured colors are mixed in the receptacle. In this version I did not use an image as the receptacle, instead a Siftable that started out with no color on its screen. The receptacle Siftable becomes brighter as component colors are added to it by pouring.

The pouring application could be abstracted away from the domain of color, and this style of interaction could be used for adding any continuous amount of a given property represented by one Siftable to another. For instance, in a game, extra energy could be "poured" into a character Siftable to increase the character's health. Pouring colors is an example of a real-world metaphor that was appropriated for use as a multi-object gestural interaction technique.

4.12.5 Tilt-to-Roll Video

I created an application sketch that uses the X-axis tilt of a Siftable to control the playback of a sequence of movie frames. The application utilizes twelve frames from Muybridge's classic series of photographs called "The Horse In Motion" that are stored in the Siftable's flash memory (see figure 4-18). When the Siftable is at rest on a table or standing on its bottom edge the screen displays the "at rest" frame in

which the horse and rider are standing still. When the Siftable is tilted to the right, it animates through the frames depicting the horse running in sequential order. When the Siftable is tilted to the left, it animates through the same frames but in reverse. The effect is somewhat informed by our experience with gravity: The user can tilt the Siftable one way or the other to "roll" the footage in the desired direction.

Name	C API	Python API	shake	tilt	acc	neighbors	drawing	bitmaps	variables	sound	screen
Equation Maker		X		X		X	X	X	X		
Scraboggle		X		X		X	X	X	X	X	
Music Sequencer		X		X		X	X			X	
Attentionables	X		X	X		X		X	X		
Maze Exploration		X	X	X		X	X	X			
Simon (single)		X		X				X		X	
Simon (multi)		X	X					X		X	
Color...Drawing		X	X	X			X	X			
Fiddle Diddle...		X	X			X		X			
Telestory		X	X	X		X		X		X	X
Image Manip...	X	X		X		X	X	X	X		X
Grouping...		X					X	X	X		
Node Edge...		X	X	X		X	X				
Position Est...		X			X						X
Pouring Colors		X		X		X		X	X		
Tilt-to-Roll...	X			X		X		X	X		

Figure 4-19: A full listing of all applications and applications sketches, showing which capabilities were utilized. *C API* refers to the application programming interface for on-Siftable firmware code, and *Python API* refers to the python library that runs from an external computer and makes wireless connections to Siftables using Bluetooth. *Shake*, *tilt*, and *neighbors* refer to events that can be detected by the Siftable. *Acc* refers to the raw accelerometer data. *Drawing* refers to the vector drawing capabilities of the Siftable, and *bitmaps* are full images stored in the Siftable's flash memory. *Variables* are individual 16-bit variables also stored in flash. *Sound* and *screen* refer to the use of the speakers or display of an external computer.

Chapter 5

Implementation

This chapter provides a discussion of the implementation details of the Siftables platform. Included in the discussion is an overview of the hardware, the firmware of the on-board operating system, and the application programming interface (API) that allows a software program running on a computer to control a collection of Siftables over a wireless connection.

5.1 Hardware

This section describes the hardware architecture of the individual Siftable devices and charging modules. I iterated several times on the hardware design to achieve a high degree of reliability, support for the interaction goals, power management, and manufacturability. The following discussion will focus on the hardware design of the final iteration.

5.1.1 Siftable Devices

Each Siftable device is a self-contained, battery powered interactive manipulative measuring 44 x 44 x 16 mm. It features a color OLED display that can be set to 8 or 16 bits per pixel, two 8-bit microcontrollers, one three-axis accelerometer for inertial sensing, 8 megabytes of non-volatile flash memory, four side-facing infrared

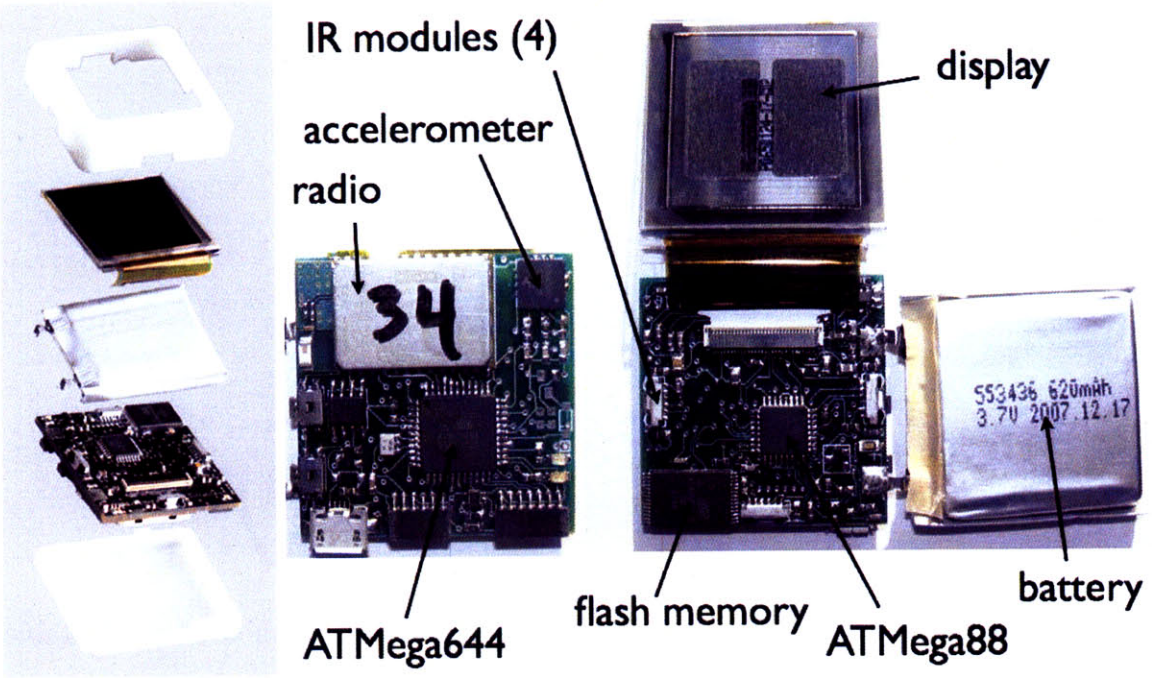


Figure 5-1: Internals of a Siftable. On the left is an exploded-parts diagram that shows (from top to bottom) the top case, display, battery, circuit board, and bottom case. The center picture shows the bottom of the circuit board in more detail, and on the right is the top of the circuit board, with display and battery attached.

communication modules (one pointed in each neighbor-facing direction), pushbuttons for device hard-reset and power toggle, a Bluetooth radio for data communication, separate pin headers for reprogramming each of the microcontrollers, a micro-USB connector for charging and power, and a rechargeable Lithium-Polymer battery. See figure 5-2 on the next page for an overview about how these parts are connected in the current system, and figure 5-1 for a diagram showing the parts in context.

Main processor

The behavior of each Siftable is primarily determined by an Atmel ATMega644 (AVR) microcontroller, clocked at 20MHz. The ATMega644 has an 8-bit reduced instruction set computer (RISC) architecture with 64 kilobytes of flash memory for program storage, 4 kilobytes (K) of static random access memory (SRAM), an 8-channel 10-bit analog-to-digital converter (ADC), serial peripheral interface (SPI) and universal asynchronous receiver-transmitter (UART) peripherals, and various other features

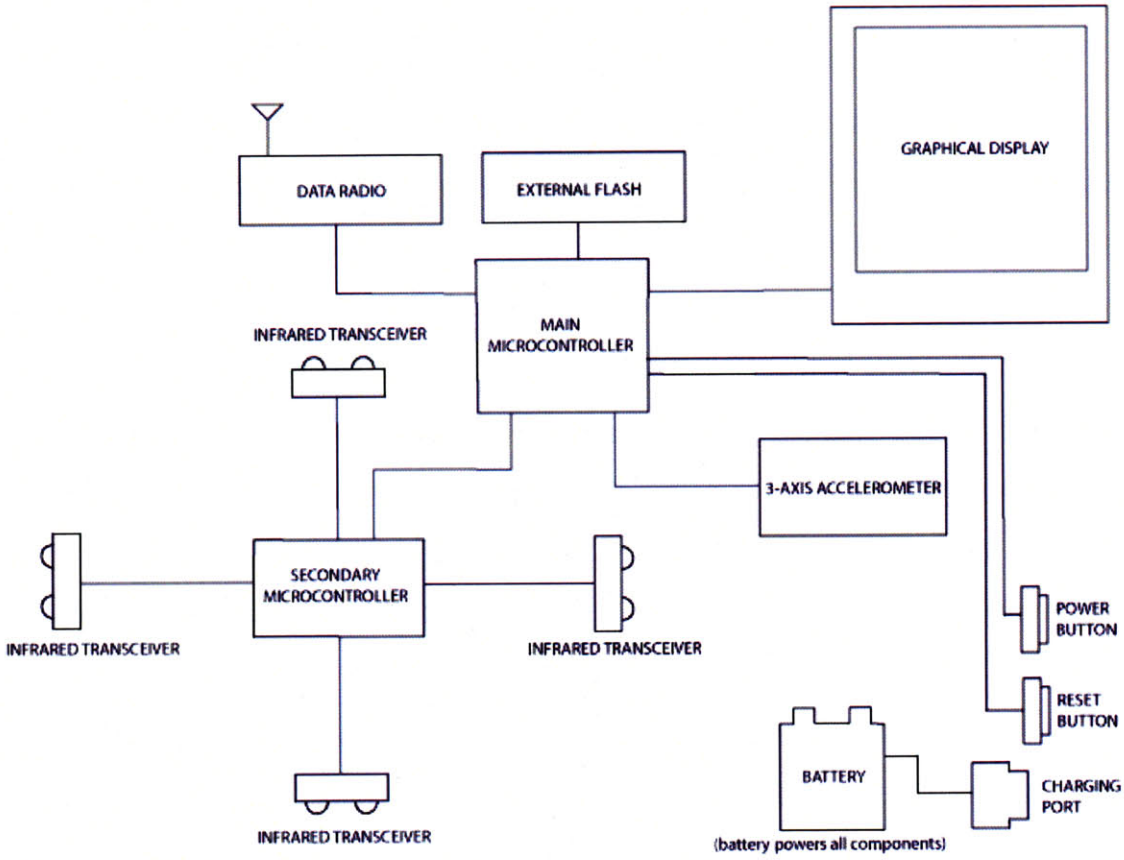


Figure 5-2: Block diagram of a single Siftable device

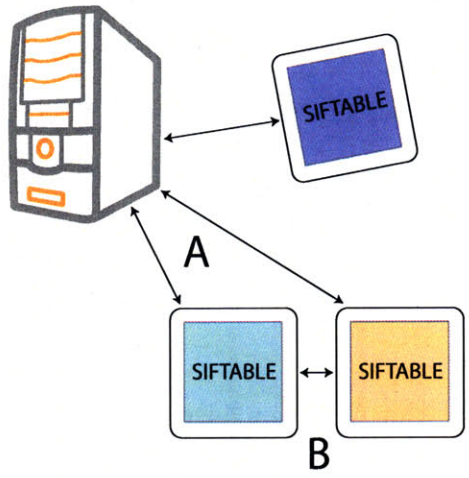


Figure 5-3: Siftables can communicate wirelessly over Bluetooth to a nearby computer (A). They use infrared communication for peer-to-peer detection of neighboring Siftables (B).

[5]. The duties of the main microcontroller include responding to commands from the Bluetooth radio over the UART, fetching neighbor information from the secondary microcontroller when it has news to report, transmitting event data over the Bluetooth radio, controlling the OLED display, reading and writing to flash memory, sampling the accelerometer and monitoring the status of the battery. A full flowchart of system behavior is shown in figure 5-6 on page 124

The AVR platform was chosen because it is a low-cost, capable microcontroller that has a strong engineering and hobbyist community that has produced online forums and example code for many common configurations. The development environment is mature and low-cost, with a free IDE from Atmel [5] that utilizes the popular open-source GNU toolchain (GCC) [31] for preprocessing, compiling and linking.

The ATMega644 has some notable limitations. For instance, the limited amount of SRAM (4K) is not sufficient to hold a single frame buffer for the 128x128 pixel screen, which would require 16,384 bytes even at its lowest bit-depth of 1 byte per pixel. The implication of this is that the graphics possibilities for Siftables are limited to displaying bitmap images that have been stored in flash or transmitted over the radio, and to using the drawing API provided by the display itself, which includes simple vector drawing commands, screen dimming and moving or copying chunks of pixels from one region of the screen to another. Another basic limitation of the AVR is that it is not capable of running a full operating system such as embedded Linux, meaning that higher-level capabilities such as implementing a file system on the flash memory chip had to be written from scratch rather than borrowed from a pre-existing open-source codebase.

Despite its limitations, the ATMega644 does an acceptable job driving the display with bitmap data and vector drawing commands, listening for neighbor data, accessing the flash memory, monitoring the battery level, sampling the accelerometer, and communicating with a remote computer over the Bluetooth radio. The range of interactive applications discussed in chapter 4 on page 85 that Siftables has proven to be a flexible platform.

Infrared Communication and Secondary Processor

Each Siftable has four short-range infrared modules. Each module is directed in a side-facing direction away from the Siftable through a rectangular hole in the body that permits communication with neighboring Siftables. A Siftable can detect the identity and orientation (i.e. which side of the neighbor is facing it) of other nearby Siftables up to a distance of 3-4 centimeters.

A simple infrared communication protocol is used to transmit identifying information between neighboring Siftables as a means of neighbor detection. Infrared communication in each direction proceeds in a round-robin pattern, with an attempt to initiate communication on each successive side every 2 msec. The transmitted data includes the numeric identifier of the transmitting Siftable, and the identifier specifying from which side of the Siftable the message originated. Although the infrared modules are capable of supporting the popular IrDA protocol [4] used by laptop computers and other personal digital devices, I have not implemented the IrDA specification or any arbitrary data transmission.

Infrared communication is performed by a secondary microcontroller, an ATMega88. When the secondary microcontroller has news to report such as the arrival or departure of a neighboring Siftable on a given side, it alerts the main microcontroller by changing the level on a line. Upon detecting the alert, the main microcontroller initiates a communication sequence using SPI to retrieve a message a data packet from the secondary microcontroller. This division of labor offloads the task of communication with neighbors to the secondary microcontroller, which was necessary due to the timing-sensitive nature of the infrared communication protocol.

The transmit range of each infrared module is limited to 3-4 centimeters by putting a 5K resistor in the current path to the transmitting LED. This limitation is deliberate, since in the tabletop usage scenario, neighbor detection of Siftables that are not immediately proximate to a given Siftable would likely be spurious and would make topology reconstruction difficult or impossible. The ability to sense the continuous distance between Siftables is a feature that was considered but not implemented.

Flash Memory

Each Siftable has a flash memory chip with 64 megabits (8MB) of data storage. The flash is accessed by a SPI interface, and is used primarily to store program variables and image data. A simple data organization scheme was implemented, described in chapter C on page 211. In summary, the flash is primarily organized into image-sized chunks. The first chunk is used for system and application variable bindings. Variable names can be up to 31 characters long, and values are 16 bits (unsigned). The second image-sized chunk is reserved for a memory map that tracks some state about how the rest of the flash is being utilized. This map was not implemented. The third chunk holds the Siftables logo, which is displayed for a few seconds after the device powers up.

The idea of using a section of the flash as a frame buffer for the display was discussed during the development cycle. However, since flash memory is slower to write than SRAM, and it has a limited number of writes per address before breaking down, this scheme was not implemented.

Accelerometer

Inertial sensing is accomplished by a 3-axis accelerometer made by Freescale [106] that uses micro electro-mechanical systems (MEMS) technology. The accelerometer has selectable sensitivity, meaning that its maximum reported values can be configured to reflect different levels of absolute acceleration ($\pm$1.5g/2g/4g/6g). This feature allows different applications to utilize the accelerometer flexibly, though most gesture-oriented applications created to date have utilized the most sensitive ($\pm$1.5g) setting. The signal from each axis is conditioned by a passive low pass filter before reaching the microcontroller to reduce spurious noise. The analog signals produced by the accelerometer are sampled at 100Hz by the main microcontroller's 10-bit Analog to Digital Converter (ADC), and the lower 2 bits are discarded. This sampling strategy results in a new 8-bit reading for each of the three axes every 10 msec.

One drawback of MEMS accelerometers is their fragility. Since a MEMS ac-

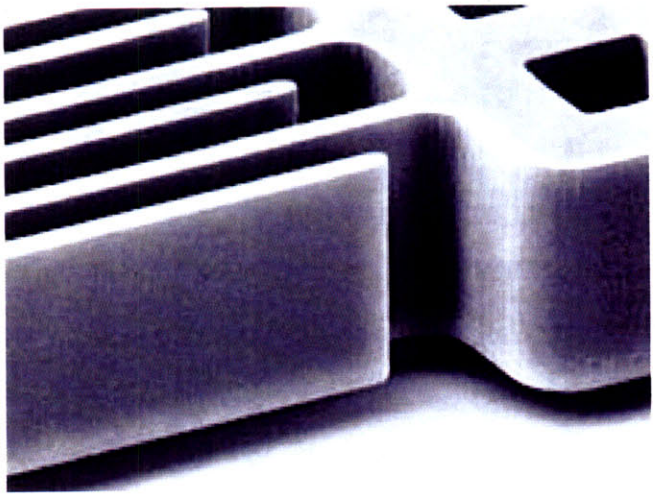


Figure 5-4: A close-up view of tiny cantilever beams inside a MEMS-based accelerometer, which can deform if the device experiences a sharp impact. Photo from [11].

celerometer utilizes tiny moving cantilever beams internally [11], a sharp impact to the device may deform these parts, damaging the functionality of the device. The accelerometer I used is drop-test rated to 1.8 meters onto concrete. Throughout this work I have only one accelerometer that I suspect was damaged due to impact trauma.

OLED Display

The graphical display of a Siftable is a 128x128 pixel Organic Light Emitting Diode (OLED) screen with a built-in controller circuit. The display is interfaced to the main microcontroller using SPI. OLED display technology is newer than LCD technology and thus the OLED display used is more expensive than comparable LCDs found in mobile phones.

OLED displays differ from LCDs in two ways that are important to Siftables. First, the visibility of an OLED display does not degrade with viewing angle as with a LCD. This feature is advantageous for tabletop use, where a collection of Siftables can be spread across the surface in such a way that the user's viewing angle on some Siftables is quite low. With LCDs, a low viewing angle can result in difficulty viewing the displayed graphics (an inversion of the colors, or total loss of visibility can result), however this is not the case with an OLED display. Second,

the OLED display contains separate light-emitting diodes for each pixel, and the brightness and color results from the relative amount of current being allowed to flow through each of a red, green, and blue LED [1]. This means that the power usage of the display varies depending on the characteristics of the image being shown. A pixel that is displaying white at maximum intensity will consume far more power than one displaying black, since to display black, a pixel just turns off its LEDs. For pixels being driven with colors between white and black, power usage will vary depending on the relative brightness of the component colors. Thus the design of the graphics that are displayed on a Siftable impacts its power consumption. Graphics featuring darker colors, especially ones that avoid the use of white pixels, will reduce the current consumption of the display, extending the battery life of the Siftable. The display can also be placed into a low-power sleep mode when it is not in use.

Battery and Power

Each Siftable contains a rechargeable Lithium-Polymer (LiPo) battery, with built-in over-current and depletion protection circuitry. The capacity of the battery is 620 Milliamp Hours (mAh). The amount of time that a Siftable can run before needing to be charged depends greatly on the application behavior, primarily on how the OLED display is being used, and varies in practice from 4-10 hours (see section 5.1.1 on the preceding page for details on power usage of the display). Testing with mostly-white images being displayed on Siftables' screens found a worst-case performance of 4 hours until shutdown, whereas testing with mostly black images achieved 10-12 hours of run-time. No optimization of the microcontrollers' programs were attempted for either test, meaning that careful use of low-power sleep states could potentially push the best-case run-time to even longer durations.

A voltage regulator converts the voltage of the battery to a stable 3.3 volts, the level that most of the on-board electronics require. The output of the voltage regulator

[1]The amount of current allotted to each pixel may be modulated by pulse-width modulation, a scheme for rapidly driving a signal high-to-low periodically and varying the percentage of time that the signal spends in each state (i.e. varying the duty-cycle). Though this method only features two states, completely on or completely off, the aggregate result to an LED being driven of different duty-cycles is a change in brightness

can be enabled or disabled by the main microcontroller, and the microcontroller must always keep the regulator in "on" mode in order for the Siftable to remain powered on. If the microcontroller fails to hold the regulator's output enable (OE) line in *enabled* state, the device will power off immediately. This allows the microcontroller to deliberately power off the Siftable, for example when the battery gets low.

There are two buttons on the side of the Siftable. Depressing the button closer to the center of the edge puts the main microcontroller into RESET mode, which causes all pins to float. The result is that the main microcontroller can no longer assert the voltage regulator's OE line, and the device will immediately shut down. The button closer to the corner is a *soft power toggle* button. When the device is turned off, depressing this button manually asserts the voltage regulator's OE line, powering up the system. Directly after power-on, the main microcontroller asserts the OE line, keeping the device powered up. When the Siftable is turned on, depressing this button triggers an interrupt on the main microcontroller. In servicing this interrupt the Siftable waits until the button is released, then initiates the power-down sequence which terminates in releasing the OE line, causing the powering down of the device.

5.1.2 Charging Dongle

The battery is charged through a micro-USB socket on the Siftable's circuit board. A cable inserted into the socket connects the Siftable to a custom charging dongle, which plugs directly into a computer's USB port or into a USB hub to draw power. When a Siftable is connected to the charging dongle, the dongle provides power to the charging circuit, and also directly powers the Siftable. During charging, the battery is electrically disconnected from the rest of the Siftable's circuit by a MOSFET. See the schematic in chapter B on page 199 for more detailed information.

5.2 Software

This section covers the software implementation, including the firmware that runs on the microcontrollers on the Siftables, the ASCII language specification for basic

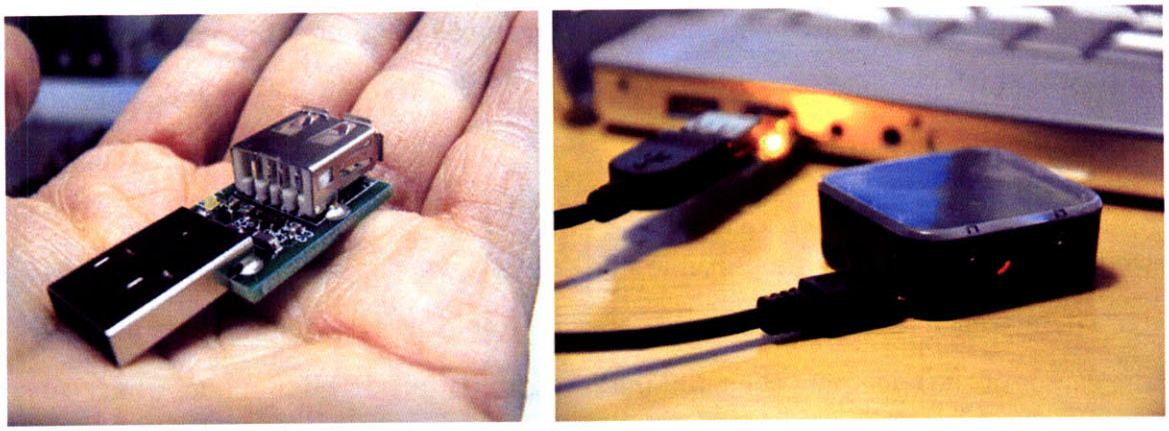


Figure 5-5: The charging dongle (left). The USB plug on the left side of the dongle is inserted into a USB port on a computer or hub, and a micro-USB cable is inserted into the socket on the right. On the right is a Siftable being charged. The LED on the dongle indicates charging.

remote control of a Siftable, and the Python API that exposes Siftables access at a higher level to provide a generic and cross-platform way for a software application to interact with Siftables.

There are two options for how the behavior of an individual Siftable can be controlled. A Siftable can be operated by a program that is installed directly in its firmware, or by a software program running remotely on a computer that communicates with the Siftable wirelessly using its Bluetooth radio. These two models for application development represent distinct options for developers, and a given application may rely exclusively on one or the other, or may utilize both local and remote code.

5.2.1 Firmware

The firmware of a Siftable comprises the basic operating system that provides data services and monitors the device's state, and optionally a developer-created application that can access this state and respond to OS-generated events.

Basic Operating System

A user powers on a Siftable (figure 5-6 on the following page) with a button-press. While powered on, the OS on the main microcontroller is responsible for periodically monitoring the battery status and sampling the accelerometer, communicating with the secondary microcontroller when it is alerted about new neighbor information, drawing graphics on the display and responding to other incoming commands over the radio. If the battery level drops beneath a threshold or the "power off" command is received over the radio, the main processor shuts the system power off, halting the Siftable.

Analog values from each axis of the three-axis accelerometer are sampled at 100Hz and the most significant eight bits are stored. The data is processed, and the results are optionally reported over the radio. Each incoming frame of raw accelerometer values is added to a buffer of previous values. From this buffer, higher-level parameters are computed, including the per-axis windowed mean and activity level, tilt, and shaking state. If software running remotely has "subscribed" to the raw accelerometer or activity level data, these values will be transmitted over the radio at a rate of 100Hz. The current tilt and shake values are compared to the previously-measured values from the last analysis cycle, and if the current values have crossed a programmer-defined threshold compared to the previous values and remote software has subscribed to *events* for either of these, the updated state is transmitted over the radio. Hysteresis is implemented to prevent "jitter" that could result from activity or tilt levels near the threshold.

A secondary processor figure 5.2.1 on page 126 is responsible for communicating with nearby Siftables using short-range, directional infrared communication (see section 5.1.1 on page 117 for hardware information). A Siftable that is close enough to be in infrared communication range is considered a "neighbor". Neighbors can be sensed in each of the four side-facing directions surrounding a Siftable. Transmitting and listening behavior may be turned on or off by the main processor. To transmit, the Siftable periodically "pings" an infrared pulse in each direction, and if a reply

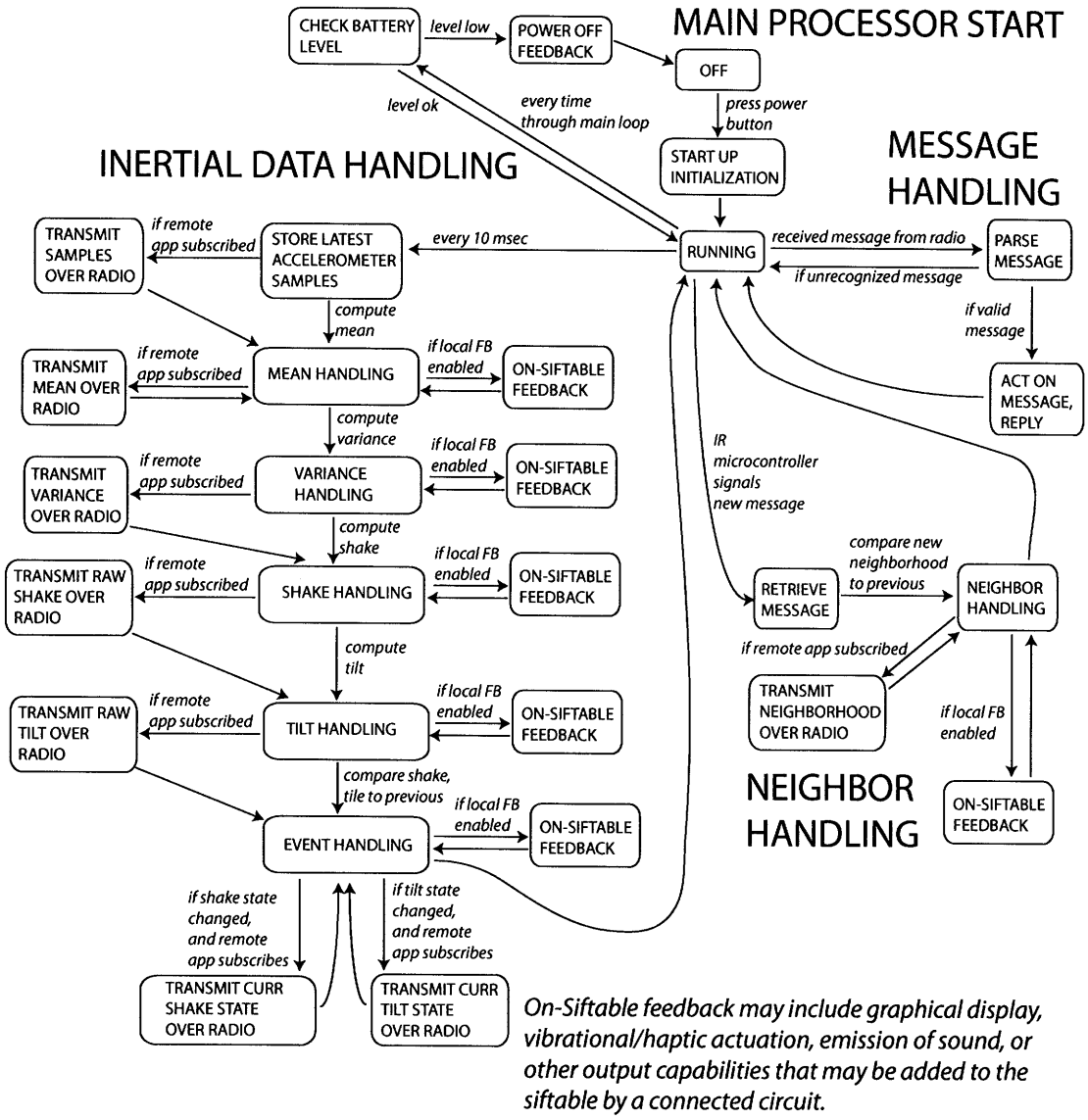
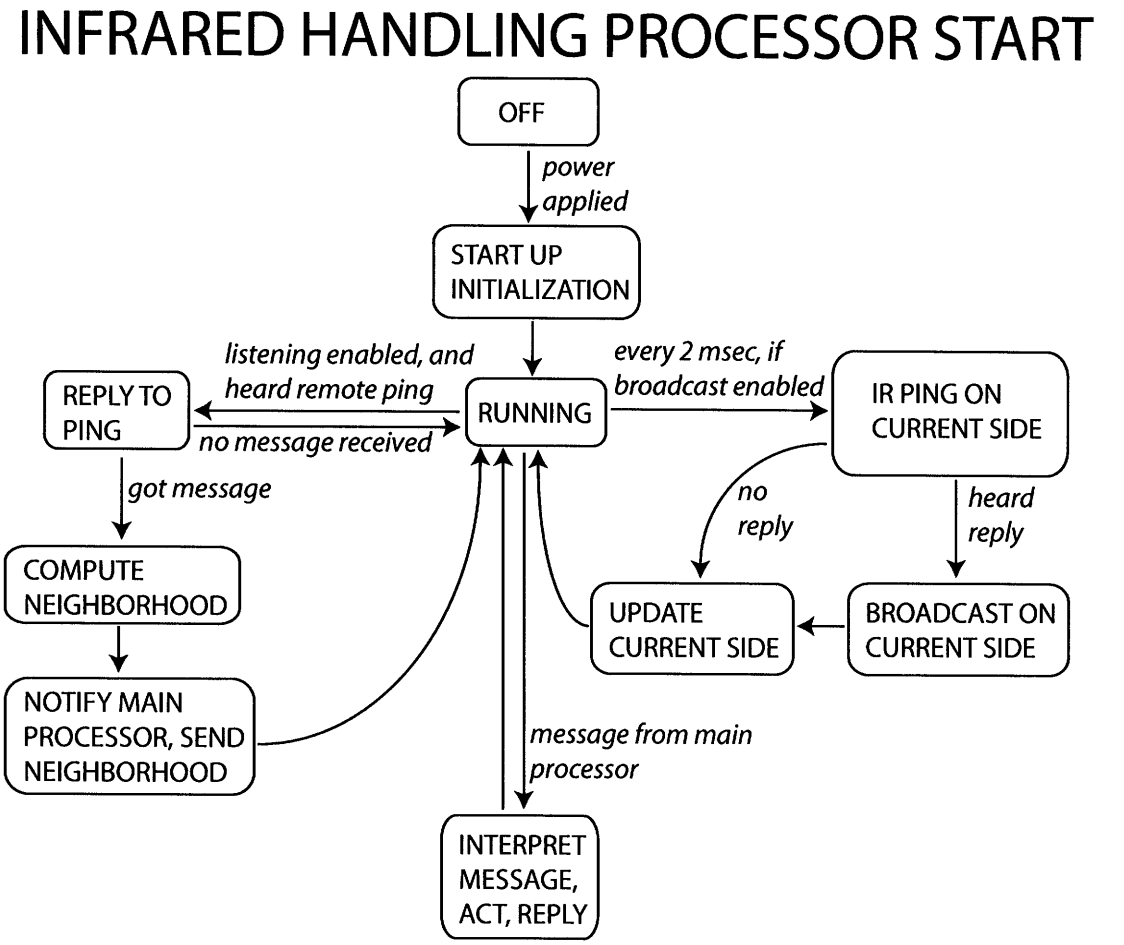


Figure 5-6: Operation flowchart for a single Siftable (primary processor)

"ping" from a neighboring Siftable is received, it transmits a message to the neighbor, communicating the Siftable's ID and from which side the message emanated. If a new neighbor arrives on a side, the stored representation of the current neighborhood is updated to reflect this addition and the updated state is immediately communicated to the main processor. In order to reduce "jitter" in the form of spurious arrival or departure messages to the main processor due to infrared message collisions or intermittent failures in infrared communication, a departed neighbor must not be detected for 0.5 seconds before it is considered gone and its departure is communicated to the main processor. By this policy, new neighbor arrivals are communicated immediately, and departures take slightly longer to be confirmed and communicated. The period between infrared messaging attempts on a given side is 8 msec. The frequency of communication results in a polling frequency that is high enough so that to an end user both arrivals and departures seem nearly immediate to a user.

The main processor drives the color screen. One way to draw graphics is to load bitmap images from the flash memory. Images can be loaded at a rate of up to 30Hz, which is faster than necessary to create smoothly moving animations. The screen can also respond to vector drawing commands such as line, rectangle, and circle. The main processor can put the screen into a low-power sleep state, can adjust the brightness and contrast, and can control a number of other display parameters.

Each Siftable has a 64 megabit (8 megabyte) flash memory module separate from the microcontrollers. This memory can be written and read by the primary microcontroller, initiated either directly by a program running on the microcontroller, or as a result of communication over the radio from a remote software program. Arbitrary data can be stored in this memory, such as images for display on the screen, variable names and associated values, samples from the accelerometer, or other values that an application requires. The operating system on the Siftable provides high-level functions allowing the main microprocessor to retrieve a sequence of images stored in this memory and display them sequentially on the screen, creating animations or movies.



Messages from the main processor to the infrared processor may be to enable or disable infrared listening or broadcast behavior, to update broadcast information such as the siftable's numerical ID or broadcast period, to query information from the infrared processor, or to command the infrared processor to perform some other duty, such as user-directed feedback that it may be configured to perform.

Figure 5-7: Operation flowchart for a single Siftable (secondary processor)

C API for In-Firmware Applications

The Siftable OS includes an event-driven handler system in the C API that provides a modular approach to adding behavior. To build an application, a developer does not need to replace all the code running on the Siftable; they can safely ignore most of it. Rather, they only need to edit one file where event-handler functions are defined and initialized. The C API includes a template for this file that developers can edit, using their own functions or functions from the rest of the C API.

The basic model for creating a firmware-based application is as follows. First, the developer writes an event-handling function and an initialization function for their desired behavior in *siftables-user-application-template.c*. Then, they modify the *InitUserApplication* function in the same file to call their initialization function and to install their event-handling function by passing it as a function pointer to the appropriate handler-installer function. Finally, they can turn on handler dispatch behavior by changing the flag for the given event type, also within *InitUserApplication*. This last step puts the OS into a state where it will call the handling function whenever the given condition arises. See figure 5-8 on the next page for a usage example of the C API.

5.2.2 ASCII Language

Radio communication with a Siftable utilizes the Bluetooth radio. This can take the form of a serial-over-Bluetooth link in which the Bluetooth connection appears like a serial port to the operating system (Windows, MAC), or it can use the more low-level RFCOMM API available in the *PyBluez* Python module (Windows, Linux). Communication using the ASCII language uses a human-readable protocol that can be typed interactively from a computer keyboard. The use of ASCII commands results in a language that is not as compact as it would be if it used binary *opcodes* and values, but the ability to type characters from a keyboard and view the results immediately sometimes makes it easier to interact with a Siftable during program development and debugging. A few commands do require binary data, such as image

```
// siftables-user-application-template.c

// this variable tracks of the current image index
uint16_t img_idx;

// a user-defined init function for any necessary initialization
void init_AdvanceImageWhenShaken( void )
{
    img_idx = 3;
    DisplayImage(img_idx);
}

// a user-defined handler function for shake events
void AdvanceImageWhenShaken(uint8_t *shake_state)
{
    if (shake_state[X_AXIS] == SHAKING) {
        img_idx++;
        DisplayImage(img_idx);
    }
}

// in InitUserApplication, the programmer inits the user-defined
// variables, installs the handler, then enables callback behavior
void InitUserApplication( void )
{
    init_AdvanceImageWhenShaken();
    setNeighborEventsHandlerFn(AdvanceImageWhenShaken);
    EnableAccelShakeEventsHandler();
}
```

Figure 5-8: C API usage example: Shake event handling. In this example, a handler function for shake events is installed. Each time the Siftable is shaken, the AdvanceImageWhenShaken function will be called. If the Shake was along the Siftable's X axis, the index of the currently displayed image will be incremented and the current image displayed on the screen.

```
# import the library
from Siftable import *

# allocate a Siftable object (makes the connection)
sift = Siftable.Siftable(bt_name='Siftable-v4-027')

# create a simple callback function that just prints
def handle_tilt(event):
  print "data: " + str(event.data)

# register the callback function
sift.install_listener_tilt_events(handle_tilt)

# enable tilt events
sift.acc_events_tilt(True)
```

Figure 5-9: Python API usage example: Tilt event handling. This example code will make a Bluetooth connection to a Siftable, then create and install a tilt-handling function. Finally, the callback behavior is enabled, so that the tilt-handling function will be called on tilt events.

uploading and miscellaneous debugging functions.

The ASCII language is a layer beneath the more often-used Python API, but it can be useful when debugging the Python API itself. In most cases it remains an intermediate building block, not directly used by the programmer. The next section will discuss the Python API.

5.2.3 Python API for Remote-Control Application Development

The Python API is the most high-level way to develop applications for Siftables. It is a library that provides high-level object-based access to the entire ASCII Language for interacting with Siftables.

Motivation for the Python API

The motivation to create the Python API came from difficulties encountered during my early experiences programming Siftables with ASCII Language. The fundamental problem was due to the asynchronous nature of socket communication, and the most obvious drawback was the following: For each ASCII command a Siftable receives, it generates an ASCII reply. Therefore, the simplest way for a program to manage communication with a Siftable was to send a command, then block, reading bytes from the communication channel until a full reply arrived, and to assume that the received reply was in response to the command. As long as there was a one-to-one correspondence between messages sent to the Siftable and messages received from the Siftable, this strategy worked: outgoing and incoming messages remained in sync from the remote software's point of view. However, Siftables can be put into event-reporting modes (reporting raw accelerometer readings, variance, tilt events, neighbor events, etc..) in which they generate messages that are not in response to a direct query from the software, meaning that the number of outgoing messages generated by a Siftable will be greater than the number of incoming messages received. In practice this resulted in rapid misalignment of messages on the computer end, since it was difficult to track which incoming message was a reply to a particular outgoing message.

Another problem that complicated Siftable programming before the development of the Python API was that the exact text string of any given command had to be explicitly formulated by the application. Correspondingly, any reply from the Siftable had to be parsed by the Python application. For instance, in order to query the current accelerometer calibration values, the command from Python (pre-API) would look like this:

```
sock.write("acc curr calib\r\n")
result = siftutil.recv(sock) # collects incoming characters until the \r\n
[x,y,z] = map(int, result.split("calib")[1].strip().split())
```

This model made it easy for programmers to make mistakes such as misspelling a command string. Python development environments are not able to detect this class of mistake since it is the command string (not the method name) that is misspelled, but at run-time the Siftable would not parse the result successfully. Moreover, the task of parsing the reply string was also left to the programmer, introducing more opportunities for errors. The Python API, described in the next section, converts the aforementioned command to the following syntax:

```
[x,y,z] = sift.acc_curr_calib()
```

The Fix: Overview of the Python API

The solution to the message misalignment problem was to change the mechanism for sending commands to the Siftable. Rather than directly pushing an ASCII command into a socket and waiting for the reply to appear in return, the Python API allows a program to invoke a single function call to transmit any Siftable-directed message and receive the reply as a return value. The library encapsulates the sending, receiving, and bookkeeping of messages, creating a convenience layer of function call access to control Siftable behavior.

The key to bookkeeping outgoing and incoming messages between software and a Siftable over the Bluetooth channel was actually quite simple. I created a message-numbering scheme that allows incoming messages from the Siftable to be matched to the outgoing message that triggered them. On initialization, a Python Siftable object creates a separate thread that blocks on incoming data from the Siftable. Using a thread allows the Siftable object to remain responsive to user input even if it is waiting to hear back from the remote Siftable. Messages sent to the Siftable are prefixed with a numeric ID. Each message that comes back from the Siftable is prefaced with the ID of the request that triggered it, allowing the Siftable object to match incoming messages with outgoing requests. Event messages that are not the result of a request have no such prefix, and are handled by a callback mechanism, described in the next paragraph.

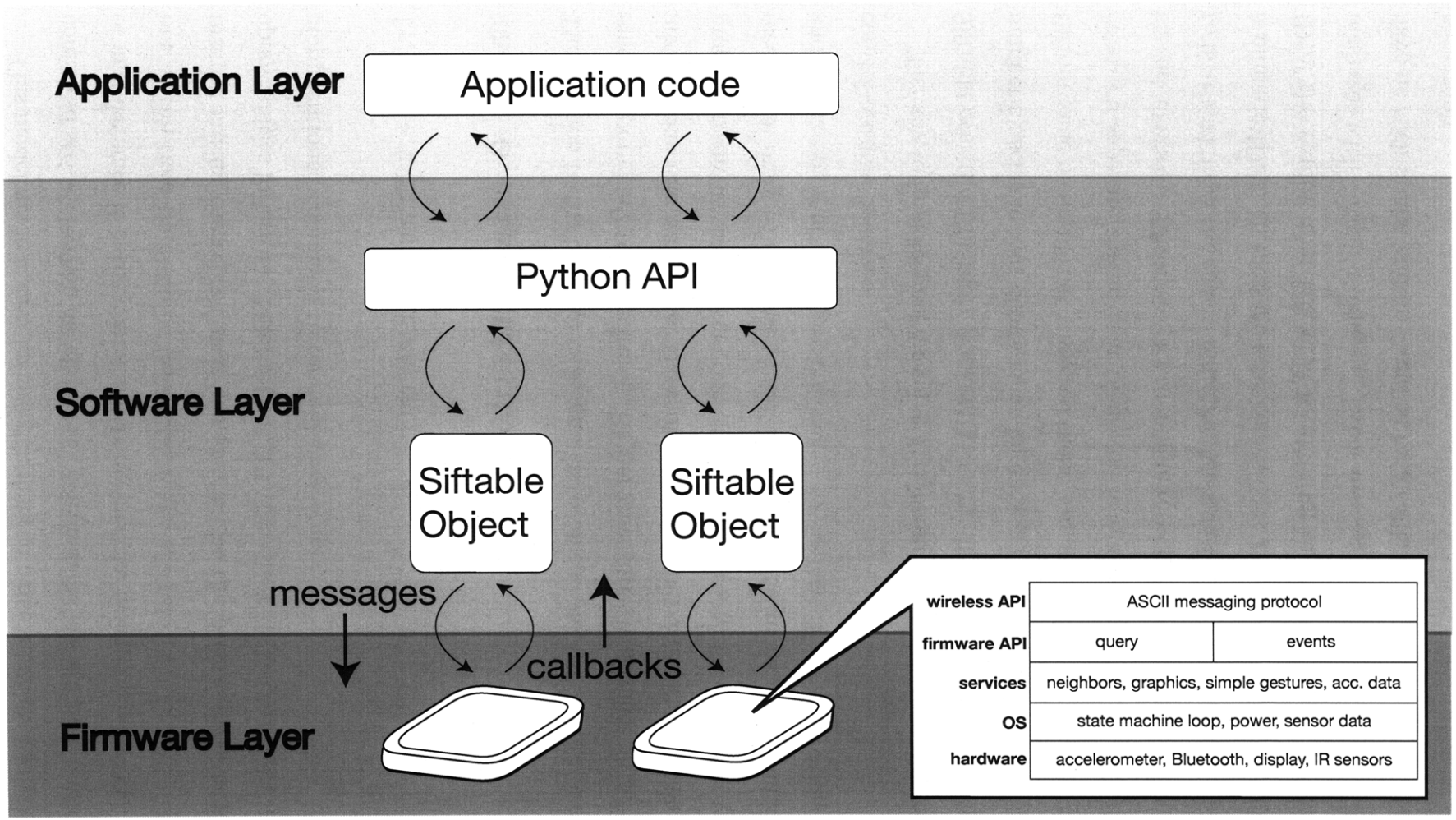


Figure 5-10: This diagram shows the layers of the software and firmware that permit programming Siftables. The callout on the right contains the on-siftable firmware layers, and on the left (Software Layer) is the Python API.

```
while True:

  # wait for something to happen (neighbor event, shake, etc..)
  this_event = wait_for_event()

  # update our model of the game state and respond to the new state
  current_state = process_event(this_event)

  # optional program response to new current_state
```

Figure 5-11: Python application template main loop. The programmer implements their desired application behavior in the *process_event* function, and optionally in code inserted into the main loop directly after the call to *process_event.*

Asynchronous messages from the Siftable can be generated in response to events such as neighbors arriving or departing, tilting, shaking, or new accelerometer or variance data. The Python API defines a callback mechanism to allow programs to handle these events. The developer writes a function to receive the event, registers the function as a listener for the given event type, then enables reporting of that event type. Whenever a message of the given event type arrives from the Siftable, the supplied event handler callback function will be invoked, with the event passed as its argument. See figure 5-9 on page 129 for an example, and Appendix chapter A on page 183 for a full API listing.

Python Application Template

I created an application template based on the Scraboggle application that handles many common tasks for a Python-based Siftables application. These tasks include connecting to a set of Siftables, instantiating data structures to keep application-specific data associated to each Siftable object, setting screen brightness to the maximum level, and entering a main loop where events are handled (see figure 5-11). This template was the starting point for most applications created after Scraboggle.

Font Library

Rick Mancuso developed a fixed-width, single-size font library. The library is written entirely in Python, and it supports drawing characters onto the Siftable's screen. The characters supported are the 26 letters of the alphabet, the digits from 0-9, and assorted punctuation. The library can write up to seven characters across each line, and up to four lines of text onto a Siftable's screen. The font library has been subsequently used during debugging, and in applications such as the music sequencer (section 4.3 on page 88).

Chapter 6

Evaluation

This chapter describes the user studies that I carried out using Siftables to better understand the possibilities and limitations of embodied media. I collected both quantitative and qualitative data, in an effort to adequately characterize these qualities.

6.1 Evaluating Novel User Interface Systems

Novel user interface systems can be difficult to evaluate quantitatively. The basic assumptions underlying formal comparative user studies are that the utility of a new system *can be measured quantitatively*, and that a useful way to evaluate its utility is to compare its performance to *another similar system* along some measurable axis.

The problem with these assumptions is that it is not always clear that an existing system is an appropriate comparison to a new user interface, nor that the measurable performance metric is a particularly informative indicator of the utility of the new interface. The new system may provide affordances that are simply *different* than existing systems, in which case gaining an understanding of user performance along an axis that is relevant to the new system may be the best way to quantify its advantages and limitations. Several recent papers in the human-computer interaction community attest to a growing acknowledgement of this problem [64] [37]. Furthermore, differences that are numerically measurable may not be the most important

factors in the user experience, so it is valuable to supplement quantitative feedback with qualitative feedback [20]. In light of these thoughts, the evaluation of Siftables both quantitatively and qualitatively in two separate studies was to provide a more comprehensive analysis of the possibilities and limitations. My decision to compare Siftables-based interaction against the mouse/GUI is because the WIMP desktop is the most widely-used interface today.

Although comparative evaluations can be problematic, it is important to understand the *efficiency* that a new user interface permits users to achieve. The literature on pointer-based systems (i.e. mouse, touchpad, touchscreen) for example, is replete with studies of target-acquisition and dragging-times that reference Fitt's Law, a measure that relates the size of the target and distance that the cursor must travel to these values. This measure has proven to be quite useful for understanding the efficiency of pointer-based interfaces. The cognitive science literature reviewed in section 2.2.1 on page 28 suggests that allowing the solution space to be explored more efficiently by the user enables them to achieve more and better results. The content-organization task, which I describe in the next section, measured the efficiency of individuals and pairs in a Siftables-based interaction.

6.2 Ordering and Grouping Study

It is known from the distributed cognition literature discussed in section 2.2.1 on page 28 that the efficiency of manipulating a problem representation that an interface permits can impact the approach that users take, and the resulting quantity and quality of their results. I evaluated the efficiency characteristics of Siftables for sorting and categorization activities because in everyday computer usage we often group and order digital content items, for instance when we create folders and sort files into them, make slides for a presentation, or sequence a video from individual clips.

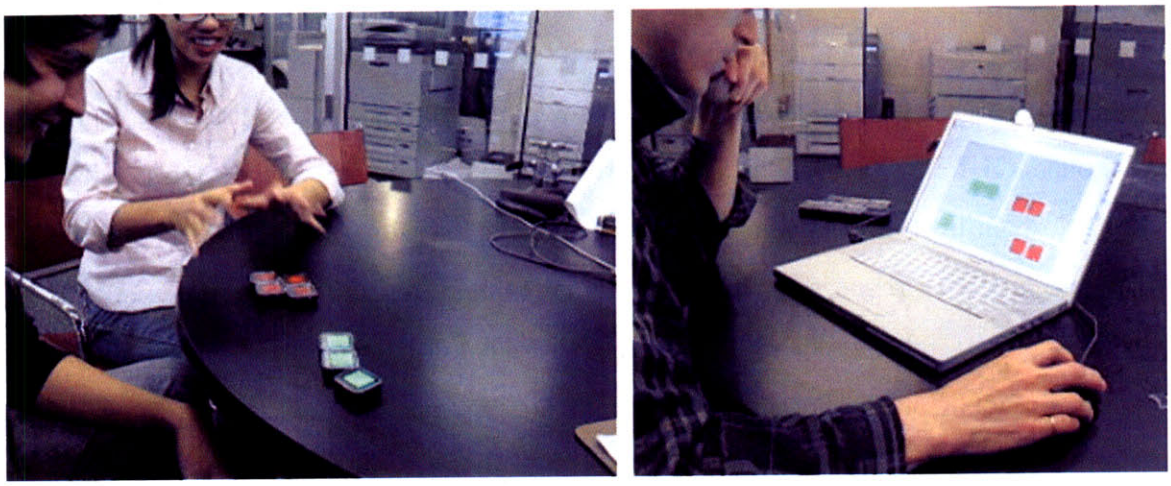


Figure 6-1: Two participants in the content-organization study have just finished separating a set of Siftables into groups based on the color shown on their screens (left). A solo participant uses the mouse to separate on-screen icons into two groups based on the color (right).

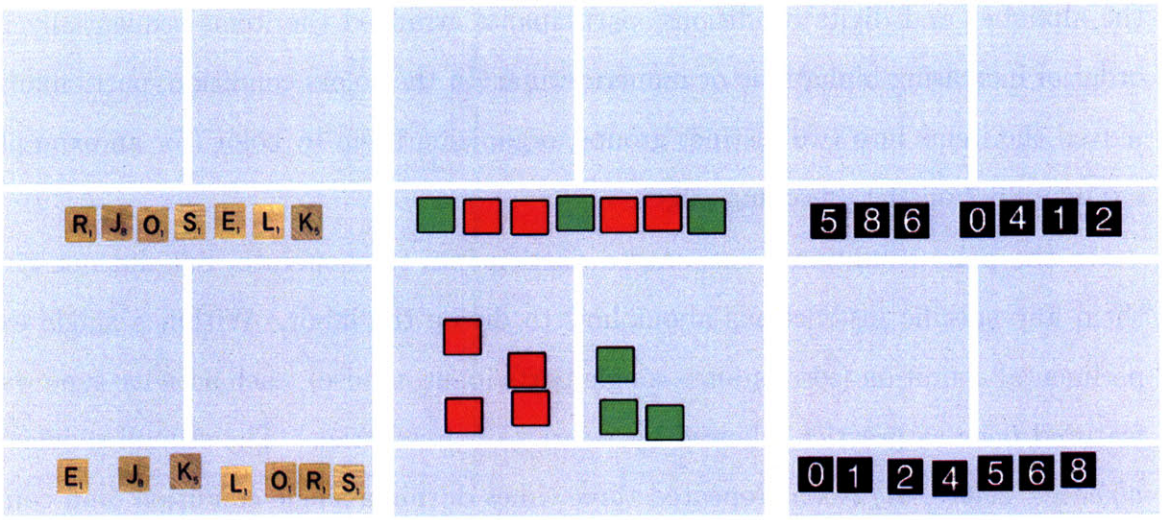


Figure 6-2: Typical *before* and *after* states for the three content types in the GUI version of the content-organization task. The content types were (from left to right) alphabet characters, colors, and digits. For each content type, the top image shows the configuration before arrangement, and the bottom image shows the configuration after.

6.2.1 Method

I conducted a grouping and ordering experiment with a $2X2$ design. The experimental conditions were pairs of participants versus single participants, and Siftables versus a mouse/GUI interface. The study involved a total of 18 participants, with 6 in the solo condition and 6 pairs of 2 in the pairs condition. Participation was within-participants with respect to the interface used, meaning that each participant or pair of participants interacted with both the Siftables and the mouse/GUI experimental conditions. The order of condition presentation between the mouse/GUI experimental conditions was randomized.

I created a pair of corresponding applications, one using Siftables, and one using the mouse/GUI, that required participants to linearly arrange and spatially group individual items. In the Siftables condition, participants manipulated Siftables showing images on their screens. In the GUI condition, participants used the mouse to drag and drop on-screen icons. The applications utilized three types of content: alphabetic characters, numeric digits between 0-9, and colors (red and green). In the alphabet and digits conditions, participants arranged the items sequentially in order of increasing alphabetic or numeric value. In the colors condition, participants moved the items into two distinct groups, organizing them by color (for an example see figure 6-2 on the preceding page).

In the pairs condition I encouraged participants to cooperate, but did not give them any specific instructions about how to divide the labor. Within a single experimental condition (for instance, single-participant, mouse), each activity type was featured once as practice followed by a series of timed trials. The stimuli sequence *alphabet, colors, digits* was repeated three times during a single condition, and completion time for each trial was recorded.

6.2.2 Results

The results of this study can be summarized as follows: Participants completed the task more quickly using Siftables than with the mouse for all activity types and in

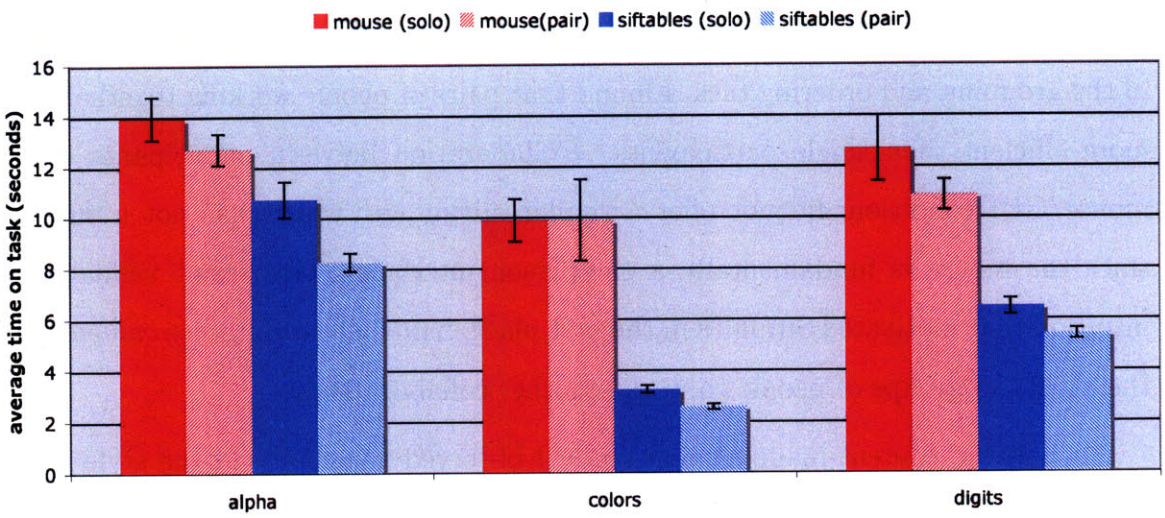


Figure 6-3: Completion time results from the content organization study. Both Siftables conditions (solo, pairs) have lower average completion time than both mouse/GUI conditions. Pairs in the Siftables condition had lower average completion time than solo participants. In the mouse/GUI condition however, pairs versus solo participants did not have significantly different averages.

both solo and pair conditions. Furthermore, using Siftables pairs were significantly faster than solo participants. Using the mouse, pairs and solo were not significantly different. See figure 6-3 for more information.

6.2.3 Discussion

The faster task completion time in the Siftables conditions of the grouping and ordering task indicate that independent physical manipulatives such as Siftables offer an efficiency advantage over mouse/GUI-based manipulation of on-screen icons for linear ordering and rough grouping activities. This efficiency advantage would be shared by any system of physical manipulatives such as a number of tabletop systems that exist today, but Siftables' mobility opens up possibilities for use in a wider variety of locations and activity contexts. This result suggests that Siftables can permit more effective and flexible problem-solving than a mouse/GUI system for activities involving ordering and grouping, because greater efficiency will allow users to explore the solution space more effectively, finding more and better solutions.

In addition to the performance advantage offered by Siftables over the mouse/GUI in the grouping and ordering task, I found that pairs of people working together were more efficient than single participants. Collaboration between participants in the mouse/GUI condition did not offer a similar advantage, which was not a surprise since the mouse is fundamentally a one-person interface. This result validates an intuition that motivated Siftables in the first place, namely that Siftables can enhance the ability of groups of people to work together collaboratively.

During the content-manipulation study I observed variability in the style of co-operation between pairs when using the Siftables interface. Some pairs talked to each other to determine strategies for efficient action, while others did not. A typical strategy for the linear ordering activities (alphabet, digits) was for one participant to collect the tiles from the lower half of the sequence, while the other participant collected the higher ones. Since the pairs were seated next to each other, this strategy allowed them to "divide and conquer", each participant arranging half of the solution before they cooperatively placed the two sequences next to each other. In the color-grouping activity, a typical strategy was for one participant to collect Siftables showing red and the other collect Siftables showing green. I also observed variability in the number of hands used by participants, noting that often one participant would use two hands while the other used only one. These observations are anecdotal, but they suggest that a system of independent physical manipulatives like Siftables can be effectively used by more than one person. Further observation could provide more detailed information about how pairs collaborate with an interface like Siftables.

6.3 Image Manipulation Study

To measure the effectiveness of Siftables as a tool for both sequential arrangement and fine adjustment of information or control items, I created an application that allows users to manipulate digital images by specifying the inclusion, ordering and magnitude of image-processing effects such as Blur, Brightness and Hue.

Figure 6-4: An original image (left), the original image with effects Threshold then Blur applied (middle), and the original image with effects Blur then Threshold applied (right). The order of the effects makes a difference to the end result.

6.3.1 Description of the Application

For a full description of the image manipulation application, see section 4.11 on page 101. The conceptual model of the image manipulation application is a signal chain, wherein a source image is processed by a sequence of filters that can be engaged and adjusted by the user. The filtering operates in an accumulative manner, meaning that the result of the first effect is fed as input to the second effect, and so on.

Five effects are included in the application: Blur, Saturation, Threshold, Brightness and Hue. I chose these particular effects because each produces a result that is visually salient and quite different from the others. The order of certain effect combinations can make a perceptible difference; for instance Blur before Threshold looks quite different than Threshold before Blur (see figure 6-4 for an example). Threshold before or after Saturation or Hue renders all but the Threshold effect imperceptible.

6.3.2 Method

Each subject was presented with a sequence of image pairs shown side-by-side on the 15-inch display of a laptop computer. The participant manipulated the right-hand "result" image to look as similar as possible to the left-hand "reference" image. The "reference" image of each pair was pre-processed by a sequence of $1-3$ effects, and the

"result" image was processed in real-time by the effects that the participant engaged and adjusted (see figure 6-5 on the facing page).

The study featured two experimental conditions, Siftables versus mouse/GUI. The experimental design was within-participants, and the order of the two conditions was randomized across participants. Within each experimental condition, I first presented participants with a single "practice" image pair, followed by five "real" image pairs. The participant was told that they could spend as much time as they wanted exploring the practice pair to get used to the system, then during the real pairs they should be as expedient as possible. I instructed them to adjust the image to be similar to the pre-processed image, to their satisfaction. When ready to proceed, participants used the mouse to click an on-screen button labeled "DONE" to advance to the next image pair. Completion time was measured for each image pair.

A survey after each condition asked participants to rate their agreement with statements about the system on a 7-point Likert scale. At the end of the study participants answered a series of comparative questions in which they reported which system they preferred or felt was superior in various ways or for various types of activities.

Condition A: Mouse/GUI

In the mouse condition (see figure 6-5 on the next page), participants used a mouse to drag effect icons in and out of an on-screen representation of the active effect chain. The effect chain was represented by a sequence of gray boxes located just underneath the pair of images, beginning with an image thumbnail showing the picture being manipulated. When an effect was dragged and dropped onto one of the gray boxes, it would "snap" into place on top of the box. When an effect was dragged and dropped anywhere else in the GUI, it would "snap" back to its original home location. Each effect icon had a linear slider situated horizontally across its bottom edge. The participants could drag the slider left and right to adjust the magnitude of the effect. These sliders were initially set to the effect's neutral point (see figure 6-5 on the facing page), and the effect icons were initially placed *out* of the effect chain.

Figure 6-5: The mouse/GUI version of the *image manipulation* application. On the left, a participant interacts with the mouse/GUI version of the image manipulation application. On the right is a screen capture in which Hue and Brightness are engaged. The Hue of the "result" image (right) has been adjusted to match the "reference" image, but Brightness has not.

Condition B: Siftables

In the Siftables condition (see figure 6-6 on the next page), participants manipulated Siftables to specify the order and to adjust the magnitude of the effects. One Siftable displayed a thumbnail image of the unmodified source picture, and each image-processing effect was represented by a separate Siftable. The effect Siftables showed either a text label of the effect's name or a live preview of the effect, or both (see the descriptions of the pilot and full study for details). Participants could insert an effect into the active sequence by placing the associated Siftable adjacent to the thumbnail Siftable. Any number of the available effects could be sequenced in this manner, and the effects accumulated in a left-to-right manner. To remove an effect from the active sequence, the participant separated it from the sequence of Siftables adjacent to the thumbnail Siftable. To adjust the magnitude of an effect, participants tilted the Siftable to the left or right. When an effect Siftable was tilted past a threshold (15 degrees from horizontal), it began to periodically increment or decrement the effect magnitude until the tilt returned to horizontal. See the descriptions of the Pilot and Full studies for differences in effect increment and decrement behavior.

Figure 6-6: The Siftables version of the *image manipulation* application. On the left, a participant has lifted the Blur Siftable off the table, and tilted it to the left to adjust the value. In the center, the value is being adjusted as a result of the tilt (note that the on-screen visuals have changed to an effect preview). On the right the participant is applying an effect to the image by placing the effect Siftable to the right of the image Siftable, inserting it into the active effect chain.

6.3.3 Feedback from Pilot Study

A pilot study was conducted with seven participants between the ages of 18-40 to collect early feedback about the image manipulation application. All participants completed both conditions, and the order of the two conditions was randomized. In the following sections certain implementation and interface details are discussed that were changed for the final study based on feedback collected during the pilot.

Pilot Feedback: On-Screen Effect Icons in the Siftable Condition

In the Siftables condition of the pilot study, the arrangement and parameter adjustment of the Siftables was mirrored on-screen by the effect icons. When a participant placed an effect Siftable next to the thumbail Siftable, the corresponding on-screen icons would "jump" into place in the correct sequence. When a Siftable was removed from the active sequence and set aside, the on-screen icon would "jump" back into the non-active area.

Some users reported that the on-screen effect icons were distracting, and they would prefer to just see the large image pair on the computer screen rather than having the position of the Siftables mirrored on the screen. See section 6.3.4 on page 149 for how this was addressed in the full study.

Pilot Feedback: Keeping the Effect Sequence Intact During Parameter Adjustment

In the Siftables condition of the pilot study, when a user picked up a Siftable to adjust its parameter, the system would attempt to keep the effect sequence intact until they either replaced the effect into the active sequence or set the effect down outside of the sequence. This feature was designed to allow the user to adjust a parameter without removing it from the active sequence so that the "result" image on the large screen could provide useful real-time feedback about the effect manipulation. The algorithm used shake and tilt-detection configured to be extremely sensitive to determine whether the user was actively holding a Siftable. When the system estimated that the Siftable had been completely stationary for a period of one second, the Siftable was assumed to be resting on the tabletop, and if it was no longer detected in the active sequence it would be removed, updating the active sequence.

The problem that users experienced was that the system's estimation of ongoing active manipulation was not perfect. Sometimes if the participant was holding an effect Siftable very still, the system would estimate that the Siftable was no longer being manipulated, and that effect would be removed from the active sequence. The effect could be re-introduced by simply placing it back into the sequence briefly, but the lack of reliability was an annoyance to some users who reported that it made the system feel difficult to control. See section 6.3.4 on page 150 for how this was addressed in the full study.

Pilot Feedback: Real-Time Preview: On-Screen versus On-Siftable

In the mouse condition of the pilot study, during parameter adjustment the user would see a live preview of the individual manipulated effect (not the accumulation) applied to the current image on the effect icon itself. When they stopped adjusting the effect (i.e. when the mouse button was released), the effect icon's appearance would revert to showing a text label of the effect's name. If the effect was in the active sequence during the manipulation, the user would also see the result of the entire

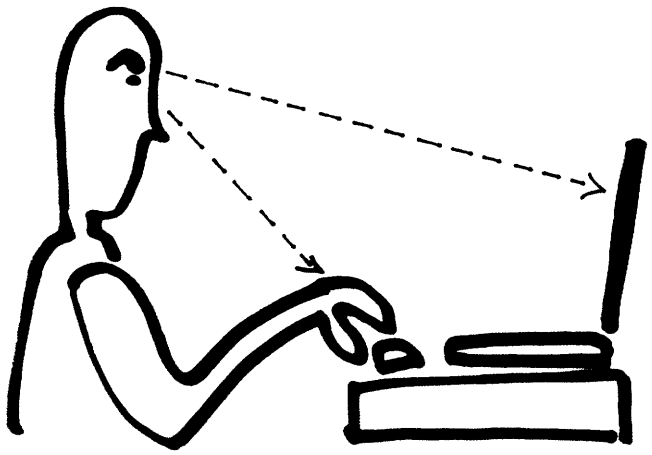


Figure 6-7: Split attention: Some participants in the pilot study complained that the distance between the laptop screen and the work area on the table where they used the Siftables was too large. They didn't like having to shift their gaze repeatedly back and forth.

effect sequence, including the manipulated-effect, on the "result" image in real-time.

In the Siftables condition, the real-time accumulated image preview was also present on the large display. However, the feedback on the Siftable's screen was only a progress-bar that stretched across the lower portion of the screen.

Some participants suggested that the progress-bar on the screen of the Siftable was not very useful compared to the (more literal) image-based feedback on the large display. Several participants proposed that the application should put more literal (i.e. effect-preview) feedback on the screen of the Siftable itself. See section 6.3.4 on page 150 for how this was addressed in the full study.

Pilot Feedback: Split Attention: Where to Look

In the pilot study, there was a relatively large difference in viewing angle between the image on the laptop display and the work area on the table where participants manipulated the Siftables (see figure 6-7). This gap was due to the side-by-side images being aligned to the top of the screen, leaving a space at the bottom of the screen, followed by the laptop's keyboard further separating the images and the Siftables.

Participants complained of having to keep shifting their attention back and forth.

One participant suggested that it would be helpful to use Siftables atop a projected surface or a horizontal display, allowing the manipulated image to be directly underneath or closer to the Siftables so both could be viewed simultaneously. Another participant claimed that the separation between the on-screen images and the physical Siftables was awkward, and suggested a Siftable-only version of the application with no large display. See section 6.3.4 on page 151 for how this was addressed in the full study.

Pilot Feedback: Tilt-To-Scroll: Responsiveness and Control

The Siftables condition of both the pilot and the full study featured a tilt-based input technique for adjusting the continuous effect parameters. In the pilot study, effect incrementing and decrementing was handled in Python on the PC. The Python program registered itself as a listener for tilt events and raw accelerometer data, and whenever the Siftable was tilted, the Python program would periodically increment or decrement the effect value based on the direction of tilt. The rate of update depended on the degree of tilt; I implemented two update speeds, fast (10 updates / second) and slow (1 update / second). A threshold value in the center of the active tilt range (about 30 degrees off-horizontal) determined whether the slow or fast rate would be used. At each effect value update, the Python program would also send drawing commands to the Siftable to update its on-screen progress bar.

Based on observing participants and reading their comments, I determined that the tilt-to-scroll affordance was difficult to control. Participants would sometimes overshoot their target value. I identified two related technical problems; round-trip latency and response jitter. Round-trip latency was due to the fact that accelerometer data had to be transmitted across the Bluetooth radio to the PC where the Python code would calculate updates then send drawing commands back across the Bluetooth to the Siftable. The minimum round-trip latency was approximately 250 milliseconds, a value clearly perceptible by participants, and at least partially responsible for their tendency to overshoot. Variability in this latency was one cause of jitter; from time to time updates on a Siftable could get "stuck" for a second or more due to Bluetooth

buffering, producing an experience of non-responsiveness. The other source of jitter was the dual update speeds. When the Siftable was tilted at an angle near the threshold between fast and slow, the update speed could jump back and forth across the threshold unpredictably due to small movement variations or normal accelerometer noise, causing the update to proceed fitfully. Along with the baseline latency, this jitter also made the interface feel difficult to control. See section 6.3.4 on page 151 for how I addressed these problems in the full study.

Pilot Feedback: Other Comments and Suggestions

Participants in the pilot study made a number of other suggestions and observations that are summarized in this section. I did not implement these suggestions in the full user study due to time constraints or limitations of the current platform, but they are presented here as ideas for future improvements.

One participant pointed out that an "undo" or "waypoint" feature would be helpful since it would allow him to explore effect configurations and adjustment levels more freely, while being able to revert easily to an earlier saved state. See chapter 7 on page 161 for a discussion of how this request might be accommodated.

One participant liked the image-manipulation application, but suggested that its advantage was primarily in the effect-chain metaphor, and that they didn't care much whether they used a mouse or Siftables as an interface. They reported that the ability to easily examine the current effect-chain, and to arrange and re-arrange effects felt better than existing applications such as Photoshop [52] that feature a history+UNDO model. This participant suggested further that the ratings for the Siftables condition might suffer because it was competing with a GUI version of the same (already-improved) task model. Overall he preferred the interaction model in the image manipulation application to Photoshop.

One participant suggested that the means of applying effects should be connected more literally to individualized spatial gestures where possible. He gave the example of placing the Siftable representing Hue in a radial position around the perimeter of the thumbnail Siftable. The position around the thumbnail Siftable would correspond

to a location in the color wheel, which would set the value of the effect.

Another participant suggested that that Siftables were better suited to playful interactions, and that a work / productivity context such as the image-manipulation application was not a good fit for the interface.

Finally, several participants commented that their feeling of control using the mouse came (at least partially) from years of experience with the desktop interface, and that with more time they would develop similar virtuosity with Siftables.

6.3.4 Changes Made to the Application

A study was conducted with 19 participants between the ages of 18-40 in order to better understand the subjective experience of interaction with an embodied media system through the lens of an updated version of the image manipulation application. The task and experimental conditions of the study are the same as described in section 6.3.2 on page 141 (within-participants, randomized condition order). In the following sections the changes to the application from the Pilot study are discussed, along with brief discussion of the impact of each change on the user experience.

On-Screen Effect Icons in the Siftable Condition

In the Siftables condition of the full study, the mirroring of the Siftables on the large screen was removed.

The benefit of this change was less on-screen visual clutter, reducing the number of places to focus attention to only two (on the images, or on the Siftables themselves) rather than the previous three (images, Siftables, or on-screen effect-chain). The drawback of this change is a reduction in feedback that the given sequence of Siftables is actually engaged. I noted this ambiguity when some participants reported being unsure about the integrity of the sequence when effects were set to neutral points and thus were not visibly contributing to the "result" image.

A refinement for future systems could be to include visual feedback on the Siftable screens to indicate when they are in the active sequence.

Keeping the Effect Sequence Intact During Parameter Adjustment

The activity-estimation heuristic from the pilot version that attempted to preserve the effect sequence during Siftable-based parameter manipulation was removed for the full study to achieve greater predictability. In the full study, as soon as an effect Siftable was removed from the active signal chain, the sequence would be recomputed without the given Siftable.

The benefit of this change was that it improved the predictability of the system. The drawback was that it became nearly impossible, and definitely impractical to have real-time feedback during effect adjustment. Some participants found that they could keep the effect chain intact during parameter adjustment by lifting all Siftables together at the same time and tilting them. This would permit real-time feedback on the large display. However, this solution was not optimal because it prevented effects from being adjusted independently.

A possible solution for future systems would be a way to manipulate the effect in-place without disturbing the effect sequence, such as a touch screen or knob on each of the effect Siftables. Another possibility would be a better strategy for knowing when a user is holding a Siftable. This strategy could be an improved heuristic that utilized motion data, or could involve touch or proximity sensing built into the body of the device.

Real-Time Preview: On-Screen or On-Siftable

In the full study the effect Siftables showed a realistic pre-computed preview of their effect applied to a sample image, with a monochromatic progress-bar overlaid on top of the preview. The graphical preview would adjust dynamically during tilting so that the user could see a sample of the effect as it was applied to the thumbnail image. After tilting had ceased for a period of one second, the screen on the Siftable would revert to showing a textual label with a red progress bar.

This effect preview was a more direct representation of the effect than the red progress bar alone. However, a software bug during the experiment resulted in the

same image preview being used for every image pair that participants edited. The mismatch between the preview image and the image being manipulated is likely to have diminished some of the advantage of the realistic on-Siftable effect preview, though the feedback on the Siftable's screen was still a more literal representation of the effect compared to a progress bar alone.

Split Attention: Where to Look

In the Siftables condition of the full study, the laptop computer was turned upside-down and placed in such a way that the left and right-hand images on the display were close to the work area where participants manipulated the Siftables. The screen was at a 30-degree angle from the table surface, and the images were rotated 180 degrees and their positions switched so that the left and right-hand positions remained as before (see figure 4-12 on page 102).

The advantage of this configuration was that participants did not have to adjust their gaze as far to switch their attention between the Siftables and the on-screen graphics.

Other solutions that could have achieved a similar advantage would have been to use Siftables atop a projected surface, or to show the result of the accumulated effects on the screens of the Siftables themselves. The current data bandwidth over Bluetooth between the PC and the Siftables prevented the latter, but such a strategy may be interesting future work.

Tilt-To-Scroll: Responsiveness and Control

In the full study, all parameter incrementing and decrementing logic was moved to the firmware of the Siftable itself, and the effect value updates were transmitted asynchronously over Bluetooth to the software application on the PC. The firmware code only implemented a single update speed, at a rate of about 2 ticks / second, in five-percent increments.

Moving the effect update code to the device's firmware removed the problem of round-trip latency, allowing the on-Siftable graphics to update more quickly and

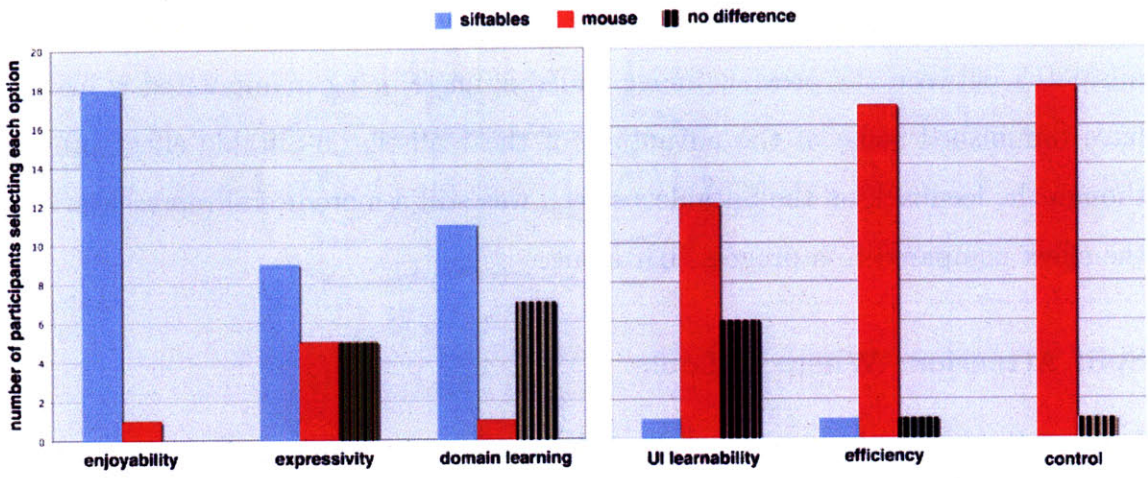


Figure 6-8: Image manipulation study, comparison ratings. After completing both conditions of the study, participants chose which interface they preferred along the given dimensions. For each comparison, they could choose Siftables, mouse/GUI, or no difference.

eliminating the jitter due to irregularities in the Bluetooth latency. Furthermore, the single update rate eliminated jitter due to tilt values near the threshold between slow and fast. These changes made the system feel more reliable, though still not as easy to control as the mouse. The disadvantage of a single update rate is that it takes longer than before to move the effect value greater distances.

Other options to solve the jitter problem that came from the dual-rate-threshold would be the inclusion of hysteresis around the threshold, or non-linear variable update rate related to the degree of tilt. Looking further, the best solution would be to completely eliminate tilt-to-scroll, as mentioned in section 6.3.4 on page 150.

6.3.5 Results

Averaging across all image pairs and participants, the study revealed a greater average completion time per image pair for the Siftables condition compared to the mouse/GUI condition (81.4 sec per image pair for Siftables versus 61.3 sec for mouse/GUI). Qualitative feedback suggests that the mapping of a tilting gesture to set continuous effect values was difficult for participants to control, which probably explains this difference.

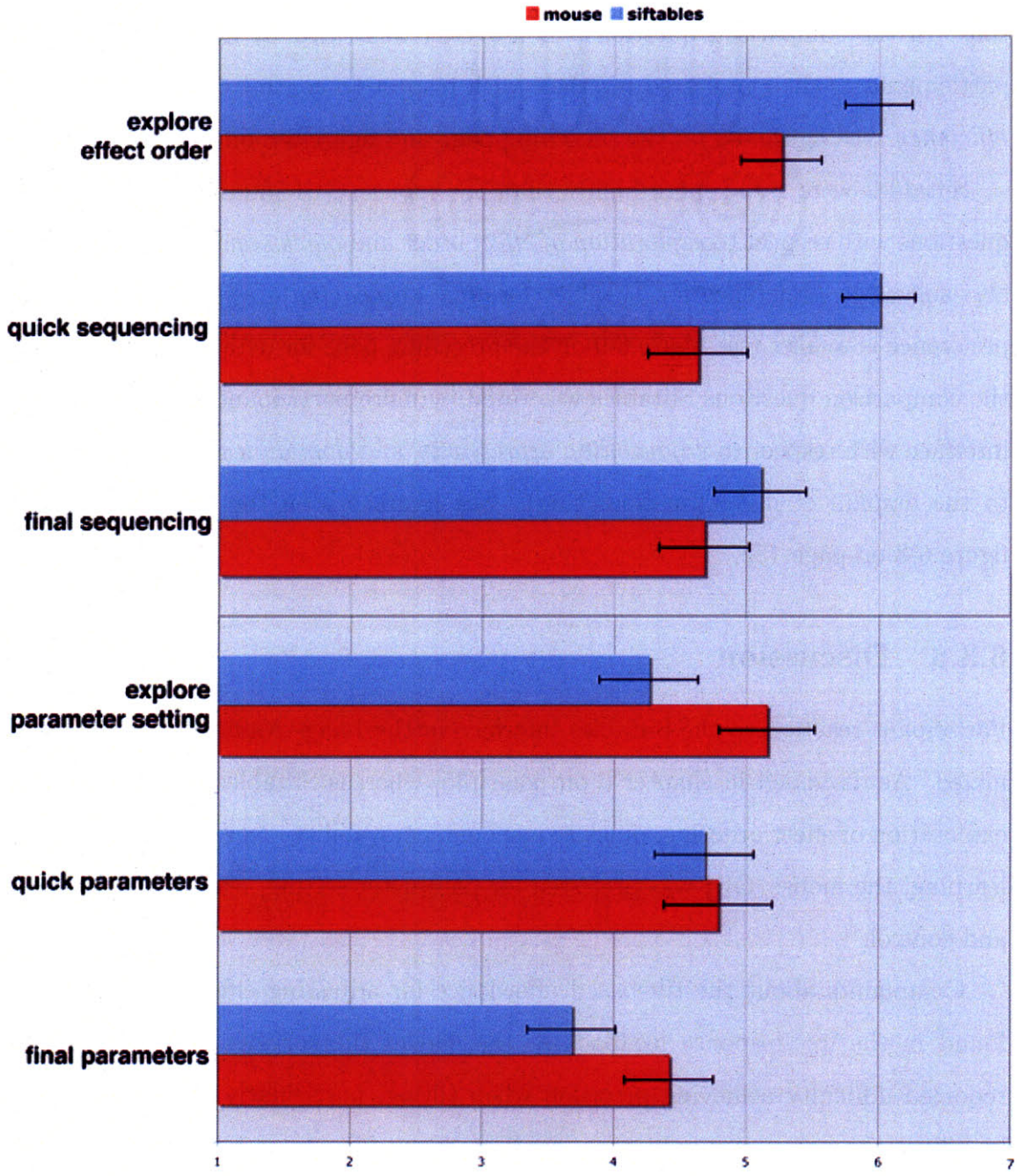


Figure 6-9: Image manipulation study, overall ratings. Participants favored Siftables over the mouse/GUI for ordering and sequencing, but favored the mouse for parameter-setting. Responses are on a scale from 1-7, and error bars show the standard error, calculated as $STDEV(values)/\sqrt{n}$

The mouse/GUI interface was rated higher on the 7-point questions with respect to *exploration of parameter setting*, *quick parameter setting*, and *final parameter setting*. In the comparison questions, mouse/GUI was rated by a greater number of participants as their preferred interface with respect to *control*, *UI learnability*, and *efficiency*. See figure 6-9 on the preceding page and figure 6-8 on page 152.

Siftables were rated more highly than the mouse/GUI interface on the 7-point questions with respect to *exploration of effect order* and *quick sequencing*. The ratings also suggest a preference for Siftables for *final sequencing of effects*, although this preference is weaker (see figure 6-9 on the preceding page for details) Additionally, in the comparison questions Siftables was rated by more participants as their preferred interface with respect to *enjoyability*, *expressivity*, and *domain learning* (with respect to the domain of image manipulation). See figure 6-9 on the previous page and figure 6-8 on page 152.

6.3.6 Discussion

Participant reaction to the Siftables interface in the Image Manipulation study was mixed. As discussed in chapter 6 on page 135, whereas Siftables were preferred for exploration of effect ordering, quick sequencing, enjoyability, expressivity and domain learning, the mouse/GUI was preferred for parameter setting, learnability, efficiency and control.

Complaints about the tilt-based affordance for adjusting effect magnitude were found in the free-response feedback at the end of the survey. Many participants reported difficulty achieving precision when tilting, particularly when they wanted to adjust the parameter only by a small amount. The greater average completion time for the Siftables condition is consistent with other work [28] [87] that reports interaction difficulties arising from the use of tilting a handheld device to adjust a continuous parameter. Translational movement (like mouse interaction), or a touch-sensitive on-screen fader would likely be more natural ways to implement continuous input. See section 7.2 on page 171 for ideas about spatial positioning of Siftables, which could be an alternative to tilt for setting continuous parameters.

Siftables were also rated more poorly than mouse with respect to ease of learning, which I attribute primarily to the problematic nature of tilt as a continuous input. Several subjects observed however that their lifetime of mouse usage gave the mouse a usability advantage for them, and that with more practice the tilt-to-adjust affordance might also become natural and more controllable. It is reasonable that in the context of a short user study with a novel user interface, participants may encounter some difficulty learning to use a system that would become second nature with time. Thus a new system may useful even if users report that it is difficult to learn initially. Doug Englebart (the inventor of the mouse) dedicated much of his career to the development of interfaces that were *not walk-up-and-use*, but rather required skill acquisition. As an example from another domain, most musical instruments require years of practice before the player reaches their expressive potential.

The positive reviews for Siftables with respect to experimentation with effect order and arrangement suggest that Siftables are useful for activities that involve the grouping and/or ordering of collections of digital items. A common activity that involves grouping (but not ordering) is the organization of files into folders, or the application of a tag/label to a group of files. Common activities that involve ordering include the creation of media sequences that unfold in time such as image slideshows, video sequences or musical compositions. Other activities where both grouping and ordering matters include the manipulation of gantt charts, system-dynamics models, or even seating guests at an event like a wedding. Furthermore many possibilities exist for educational activities such as word-building from individual letters (each represented by a Siftable), or chemistry or mathematics puzzles where the participant must arrange chemical elements or equation terms (each displayed on a Siftable) into correct sequences. Although applications were built to explore some of these ideas, evaluations would be required to fully understand the efficacy of Siftables on their educational potential.

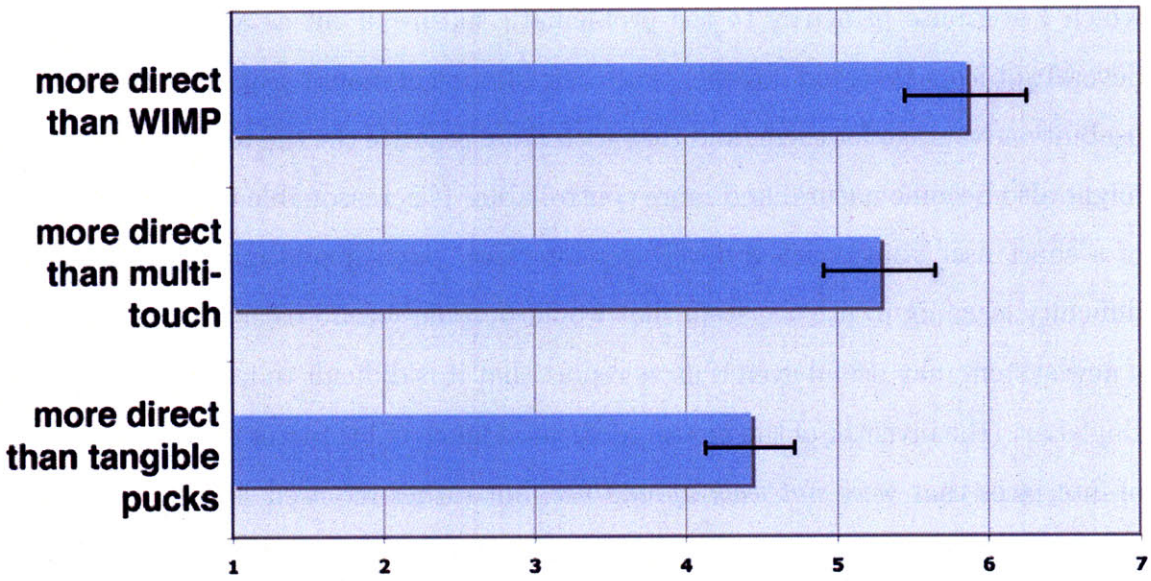


Figure 6-10: Developers' ratings of directness of Siftables as compared to other UI categories, on a scale from 1 (strongly disagree) to 7 (strongly agree), and error bars show the standard error, calculated as $STDEV(values)/\sqrt{n}$

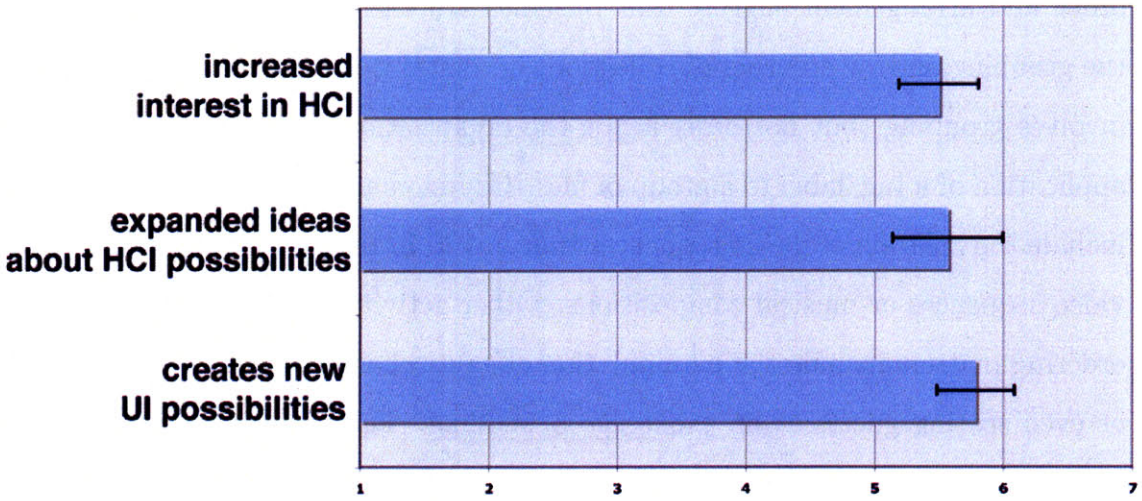


Figure 6-11: Developers' agreement with statements that Siftables creates new UI possibilities, that Siftables increased their interest in HCI, and that Siftables expanded their ideas about HCI, on a scale from 1 (strongly disagree) to 7 (strongly agree), and error bars show the standard error, calculated as $STDEV(values)/\sqrt{n}$

6.4 Developer Feedback

Since one goal of this work was to create a reusable development platform that would enable other developers to create applications for Siftables, I created an online WIKI to host instructions for getting started, API documentation, code downloads, hints, and known bugs. A successful deployment of an embodied media user interface in the world would likely require a community of developers to create applications, and the members of the Siftables development WIKI are thus a microcosm of such a future community. I sent a web survey by email to the 29 users of the Siftables developer WIKI to collect feedback about their experiences developing with Siftables and their impressions of the potential of the platform. I received 14 complete responses. The respondents are all involved in Siftables application development in some capacity. Four responses were from colleagues and undergraduate researchers at MIT, eight were from engineers and managers in industry, and two were from academic collaborators from other universities.

Respondents rated their impression of the directness of Siftables-based interaction compared to other paradigms, including Windows/Icon/Menu/Pointer (WIMP), multi-touch and tangible-pucks interfaces. The question asked variations of the following: "Please rate your level of agreement with the following statement: Siftables can be used to create a user interface (UI) that allows users to feel that they are interacting more directly with digital media than a mouse-based WIMP interface." On the scale from 1 (strongly disagree) to 7 (strongly agree), the average ratings of direct-ness were as follows: compared to WIMP (5.9), compared to multi-touch (5.3), and tangible pucks (4.4).

Respondents rated their level of agreement with statements about whether Siftables created UI possibilities and expanded their ideas about HCI possibilities, as well as about whether Siftables increased their interest in HCI, on a 7-point Likert-style scale. The results, at 5.8, 5.6 and 5.5, suggest that developers do feel that Siftables create new UI possibilities, and that working with Siftables increases their ideas about HCI, and increases their interest in HCI.

6.4.1 Developer Survey Discussion

The feedback from the developer survey is interesting insofar as it reveals attitudes that future developers might have regarding Siftables or another embodied media platform. The fact that developers felt Siftables *created new UI possibilities* and *expanded their ideas about HCI possibilities in general* offers evidence of the platform's utility in pushing the boundaries of how computers can become better tools for human beings. The fact working with Siftables *increased their interest in HCI* was personally interesting because it suggests that Siftables inspired them, which I believe is an important characteristic in order to build an enthusiastic developer community.

Developers' feeling that Siftables offer increased directness in interaction as compared to mouse/GUI and multi-touch interfaces is perhaps not surprising, given the physical nature of the manipulatives. I interpret this result as additional evidence that Siftables offer possibilities that are substantively different than other existing user interfaces.

6.5 Summary of Results

In summary, I conducted a pilot study and two full studies. From a simple content sequencing and grouping study, I learned that Siftables allowed for faster task completion than the mouse/GUI, and that pairs of participants using Siftables worked more quickly than solo participants. Given that small differences in the efficiency of a tool can have profound effects on the strategies that users employ, these results indicate that Siftables are a useful system for representing classes of problem-solving activities involving one or more users that can be easily mapped to grouping and sequencing of elements.

The pilot study was an early version of the image manipulation study, and feedback pointed out a number of interaction shortcomings, some of which I addressed before the second full study. From the full image manipulation study, I learned that while participants preferred Siftables for effect-ordering, they preferred mouse/GUI for fine adjustment of parameters. Participants also preferred mouse/GUI in terms

of UI learnability, efficiency and control; however, participants preferred Siftables in terms of enjoyability, expressivity and domain learning. Participants' preference for Siftables in terms of enjoyability and expressivity suggests their use for playful and creative activities like gaming and music. The reported preference for Siftables for domain learning is interesting, but requires further study.

Finally, developers reported that Siftables enabled them to create user interfaces that are more direct than WIMP or multi-touch, and that Siftables create new UI possibilities. Developers also reported that working with Siftables increases their ideas about, and their interest in HCI. These reports suggest that researchers and developers are enthusiastic about Siftables, and that embodied media systems extend their capabilities to push the boundaries of human-computer interaction.

6.6 Outside Interest in Siftables

Representatives from more than ten sponsor companies have requested a development kit for their engineers to work with. At this time, eight companies have paid for and received development kits containing from three to twelve Siftables. My impression of this interest, based on many conversations at the Media Lab and via phone and email correspondence, is that while the companies are familiar with the parts in a Siftable (all common components in today's consumer electronics devices) they are intrigued by the novelty of how the combination of these parts produces new user interface possibilities. Feedback from the developers currently working with Siftables was summarized in the Developer Feedback section.

Researchers in academia have also expressed interest in using Siftables as a platform to support their own work. Hartmann and Klemmer in the Stanford Computer Science Department experimented with Siftables used as dynamic graphical labels that they could place atop other physical user interface controls, such as faders on a musical mixer used as a digital input device [43]. Seth Hunter, a colleague at the Media Lab, built a series of four separate applications to teach language and reading skills to children using Siftables, some of which are described in chapter 4 on page 85.

A number of other ongoing conversations with academics indicates a clear interest in the new UI possibilities that Siftables offer.

A Google search today (May 1, 2009) on the terms "Siftables" results in about 26,000 hits, and a video of my presentation about Siftables at the TED conference has been viewed more than a million times.

Chapter 7

Discussion and Future Work

At the outset of this thesis I introduced the concept of embodied media, a new model for distributed, physically embodied user interfaces. In the second chapter I discussed related previous work that set the stage for the brainstorm and design discussion of Siftables and gesture language possibilities related in the third chapter. In the fourth chapter I enumerated a number of applications that have been implemented using Siftables, and the fifth chapter covered the technical details of the system. The sixth chapter described a pilot and two user studies that investigated some efficiency characteristics of embodied media to the mouse/GUI and the attitudes of users and developers that have worked with Siftables, finding that study participants preferred Siftables in terms of enjoyability, expressivity, domain learning, and for exploratory/quick arrangement of content items, and that Siftables offered an advantage for task completion time (particularly when participants worked in pairs).

This chapter begins with a discussion that summarizes the benefits of embodied media, outlining the types of applications to which it is particularly well-suited. It continues by identifying a number of specific lessons regarding the interactive possibilities, design techniques for applications, and recommendations for developers creating embodied media user interfaces such as Siftables. From there, I look forward to features that could be incorporated into the next generation of the Siftables platform and some open questions, and finish with thoughts on the future of our interactions with collections of networked smart objects.

7.1 Summary: What is Embodied Media Good For?

User feedback from the applications that I and others have created suggests that an embodied media user interface platform like Siftables is a compelling and useful advancement of the state of the art in human-computer interaction. The following section enumerates properties that are characteristic of activities that are typically accomplished today using desktop computers, but that an embodied media interface is intrinsically well-suited to support.

7.1.1 Characteristic Properties of Well-Suited Applications

- *Involves spatial arrangement of items*: Physical manipulatives can be arranged into spatial configurations as an input. For instance, language, science or mathematics tools using embodied media could allow learners to compose words, molecules, or equations from component parts and receive real-time feedback about the correctness or implications of their solution. Distributed cognition predicts the advantages of manipulating the physical position of items in these interactions. The current Siftables instantiation permits two-dimensional arrangements, but this limitation could be lifted in future versions of Siftables or other embodied media instantiations.

- *Involves iterative definition of relationships between content items*: A distributed physical interface that can be manipulated efficiently permits quick experimentation with the relationships between items. Visual on-manipulative feedback allows the assignment of content to each manipulative to be legible to the user. In addition to the distributed cognition advantages of spatial layout, educational interactions are particularly well-suited to embodied media due to the presence of on-manipulative feedback. Furthermore, there is a strong connection between educational interactions and gaming. Embodied media is also a promising interaction paradigm for puzzle, action, and narrative oriented games, particularly given its ability to support the face-to-face, social play patterns of classic games like dominos, board or card games.

- *Benefits from collaborative interaction*: The collection of independent manipulatives enables groups of people to interact simultaneously with an embodied media application in a manner that is difficult or impossible with a typical desktop system. Multi-person collaboration is an important human way of working and playing, so interfaces like embodied media that support parallel interaction are compelling and valuable.

- *Users are children or special needs community*: Embodied media manipulatives can have a similar size and physical interaction style as traditional wooden manipulatives; they offer accessibility to children who are not yet proficient with the mouse and/or keyboard. Additionally, the embodied media interaction style may make computer use accessible to special needs users with physical or mental disabilities (I have been told this by several parents of children with learning disabilities; however I have not tested Siftables with these populations).

- *Involves non-precise multi-manipulative gestural input*: Tilt-to-adjust was difficult for users to control, but the Wii demonstrates playful interactions can be designed in such a way that the lack of precision is acceptable. The effects in the music sequencer application were more satisfying in informal tests, and there are many other possibilities for expressive, continuous multi-object gesture that have still not yet been explored thoroughly. Live music and video *performance* settings are one such promising context of use. The expressive potential of three-dimensional gesture with a collection of separate devices, each producing a different sound or effect type, could make for a compelling stage show. Data visualization and search is another domain of practice where multiple "handles" can be advantageous, and if the interaction style is designed to accommodate non-precise input, the operator may be virtuosic, efficient, and effective.

- *Involves content manipulation, not entry*: Embodied media is good for arranging and adjusting content, but not as useful for data entry. Consider a mind-mapping application for example: manipulatives will not replace text input techniques such as the keyboard or speech recognition, however they can offer a

benefit when the user needs to define and adjust the relationships between the concepts. Said another way: While I wouldn't want to use Siftables instead of my laptop computer to write an email to a friend, I would rather use them to determine the seating arrangements at my wedding, to teach my child how to put sentences together, or to quickly try twenty different search term combinations to learn which produces the best results! As we have more and more digitized information on hand, we will need more effective ways to sift and sort our way through the mountains of data, and interfaces that leverage our existing physical-world skills can provide an advantage.

There are certain application characteristics that are *not* well-suited to embodied media, or that at least must be designed with caution and strong user testing. Applications that split the user's attention between the manipulatives and a larger display are one problematic example, since the user may not know where to look at a given moment or may tire of shifting their gaze back and forth. Another example is the use of tilt as an input for an application that requires fine continuous control.

7.1.2 Takeaway Lessons: Design Opportunities and Recommendations

The set of diverse applications for Siftables that have been implemented by myself and by other developers is further validation of the contribution of embodied media. From user feedback and my own experience with these applications I now distill some lessons learned about the unique possibilities that embodied media provide for application development, and some specific design techniques that have proven useful.

Multi-Person Collaboration Around Collections

The potential to enable collaborative work is shared with other tabletop and multi-touch systems, but the combination of physical manipulatives, *anywhere tabletop interaction* (see below), and on-object graphical feedback gives embodied media some unique possibilities for collaborative interaction with collections of information and

media content. Specifically, the on-object content depiction and feedback and gesture/neighbor sensing with minimal-infrastructure requirements opens up possibilities for collaborators to change their venue if needed (mobility), to use the physical environment more readily (non-electronic props), and to keep their attention focused on the manipulated items (rather than on a separate display).

Two-Handed, Physical Manipulation

Two-handed, physical manipulation is a characteristic embodied media shares with tabletop and multi-touch systems. However, neighbor sensing and three-dimensional interaction enable a richer set of possibilities for how two hands can be used together to interact. When grouping, sequencing and arranging, shaking or tilting, the use of both hands is natural and efficient. Taking this idea further, a greater degree of *bodily* interaction is enabled by multiplicity of manipulatives in an interaction, for instance when using a forearm to number of Siftables across a desktop at the same time.

Anywhere Tabletop Interaction

As discussed in chapter 2 on page 25, tabletop interfaces have recently become quite popular in the research community. The advantage of fully self-contained manipulatives such as Siftables is that they can implement some of the same interactions while obviating the need for the (typically) non-mobile sensing and display infrastructure of most tabletop systems. This allows an embodied media system like Siftables to be mobile in a way that large multi-touch surfaces or tangible pucks systems are not.

3D interaction

Another limitation typical of tabletop interfaces that an embodied media interface such as Siftables overcomes is the constraint of two-dimensional planar interaction. Although only simple three-dimensional gestures have been explored in the scope of this thesis (shaking, tilting), sensing of three-dimensional gestures of greater complexity is possible to implement. This allows for spatial interaction possibilities that would not be feasible with multi-touch or tangible pucks systems.

Multi-Object Gestures

Work with siftables has enabled the exploration of several interesting multi-object gestures. For instance, the way that the player can "dump" their character from one tile to the next in the Maze Exploration game, and the "color pouring" interaction sketch both involve more than one device being used gesturally together. The "thump" gesture offers a way to interact with a group of Siftables in parallel, the way that shouting can attract the attention of a group of people in a single instant. The distributed, physical nature of embodied media interafaces permits three-dimensional gestural interactions with collections of manipulatives.

New Opportunities for Action-Borrowing

The aforementioned multi-object gesture examples (dumping, pouring, thumping) highlight a key property of embodied media: as physical-digital tools acquire more sophisticated ways to sense each other and to sense the world around them, designers acquire an expanded set of possibilities for how actions can be borrowed from our everyday interactions and implemented metaphorically (for instance transferring an item from one container to another).

Design Rules for Tilt: Two Interaction Strategies

The different tilt strategies I used for sample effects versus global effects in the music sequencer application resulted from different interaction needs for these two manipulations. Sample effects (such as filter) are manipulated with expressive intent, which made a direct instantaneous-tilt-to-value mapping appropriate. A global effect (such as volume), however, was more likely to be used in a non-expressive manner, where it would be adjusted to a certain value then left at that value. This usage suggested the "deadband-on-flat" strategy whereby no adjustment would be made when the Siftable was laid flat on the table.

Other designers might have different sensibilities about the categorization of these particular effects, but the general principle of using instantaneous tilt for expressive

manipulation versus deadband-based tilt for "set and leave" manipulation should be a useful principle in future embodied media instantiations. As discussed already, the embodied media designer should carefully consider the usability of tilt as compared to other affordances for setting continuous values, such as knobs or touch screen controls.

See section 4.12.2 on page 105 for more information and background related to the design problem of tilt-based input.

Shake-to-Lock/Unlock: A Method for Button-Free Direct Tilt Mode Enable/Disable

The node-edge graph creation tool combined a direct instantaneous-tilt-to-value affordance with a shake-detection-based way to enter and exit the direct tilt-based interaction mode (see section 4.12.2 on page 105 for more information and background to the design problem).

The key design insight that made this scheme work was that the value should be taken from about 0.5 seconds *before* the shake event is detected, thus capturing the steady-state tilt angle before the motion of the shake event perturbs it. This design pattern may be useful for other embodied media user interfaces that require mode-switching into and out of direct tilt-based input (or another motion-sensing mode) without the use of buttons.

Exploring Larger Content With Smaller Screens

The Maze Exploration application provides an example of how an embodied media interface with at least two small screens can be used to explore a spatial problem or a digital content item that is too large to be fully represented on their screens at a single moment. Even a single Siftable could be used as a window onto a larger territory, for instance using the nudge-based traversal idea outlined in section 3.4.1 on page 69. This *progressive disclosure* approach to overcoming the limitation of a small screen could be useful to other mobile device scenarios, particularly ones with a narrative or puzzle element.

A related idea that I do not find particularly compelling, but that many people have suggested when they first encounter Siftables, is the possibility of using a collection of embodied media manipulatives placed together into a grid to display a single larger image or video that is too large to display on their individual screens. Yes, this would totally be possible.

Importance of On-Manipulative Feedback

Since the ability for Siftables to detect their spatial arrangement and sequencing with respect to each other is not based on physical contact, there are conditions in which, without appropriate feedback, the user may be uncertain whether adjacent Siftables have successfully recognized each other. This uncertainly primarily occurs when the two manipulatives are spaced far enough from each other so that they are right at the edge of their ability to communicate using infrared. In this situation, as well as in the less common scenario in which a communication error allows adjacent devices to "miss" each other, some on-screen indication that the two manipulatives are aware of each other's presence is useful. Furthermore, in practice such responsive feedback based on adjacency seems to be pleasing to users.

On-Manipulative Hints

A feature that Fiddle Diddle Make a Riddle explores is just-in-time graphical hints that are displayed on the manipulative when the user appears to be stuck. This approach could be useful for many games or puzzle applications where the location of feedback is important. This self-description is akin to the already-noted possibility that embodied media devices can represent problem constraints graphically (see the following section).

Distributed Cognition Advantages

As discussed in chapter 2 on page 25, Kirsh and Maglio's finding [71] highlights the utility of physically arranging Scrabble tiles. Siftables in *Scraboggle* leverage this utility since they are also physical and can be rearranged, but there is even more

to explore regarding the use of the manipulatives' screens to represent relational constraints. For instance, in an application that features a node-edge topology such as a project-planning tool, compatible connections between items could be represented visually by matching colors or shapes at the edge of the displays that are visually similar or that suggest interlock to indicate their compatibility (borrowing the look of male and female puzzle-piece connections is one possible design). A simpler example would be an extension to the music sequencer application that would draw the left and right edges of the sequence Siftables differently than the top and bottom edges, to indicate the potential to connect the left-right edges to other sequence Siftables. In general, embodied media manipulatives can use their screens to dynamically represent the ongoing state of a problem or partial solution as a user interacts with the system.

Predictability is Key

A challenge that many gesture-based interactive systems encounter is the difficulty of responding in a way that seems completely predictable to a user. My discussion of the Theremin in chapter 2 on page 25 pointed out a problem of free-gesture, namely that it is difficult for a person to move their body in exactly the same way twice. Since gesture detection algorithms (even simple ones such as the detection of tilt past a threshold) must respond to the user's imprecise body motion, there are ample opportunities for gesture-based systems to feel unreliable or unpredictable.

An example of prioritizing predictability over responsiveness was my "downgrade" of the tilt-to-adjust method in the image manipulation application from dual-rate in the pilot to single-rate in the full study. Although the pilot method and the full study method were not compared directly against each other, after trying both I can say that the single-rate method felt more predictable and reliable. In general, systems that afford multi-object gesture should strive to be as predictable as possible if the designer wishes them to feel reliable.

Split Attention Problems With Large Screens

A takeaway from both the image manipulation study and our observations of children using Telestory is that interaction with a mixture of manipulatives and a large display must be designed carefully. In the image manipulation task, some participants complained about having to look back and forth repeatedly, and the children using Telestory hardly looked at the Siftables at all, keeping their attention focused almost exclusively on the large screen except when looking for another manipulative to pick up. The children's behavior is not surprising, given that in Telestory all of the action happened on the large screen, and its size made it an easier (and perhaps more appealing) location for visual focus. However, the difficulty in getting the children to look at the manipulative instead of at the large screen seemed to increase the amount of intervention required to teach the children about the tilt-based on-device menu.

One solution would be to remove the large screen from these interactions completely, locating all visual information and feedback on the manipulatives' screens. This may be appropriate for activities in which the content does not require high resolution graphics or close inspection. However, for activities that do need a large screen, tighter and more granular temporal coordination between the activity on the large and small screens may be a solution. For instance, a character on the large screen could audibly tell the user to look at a given manipulative, then disappear from the large screen and appear on the device's screen to continue the instruction from there. This transition could guide the user's attention, handing it off from the large screen to the device, and a similar transition could guide their attention back to the large screen.

Feedback Latency: Short-Circuit when possible

As with all interactive systems, latency matters a lot, and lower is better. The implication for embodied media is that certain feedback displays may work better if they operate as a "short circuit" that directly connects an interaction sensed by the manipulative to a corresponding visual display. The latency for such feedback can

be lower if it doesn't have to first go across the radio, then incur processing on the computer, then return over the radio before appearing.

The decision to "short-circuit" the feedback in the image manipulation application greatly reduced the latency between tilting to adjust the value and the length of the on-screen feedback bar changing. Neighbor detection is another embodied media scenario that suggests short-circuiting is feedback, for instance showing an on-screen marker when a neighbor exists on a given side.

7.2 Future Work

Although the end of this thesis approaches, the story of embodied media continues. In this section I first examine the ongoing work that is in progress at MIT through sponsor collaborations and student investigations. I then look forward to how Siftables might develop in the future, followed by some open questions about embodied media that could be answered with further study. Finally, I conclude with a discussion of the big picture, looking towards the kind of user interface that my work with Siftables suggests but that is still some years off.

7.2.1 Iterating on Siftables

There is ongoing momentum around Siftables at MIT. One aspect is the number of collaborations with research groups at our sponsor companies that seek to explore a range of application possibilities and extend the capabilities of the platform. One laboratory is interfacing a vibrotactile actuator to each of their Siftables in order to provide feedback to a user holding them. Another team is building an application to allow their customer service representatives and customers to work together more effectively. This application uses Siftables in face-to-face sales situations, to allow the staffer to work with the customer to better understand their financial situation and investment goals. Other research and development groups are pursuing their own application ideas. All together, close to one hundred Siftables have been distributed to eight separate development groups in our sponsor community.

Student work at the MIT Media Lab will also carry Siftables development forward. My colleague Seth Hunter has created a number of narrative-oriented applications for children, as discussed in chapter 4 on page 85, and plans to continue to work with Siftables. Other students at MIT are actively involved as well, currently prototyping a richer equation editing tool and extensions to online social networks.

My own work, as well as the explorations of developers at sponsor companies, and my colleagues at MIT, has brought both the capabilities and limitations of the current Siftables platform info sharp relief. The following sections suggest a number of novel not-yet-implemented capabilities that are under consideration for future development.

Locomotion

Interaction possibilities would be enhanced if Siftables had the ability to move themselves across a surface. Small tractor-style treads, wheels, or even hair-like *cilia* coupled with the ability to vibrate could allow Siftables to arrange themselves into different spatial configurations. This would enable, for instance, an UNDO capability that could put the manipulatives back into a previous configuration.

Absolute Position Sensing

Autonomous movement would be most useful if it were combined with absolute position sensing. The existing neighbor detection supports problems that can be mapped to sequences, chunks of contiguous 2D topologies or groupings of content items. However, continuous sensing of position (or even continuous sensing of the distance between manipulatives) would allow the distance between Siftables to be used as an input, for instance in the manner that Audiopad utilizes a sound-clip puck's distance from the "microphone" puck to control its volume. A number of technical approaches could provide Sfitables with the ability to sense absolute position, one of which is discussed in the next section.

Microphone(s) and speaker(s) on board

The inclusion of one or more microphones and speakers on each Siftables would open up several interesting possibilities. The primary interaction opportunity would be for sound recording through the microphones, either in the form of spoken commands or capturing and responding to the ambient sonic environment. A multi-modal approach that allowed users to interact by voice and gesture together could create further interaction efficiencies or accessibility-related affordances for disabled users. Language-learning applications could especially benefit from this. A speaker on each Siftable would permit localized auditory feedback that could emanate from the manipulatives themselves rather than from a nearby computer as in the current setup. A secondary opportunity that the inclusion of several microphones and at least one speaker could enable is absolute positioning. Research in the distributed sensor networks community has explored the use of audible or ultrasonic "chirps" to allow a collection of sensor nodes to collaboratively understand their spatial layout. Such an algorithm could allow Siftables to locate each other precisely in space while preserving their minimal reliance on environmentally installed infrastructure.

Mesh Network Architecture

For the purposes of understanding the interaction implications of Siftables, true mesh networking capabilities were not necessary. However, for more realistic deployments a mesh architecture could be an advantage, as it would allow Siftables to more fully leverage state-of-the-art advances in wireless sensor networks. It could also enable applications that that do not require a wireless connection to a computer to be created more easily. In order to create mesh-capable Siftables, the type of radio used would have to be changed. Rather than Bluetooth, which is optimized for point-to-point "cable-replacement" scenarios, a ZigBee-capable or other mesh-oriented radio technology would be appropriate.

Electronic Ink / Reflective Display

One of Siftables' most notable departures from wireless sensor network architectures is power usage. Rather than permitting days or weeks of run-time on a single battery charge, Siftables last from 4-10 hours before needing to be recharged. The most significant consumer of power is the graphical display. Although the current OLED display is capable of being used in a power-thrifty manner, reflective display technologies would permit much lower power usage overall without necessarily compromising graphics usage. Certain types of *reflective* displays [23] only draw power when the displayed image is being changed, which could mean significant energy savings for applications that do not require constant screen refresh. Examples include Siftables versions of card or token-based games like Pokemon or Dominos.

Graphics in the Spaces Between Manipulatives

Another limitation of the current platform is the confinement of graphical feedback to the Siftables' screens. For applications with an underlying graph representation (i.e. nodes + edges), graphical edges drawn between the nodes would help the user understand the connectivity of the graph. One straightforward way to solve this problem would be to combine Siftables with a larger graphical display, such as the type of projected surfaced used in tangible tabletop interfaces. I may try this, however this solution introduces infrastructure that would limit the mobility of the interface. An alternative solution recently proposed by a colleague for the Siftables themselves to project visual feedback; for instance, low-power lasers could be leveled horizontally and aimed toward other Siftables to illuminate paths of interconnectivity between the devices. This would require spatial positioning capabilities.

Non-Contact Free Gesture Sensing

It would be interesting to allow Siftables to sense the proximity of a user's hand. This could be accomplished by capacitive or optical ranging, and would allow Siftables to respond to hand gestures that the user makes in the air above a Siftable. For instance,

a row of Siftables could become a continuous fader as the user moves their hand back and forth above them. Such sensing would provide an alternate input modality, and could also allow Siftables to enact anticipatory feedback when they are about to be picked up or moved by the user.

Alternate Form Factors

Siftables currently feature a generic square tile form factor. Compared to tangible interfaces that build interaction affordances or constraint representations into the shapes of the physical interfaces themselves, Siftables' rectangular tiles offer only a generic physicality, similar to that of the "pucks" found in most tangible tabletop interfaces. Their current ability to conform to heterogeneous task domains comes largely from their graphics capabilities. However, I am interested to experiment with variably-shaped manipulatives that can be used together.

Shape-Changing and Actuation

Related to the point about alternate form factors, I would also like to explore ways that a single Siftable might change its shape dynamically to fit specific interaction roles or represent problem constraints. For instance, such a "shape-changing" Siftable might become taller to indicate an increase in the quantity that it represents, or morph its edge profile like a puzzle piece to allow interlock with a matching type of edge profile on other Siftables.

The ability to change shape or actuate physically could also allow Siftables to push other Siftables or other objects away from them, which could be used to enforce problem constraints. For instance, in the task of seating guests at a dinner this feature could be used to disallow certain people being placed side-by-side. The feature could also be achieved with embedded electromagnets, which could allow a complimentary "stickiness" that would bind certain manipulatives to each other based on the problem state.

Higher level authoring

Authoring coordinated behavior for a distributed system is known to be a challenging problem. One of the side-effects of my work on Siftables has been the creation of a relatively high-level framework for wirelessly controlling individual manipulatives. This framework, however, is really only a mid-level abstraction architecture. The current API permits a star topology, where the computer acts as the control node, with a single bidirectional connection to each Siftable. It should be possible in the future to reach even higher levels of behavioral abstraction.

A question that came up during the early development of Siftables was the following: what would a higher-level API for embodied media be like? An analogy to other application development frameworks provided useful ways to think about this question. For instance, graphical dataflow authoring environments such as Max/MSP and Pure Data have lowered the barrier to entry for many interaction designers and computer music enthusiasts. The key factor underlying the success of these environments was that the dataflow representation was well-suited to implementing certain types of programmatic behavior. Similarly, the Flash authoring environment has allowed multimedia developers to create interactive programs without requiring much in the way of procedural programming skills. Given the success of these environments that offer high-level, alternative representations for programming, the question can be posed as: what would "PD" or "Flash" for embodied media look like? Specifically, what alternative representation of embodied media behavior could dramatically lower the difficulty level in authoring complex coordinated behavior for the platform?

Beginning with some infrastructure that was developed for my own applications, I can suggest a few possibilities for alternate representations for programming an embodied media system like Siftables. The first is sequence-detection, an infrastructure piece that I built for the *Scraboggle* application. I wrote a function that abstracts the task of determining rows or columns of Siftables that the user has created, returning a list of such row/columns. This abstraction encapsulates the numerous individual topology change messages that are conveyed each time a Siftables gains or loses a

neighbor. Here is a simple example: When Siftables A and B are placed side-by-side, A generates a message saying *B is now on my right*, and B generates a message saying *A is now on my left*. The sequence-detection function coalesces these messages and returns the string "AB" as a result. Of course the number of such messages scales up with the number of Siftables involved, and the benefit of such an abstraction becomes clear.

An obvious extension to the sequence-detection abstraction would be a function that returns all instantaneous 2-dimensional topologies of a set of embodied media manipulatives. This would facilitate a greater number of application possibilities, since simple topology detection would not be limited to 1-dimensional patterns. An event-driven architecture for detecting topological patterns with certain properties could also be a useful abstraction. Specifically, it might simplify the programmer's task if they could provide a callback function that would be executed whenever certain topological conditions were met. A flexible language for describing such conditions could be extremely valuable.

Sequence and topology detection abstractions are specific examples of the more general notion of gesture-detection. Defined broadly, gestures might consist of user actions on a single embodied media manipulative or multiple manipulatives, as discussed in chapter 3 on page 47. An architecture for specification of *compound gestures* would permit great flexibility and would be a useful abstraction, particularly if coupled with the event-driven model already articulated. For instance, a programmer could implement user-initiated pairing behavior by shared synchronous motion by installing a callback listener function that would initiate a pairing attempt whenever two manipulatives were moved similarly at the same time.

Another tool that could make embodied media programs more easy to author could be a program-by-example tool for specifying system behavior. Building a detector for a particular gesture could be accomplished by executing the gesture on a set of manipulatives. To disambiguate which properties of the gesture and/or manipulatives involved, a representation of the detected action could be displayed in the programmer's development environment and they could indicate which features

were important (i.e. Was the specific motion important, or would any motion do? Should any three manipulatives being placed in a row be considered important, or is some feature of this particular three important). The disambiguation interface would not be trivial to create, but could greatly streamline the creation of gesture-driven behavior.

Finally, it would be interesting to enable end-users to program their set of embodied media manipulatives. For example, a gamer could program the response of their virtual character to different gestures, devising mappings that the game designer may never have anticipated. A child could help their parent learn to use a computer by creating personalized gestures that are easy for their parent to enact with their manipulatives. Such accessible customization would create exciting future possibilities for embodied media.

Gesture Possibilities

The work in this dissertation has explored relatively simple single-manipulative gestures (i.e. shaking, tilting). However, as mentioned in section 3.4.3 on page 73, the inertial motion-sensing capabilities of Siftables (and other future embodied media manipulatives) create compelling possibilities for expressive and nuanced three-dimensional gesture. Recognition of user-defined, complete gesture shapes a direction that I explored in my master's thesis [78] [80]. A user-defined language of more granular gesture "atoms" and an accessible way to easily string them together would be an interesting and useful abstraction for embodied media development.

Mapping expressive continuous gesture to music or video control is another possibility for future work. It is beneficial for assigned mappings to be displayed graphically as in the music sequencer application described in section 4.3 on page 88, but it would be interesting to explore more nuanced sound generation techniques like granular synthesis. For instance, grain size and density could be mapped to the forces sensed on different axes of a manipulative. An idea for video performance is that a stack of effect-manipulatives could be assembled on top of a manipulative representing the currently playing video clip, as a way to apply the stacked effects to the clip. The

user's gestural movements of the stack would control their parameters in real-time, or each could be removed and gestured with by itself. With either audio or video performance, the gesturally-sensitive effect manipulatives could be attached to the performer's body to enable dance to impact the media output.

Earlier work in shared synchronous motion was discussed in section 2.7.5 on page 46, and an embodied media interface could leverage this principle in interesting and novel scenarios. I suggested a heuristic based on this work for detecting a user's grouping action in chapter 3 on page 47 that would monitor the inertial patterns of an entire set of manipulatives. I did not explore this, but the application of this known technique to an arbitrary number of interactive manipulatives would be a useful building block for future application design.

7.2.2 Open Questions For Future Investigation

In the course of this dissertation I have attempted to make a broad range of contributions surrounding the definition, characterization, instantiation and evaluation of the embodied media design concept. The studies that I ran indicate efficiency and collaborative use advantages for a simple sorting task and user preference for Siftables over the mouse/GUI in terms of enjoyability, expressivity, domain learning, and for exploratory/quick arrangement of content items.

However, there are other properties of user interaction with embodied media that would be interesting to measure in the future. For instance, quantifying the interaction characteristics of each action in the gestural language described in section 3.4 on page 65 (shake, tilt, thump, etc.) in terms of learnability and user performance would give application designers more guidance when determining appropriate mappings from these actions to application scenarios. Relatedly, precise measurement of the capabilities and limitations of the *position estimation* interaction sketch (section 4.12.3 on page 108) could provide guidance about what types mappings are appropriate for this technique and what feedback is necessary to make the interaction easily controllable.

Longitudinal studies with an application such as the image manipulation tool or

the music sequencer would shed light on how user attitudes and performance changes with longer-term use of embodied media. It would be interesting to learn how practice would impact users' performance when controlling a tilt-based interaction, and if their frustration would diminish over time regarding shifting their visual attention back and forth between the Siftables and a large screen.

It would also be valuable to quantitatively measure the effectiveness of embodied media as an educational tool, compared to current methods in education. This data would be useful to designers of educational applications for Siftables and other embodied media platforms. Finally, I suspect that there are age-related differences in how embodied media applications should be constructed for maximum efficiency, including a minimum age at which the manipulatives are effective at all. Studies that could produce age-specific design recommendations would be useful to creators of both educational and game applications.

7.2.3 Physical Interactions with Collections of Networked Smart Objects

The future of information and media technology will increasingly contain collections of interactive, wirelessly-networked "smart" electronic objects. Predicted for at least a decade by researchers and analysts in ubiquitous computing, we can see the first wave of this future upon us in the ever-present mobile phone. Patterns of decreasing technology costs including Moore's law have moved us from the paradigm of many people using one computer (many to one), through the one person per computer (one to one) model of the late 1990's. The emergence of personal digital assistant devices and mobile phones pushed the pattern further along the same trajectory, to today's world in which one person with several devices (one to many) is common. The typical young person in the developed world today owns a laptop computer, a mobile phone, and a personal music player. She probably owns a number of other wirelessly-connected computers that she may not even recognize as such, for instance a bicycle odometer that receives data wirelessly from a wheel sensor, a Bluetooth

headset that communicates wirelessly with her phone, or a "key fob" that remotely opens the doors of her automobile. While this stereotype is not yet representative of the overall worldwide population, it does not seem outrageous to consider her a leading indicator of tomorrow's world.

Although an individual in a developed country can already afford to own a multiplicity of computing devices, the pattern is set to continue. The cost of making silicon chips is still decreasing, and device-fabrication techniques are becoming ever more efficient. With upcoming advances in portable power, we are likely to find ourselves surrounded by at least order of magnitude, and ultimately more, digital devices. Wireless communication standards will enable heterogeneous classes of devices to communicate with each other, autonomously gathering and sharing information about weather, the sonic environment, the movement of vehicles, and people's interactions with them, thus becoming an extension of our collective sensory apparatus.

Critics of ubiquitous computing suggest that this future may become increasingly inconvenient, even hostile to its human inhabitants. We will find ourselves ever more frequently at the mercy of poorly-designed or malfunctioning devices, for instance locked out of card-access areas when the power goes out [38] or spending more and more time managing the files and emails on computers. They contend that we will become simultaneously dependent on, and beleaguered by, a world full of interactive technology that demands our attention and controls our activity patterns.

Although I understand the critic's point, I am more optimistic. To avert the inconvenient and possibly hostile future, I believe that the most important direction to focus our future work in ubiquitous computing technology is toward better design of the user interfaces that we create for it. The positive outcome of a growing population of technology devices in our everyday lives is the possibility of ever more engaging and valuable interaction opportunities; sensor network technology can implement a functional fabric around us that we interact with to instantly access information and to connect with other people in ways that our parents and grandparents never imagined. Whereas much of the ubiquitous computing community sees this functional fabric being built into the spaces that we inhabit and the garments that we wear, I

envision many new interactive systems that allow us to directly manipulate digital information and media content with our hands, in the timeless style of craft.

I believe that the world needs hand-tools for the digital age, and embodied media is a step in this direction. These tools will draw upon a long history of interaction with physical objects, but will be easily portable and generalizable across many different tasks. They will become a new ecosystem of instruments for interacting with digital information and media in ways that are a better fit to how our our brains and bodies evolved. When we as technologist-designers creatively blend physicality with flexible input and output possibilities, we can enable interactions with digital content that are useful and compelling, though tools that bend to meet our needs, rather than bending us to meet their limitations.

Appendix A

Python API

This appendix contains a listing of the Python API that was used to implement many of the applications for the Siftables platform.

__init__(self, conn=None, bt_name='', bt_id='', serial_port='')

siftable constructor
if a connection is passed in, the constructor will use that connection
if a bt_name is passed in, the constructor will attempt to make a connection to that name using a pybluez RFCOMM connection. (Windows/Linux only)

acc_calibrate(self)

calibrates the accelerometer. note: this takes more than a second

acc_curr_calib(self)

returns the current accelerometer calibration values

acc_curr_frame(self)

returns the current raw accelerometer data frame. format is [x,y,z], where each value is on [0-255]

acc_curr_shake(self)

returns the current shake state. format is [x,y,z], where each value is 0 (not shaking) or 1 (shaking)

acc_curr_tilt(self)

returns the current tilt state. format is [x,y,z], where the value is:
on x: 2 is tilted left, 1 is neutral, and 0 is tilted right
on y: 0 is tilted up, 1 is neutral, 2 is tilted down
on z: 1 is right-side up, 0 is upside-down
note: accelerometer must be calibrated before this command will work. see acc_calibrate

acc_curr_var(self)

returns the current accelerometer variance frame. format is [x,y,z], where each value is on [0-255]. note: this may be a bug, since variance values are 16-bit unsigned

acc_events_shake(self, command)

turns reporting of shake events on or off. you should have a handler installed before turning this on, or the events will be discarded. (takes True/False)

acc_events_tilt(self, command)

turns reporting of tilt events on or off. you should have a handler installed before turning this on, or the events will be discarded. (takes True/False)

acc_get_sensitivity(self)

Sets the sensitivity of the sensitivity by altering the gain on the input stage of the device.

The values that will be returned by acc_get_sensitivity are:

'1.5g'

'2g'

'4g'

'6g'

acc_set_sensitivity(self, sensitivity)

these are the values to feed to acc_set_sensitivity

siftable.ACC_SENSITIVITY_1p5G

siftable.ACC_SENSITIVITY__2G

siftable.ACC_SENSITIVITY__4G

siftable.ACC_SENSITIVITY__6G

acc_set_shake_threshold_all(self, threshold)

sets the shake threshold for the x, y, and z axes to the same value, on [0-65535]

acc_set_shake_threshold_x(self, threshold)

sets the shake threshold for the x axis, on [0-65535]

acc_set_shake_threshold_y(self, threshold)

sets the shake threshold for the y axis, on [0-65535]

acc_set_shake_threshold_z(self, threshold)

sets the shake threshold for the z axis, on [0-65535]

acc_smooth(self, command)

turns on smoothing for the accelerometer data, which is implemented by a running-average style low pass filter. (takes True/False)

acc_stream(self, command)

turns on streaming of the raw accelerometer data. you should have a handler installed before turning this on, or the frames will be discarded. (takes True/False)

acc_stream_var(self, command)

turns on streaming of the raw variance data. you should have a handler installed before turning this on, or the frames will be discarded. (takes True/False)

app_count(self)

returns the number of apps in the flash

app_delete_all(self)

deletes all apps from the flash

app_delete_atslot(self, slot)

deletes the app at the given slot in the flash

app_delete_withname(self, name)

deletes the app with the given name from the flash

app_exists_atslot(self, slot)

returns 1 if an app exists at the given slot, 0 otherwise

app_exists_withname(self, name)

returns 1 if an app exists with the given name, 0 otherwise

app_get_current_name(self)

retrns the name of the currently selected app

app_get_current_slot(self)

returns the slot of the currently selected app

app_get_name_atslot(self, slot)

returns the name of the app at the given slot

app_get_slot_withname(self, name)

returns the slot where the app with the given name resides

app_new_atslot_withname(self, slot, name)

creates a new app in the flash, at the given slot, and with the given name

app_new_withname(self, name)

creates a new app in the flash, at the next available slot, with the given name

app_reset_withname(self, name)

restarts the application with the given name

app_restart_atslot(self, slot)

restarts the application at the given slot

app_restart_current(self)

restarts the current application, re-reading any initialization information from the flash

app_set_current_atslot(self, slot)

sets the current app to be the one at the given slot

app_set_current_withname(self, name)

sets the current app to be the one with the given name

app_set_name_atslot(self, slot, name)

sets the name of the app at the given slot to the given name

close(self)

attempts to shut down the Bluetooth connection to the Siftable

color_get_depth(self)

returns the current color depth being used for graphics

color_set_both(self, r, g, b)

sets both outline and fill colors to the same value. r, g, and b are on [0-255]

color_set_depth(self, depth)

sets color depth for graphics. allowed values are 8 and 16

color_set_fill(self, r, g, b)

sets the fill color for shape drawing. r, g, and b are on [0-255]

color_set_outline(self, r, g, b)

sets the outline color for shape drawing. r, g, and b are on [0-255]

draw_allborder(self)

draws a border all the way around the siftable's screen, using the current colors

draw_border(self, side)

draws a rectangle that spans the given side

draw_circle(self, col, row, radius)

draws a circle at the given row and col, with the given radius

draw_line(self, col1, row1, col2, row2)

draws a line. note: col2 must be greater than col1, and row2 must be greater than row1

draw_neighbormarker(self, side)

draws a simple marker in the center of the given side. useful for debugging, when you want to show that the siftable is aware of a given neighbor

draw_pixel(self, col, row)

draws a single pixel. note: currently uses the draw_rect routine internally - not efficient

draw_rect(self, col1, row1, col2, row2)

draws a rectangle. note: col2 must be greater than col1, and row2 must be greater than row1

draw_testpattern(self)

draws a simple test pattern to the screen

echo(self, command)

toggles character echo behavior for terminal access. (takes True/False)

flash_getstatusbyte(self)

returns the current status byte of the off-board flash memory

flash_setbinary(self)

sets the off-board flash memory to use a power-of-two page size. all siftables should be configured with this option already, so you should not need to use this command

handler_100hz(self, command)

turns the internal (C firmware API) handler on the 100hz interval on or off. (takes True/False)

handler_10hz(self, command)

turns the internal (C firmware API) handler on the 10hz interval on or off. (takes True/False)

handler_1hz(self, command)

turns the internal (C firmware API) handler on the 1hz interval on or off. (takes True/False)

handler_25hz(self, command)

turns the internal (C firmware API) handler on the 25hz interval on or off. (takes True/False)

handler_50hz(self, command)

turns the internal (C firmware API) handler on the50hz interval on or off. (takes True/False)

handler_5hz(self, command)

turns the internal (C firmware API) handler on the 5hz interval on or off. (takes True/False)

handler_acc_data(self, command)

turns the internal (C firmware API) handler for accelerometer data on or off. (takes True/False)

handler_acc_shake_events(self, command)

turns the internal (C firmware API) handler for shake events on or off. (takes True/False)

handler_acc_tilt_events(self, command)

turns the internal (C firmware API) handler for tilt events on or off. (takes True/False)

handler_neighbor_events(self, command)

turns the internal (C firmware API) handler for neighbor events on or off. (takes True/False)

id_get(self)

returns the numeric ID of the siftable

id_set(self, new_id)

sets the ID of a siftable to a new value. note: ids can be in the range of [0-255]. all existing siftables have an ID already, so you should not need to do this. note also that this will NOT change the Bluetooth name of the sift to reflect the new ID. you should not need to use this function!

image_animate(self, start_idx, end_idx, delay_ms=0)

animates through images stored in the flash memory, from start_idx to end_idx, with a short delay between each. note: delay_ms is currently ignored

image_display(self, idx)

instructs the siftable to display the image at the given index. note that image indexing depends on the current color depth. we recommend that you stick to a single color depth for images stored on a given siftable

image_set_current(self, idx)

sets the 'current image' to the given index.
note: this is only used with neighbor-marking behavior

image_stream(self, im)

streams the passed-in image to the siftable's screen

image_upload(self, im, idx, force=False)

uploads the passed-in image to the given index. note that image indexing depends on the current color depth. we recommend that you stick to a single color depth for images stored on a given siftable. to upload images to slots 0, 1, or 2 you have to pass force=True, since these are system-reserved areas of the flash

install_listener_custom_events(self, listener, event_name_string)

install listener function for custom events

install_listener_neighbor_events(self, listener)

install a listener function for neighbor events

install_listener_raw_acc_data(self, listener)

install a listener function for raw accelerometer data frames

install_listener_raw_var_data(self, listener)

install a listener function for accelerometer variance data frames

install_listener_shake_events(self, listener)

install a listener function for shake events

install_listener_tilt_events(self, listener)

install a listener function for tilt events

led_green(self, command)

turn the green LED on or off. (takes True/False).
on the current siftables, the LEDs are not visible, so this command is not very useful anymore.

led_green_toggle(self)

Toggles the green LED. on the current siftables, the LEDs are not visible, so this command is not very useful anymore

led_red(self, command)

turn the red LED on or off. (takes True/False).
on the current siftables, the LEDs are not visible, so this command is not very useful anymore.

led_red_toggle(self)

Toggles the red LED. on the current siftables, the LEDs are not visible, so this command is not very useful anymore

neighbor_broadcast(self, command)

turns broadcasting of this siftable's ID and side on/off (takes: True/False)

neighbor_events(self, command)

turns event-reporting for neighborhood changes on or off (takes: True/False)

neighbor_markers(self, command)

turns neighbor markers on or off (takes: True/False)
note: neighbor-marking behavior utilizes the current image as a background

neighbor_snapshot(self)

returns an array representing the current neighborhood, as tracked by the siftable.
the format of this array is: [neighbor_TOP_id, neighbor_TOP_side, ...]
the order is TOP, LEFT, RIGHT, BOTTOM
a sample return value is: [0,0,25,1,0,0,42,0]
meaning that: siftable 25 is to the left, and its left side is facing, and siftable 42 is to the bottom, and its top side is facing

ping(self)

just lets you know that the sift is ok. returns: 'ping'

power_off(self)

immediately powers off the siftable. note: use of this function typically makes it difficult to detach cleanly from the Bluetooth radio.
see power_shutdown_withdelay for a better way to do this

power_shutdown_cancel(self)

cancels a pending power_shutdown_withdelay command

power_shutdown_withdelay(self, delay)

shuts down after the given number of seconds. use this to allow your code to cleanly disconnect from the siftable before it shuts off

power_status(self)

returns the status of the power_good line on the main micro. if you get a reply, the value will be 1

remove_listener_custom_events(self, event_name_string)

remove_listener_neighbor_events(self)

remove the listener function for tilt events

remove_listener_raw_acc_data(self)

remove the current listener function for raw accelerometer data frames

remove_listener_raw_var_data(self)

remove the listener function for accelerometer variance data frames

remove_listener_shake_events(self)

remove the listener function for shake events

remove_listener_tilt_events(self)

remove the listener function for tilt events

return_acks(self, acks_on)

determines whether the siftable library will returns acknowledgements from the siftable, such as: 'ok acc calibrate'
communication with the siftable will be much faster if acknowledgement returning is off. (takes True/False)

screen_awake(self)

puts the screen into awake mode (also see screen_sleep)

screen_bright_max(self)

sets the screen brightness to its maximum value

screen_bright_min(self)

sets the screen brightness to its minimum value

screen_bright_val(self, val)

sets the screen brightness to a given value on [0-255]

screen_clear(self)

clears any graphics on the screen, returning it to all black pixels

screen_sleep(self)

puts the screen into power-saving sleep mode (also see screen_awake)

send_packet(self, data, return_result=True)

var_count(self)

returns the number of variable / value bindings on the current application page

var_delete(self, name)

removes a variable / value binding from the current application page. if there is no such binding, returns an error

var_get(self, name)

returns the value associated with a given variable name, if that binding exists on the current application page. if there is no variable with that name, returns None

var_set(self, name, val)

writes a variable / value binding to the flash memory, on the current application page

Appendix B

Siftable Hardware Schematics and Circuit Board Layout Designs

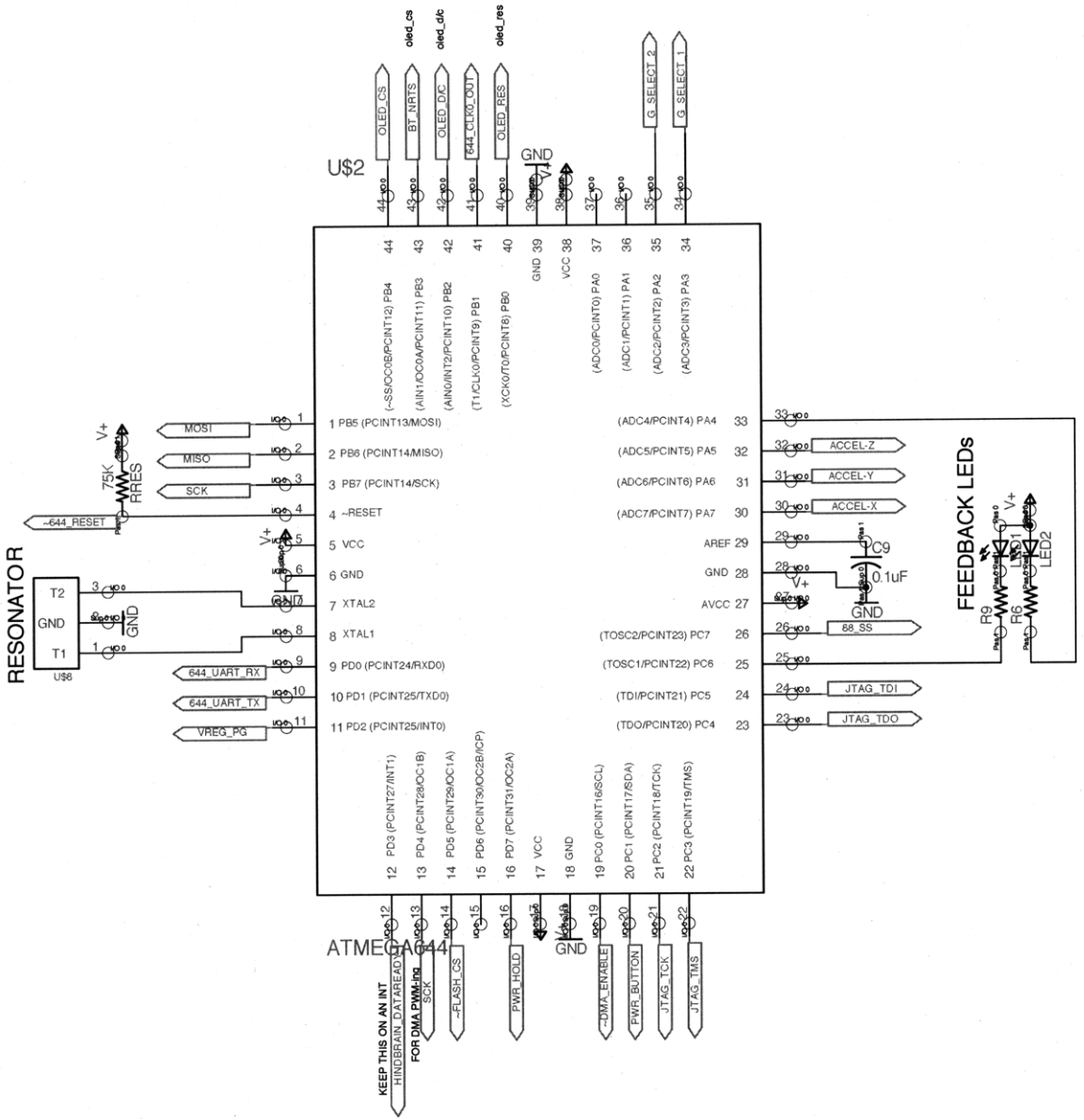


Figure B-1: Schematic diagram for the main microcontroller

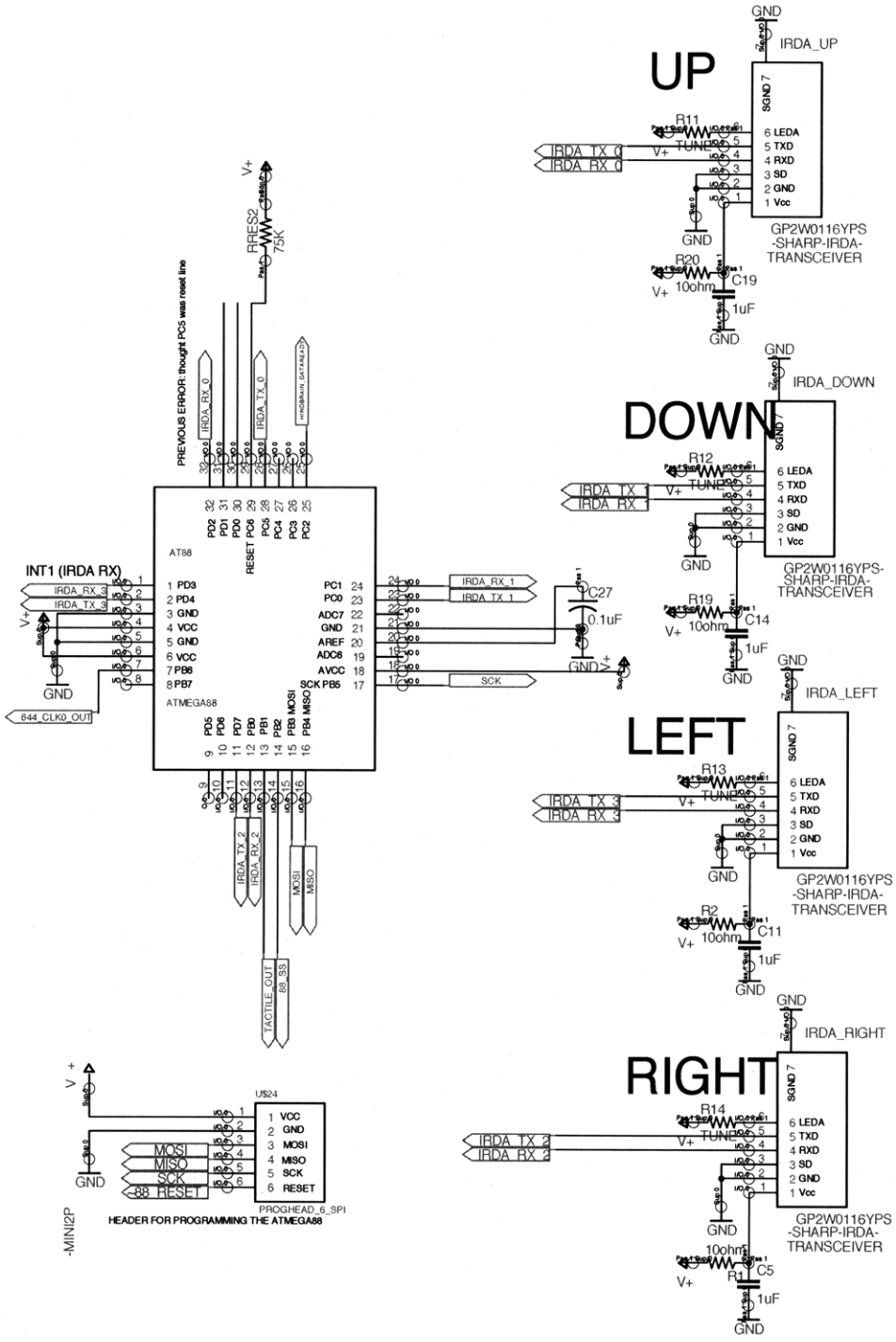


Figure B-2: Schematic diagram for the secondary microcontroller and infrared communication circuitry

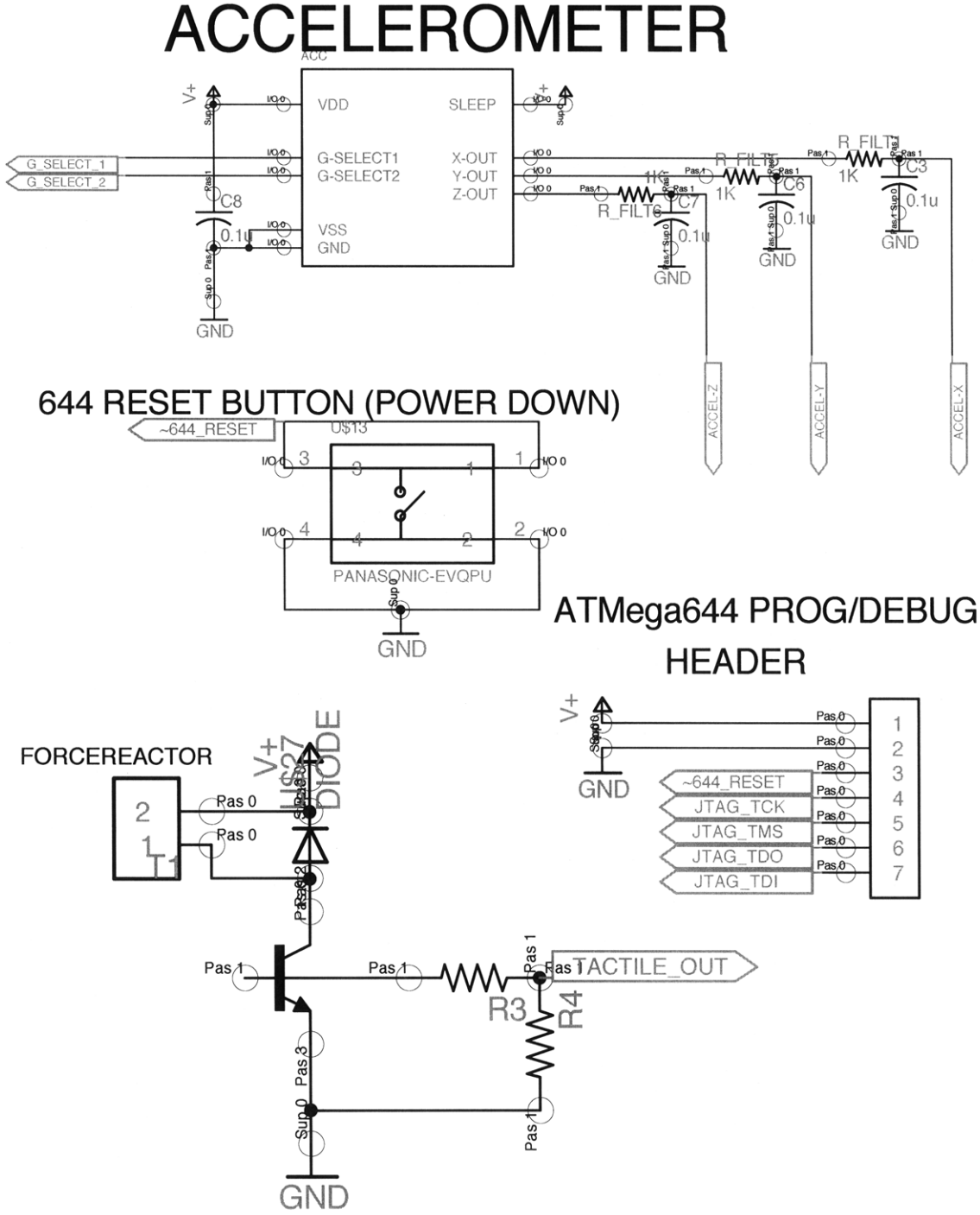


Figure B-3: Schematic diagram for the main microcontroller's programming header, the power toggle button, accelerometer and signal conditioning circuitry, and tactile actuation driver circuit (not used)

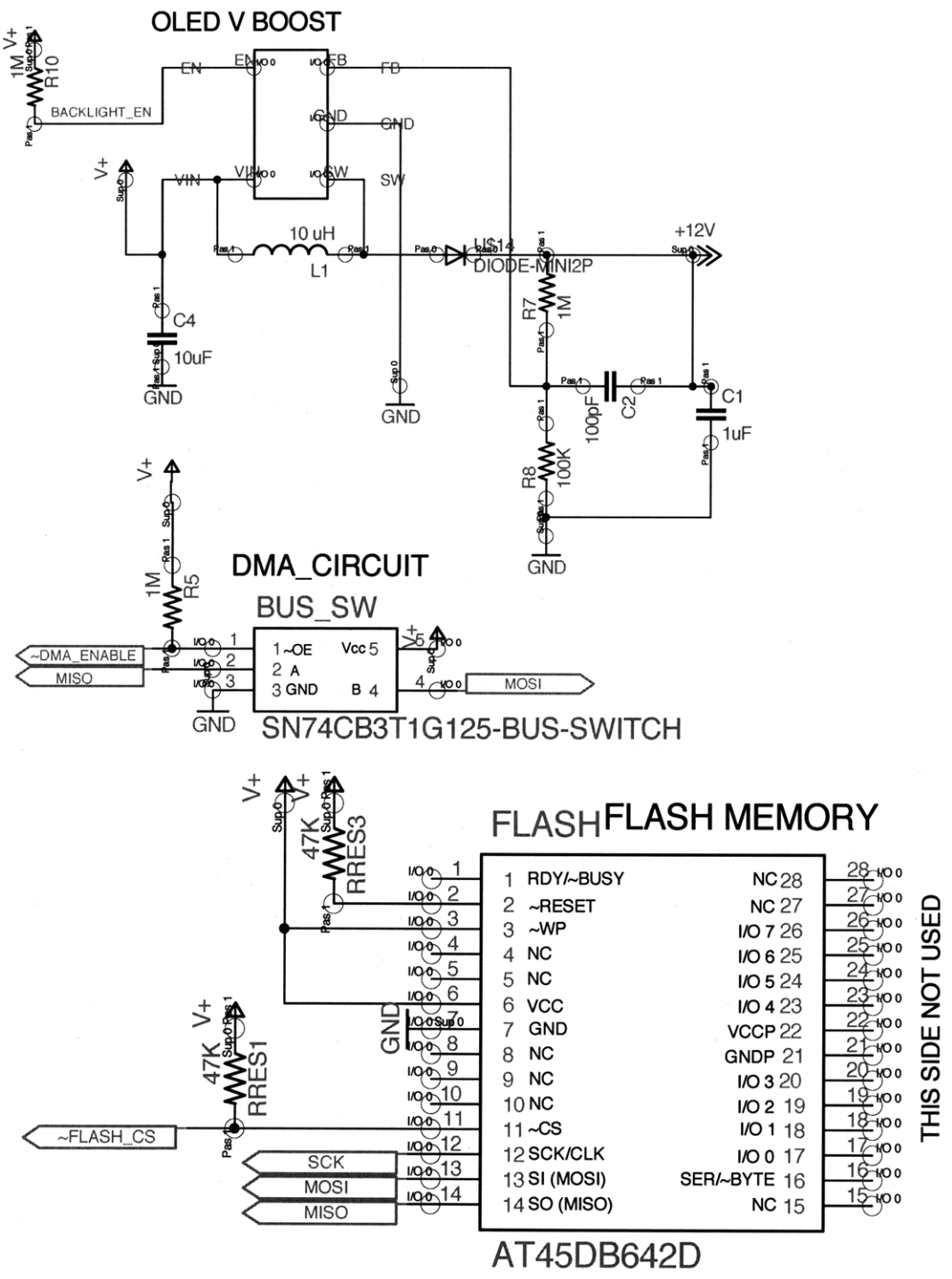


Figure B-4: Schematic diagram for the voltage boost circuit for the OLED display, the Direct Memory Access circuit that allows images to be fetched at high speeds from the flash to the display, and the flash memory IC

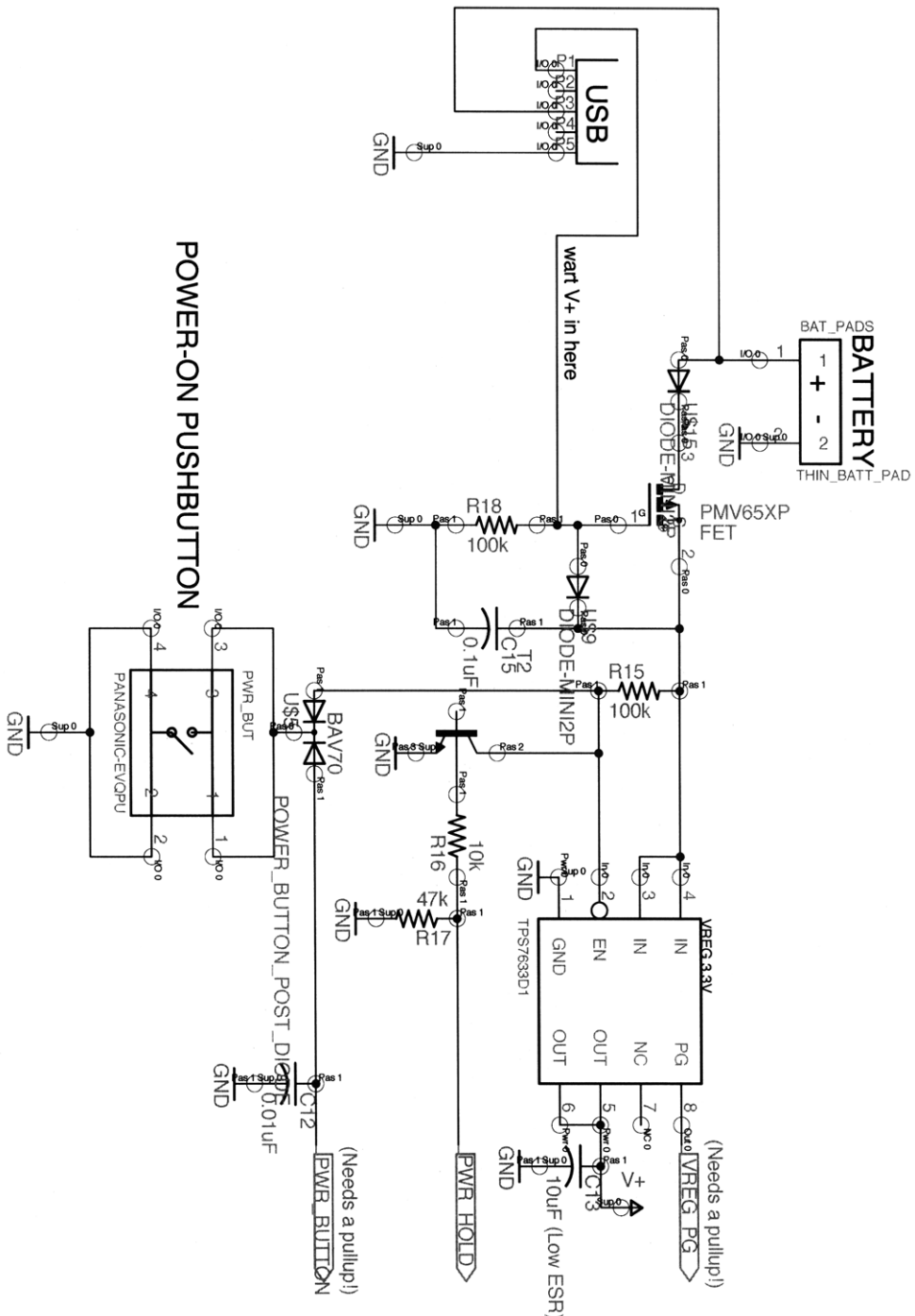


Figure B-5: Schematic diagram for the power handling circuitry

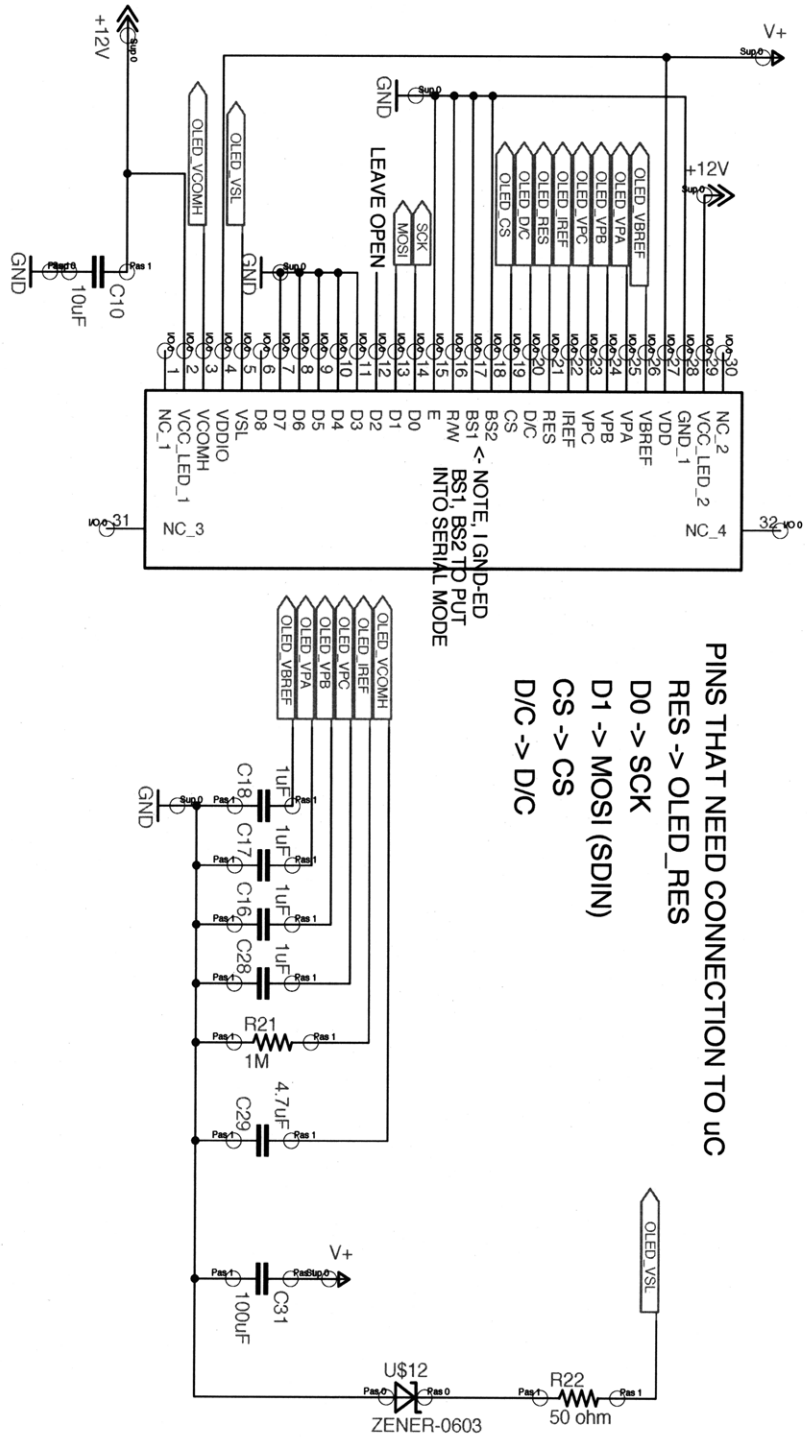


Figure B-6: Schematic diagram for the OLED display connector

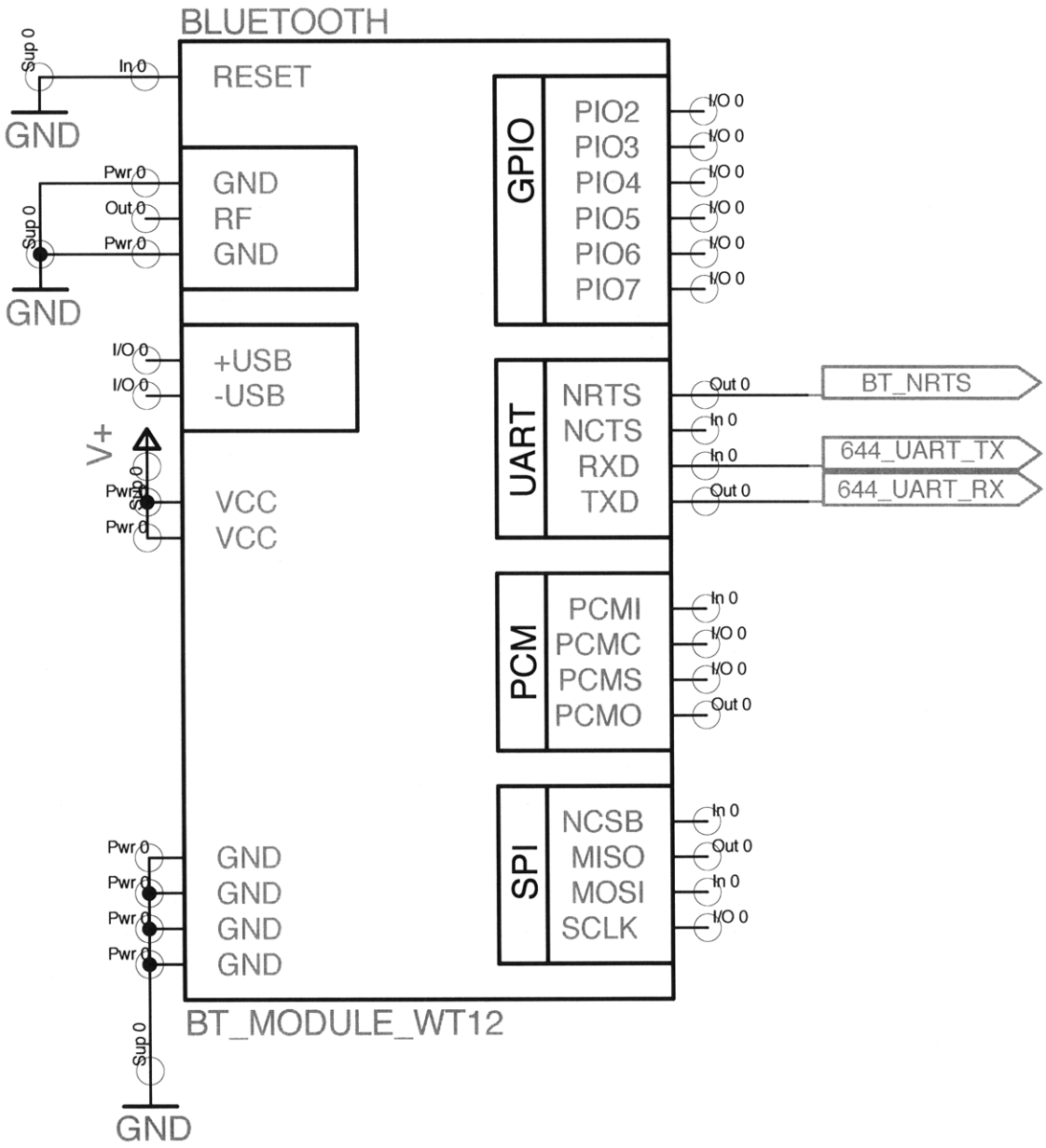


Figure B-7: Schematic diagram for the Bluetooth radio

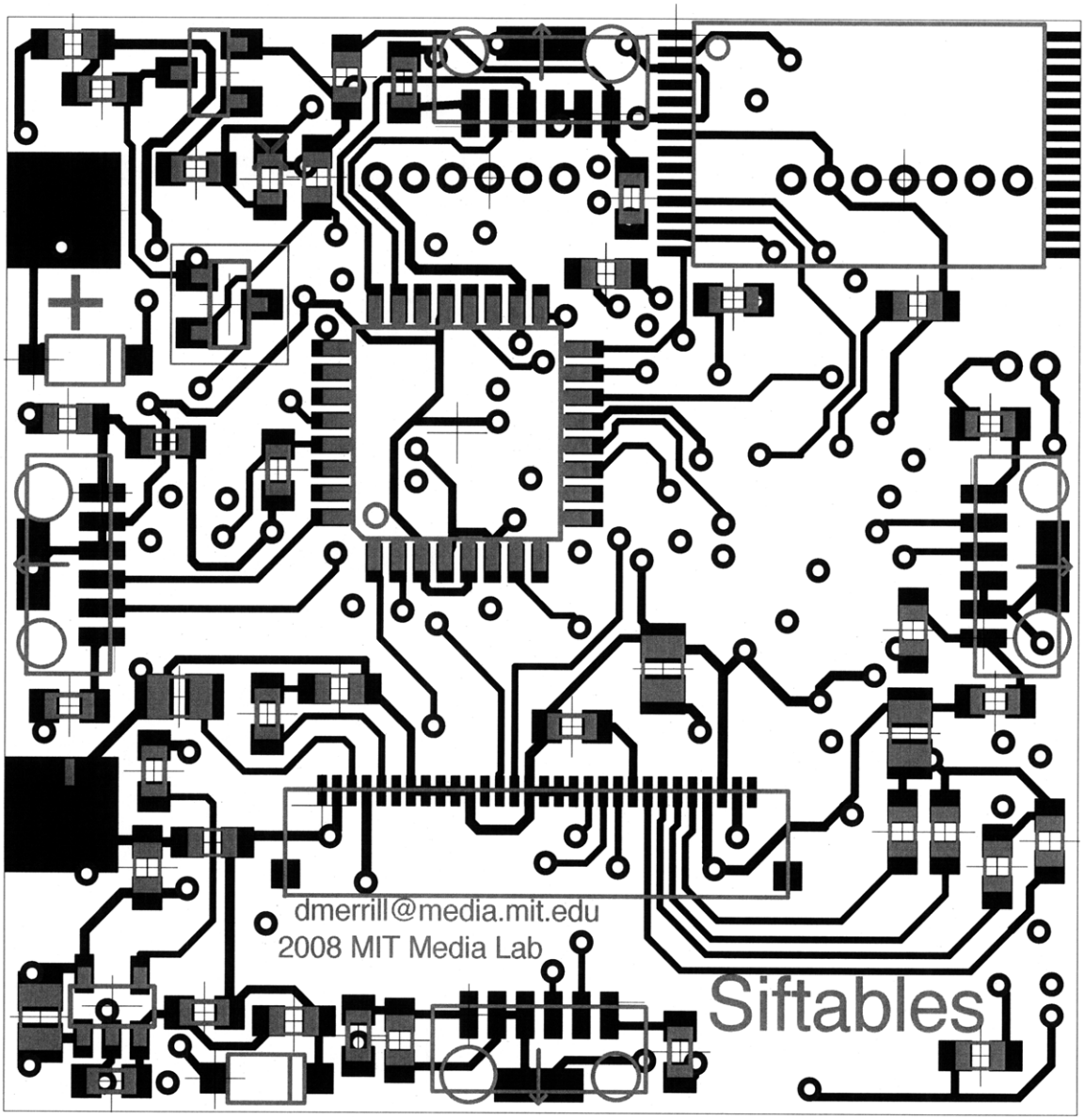


Figure B-8: Printed circuit board layout (top layer)

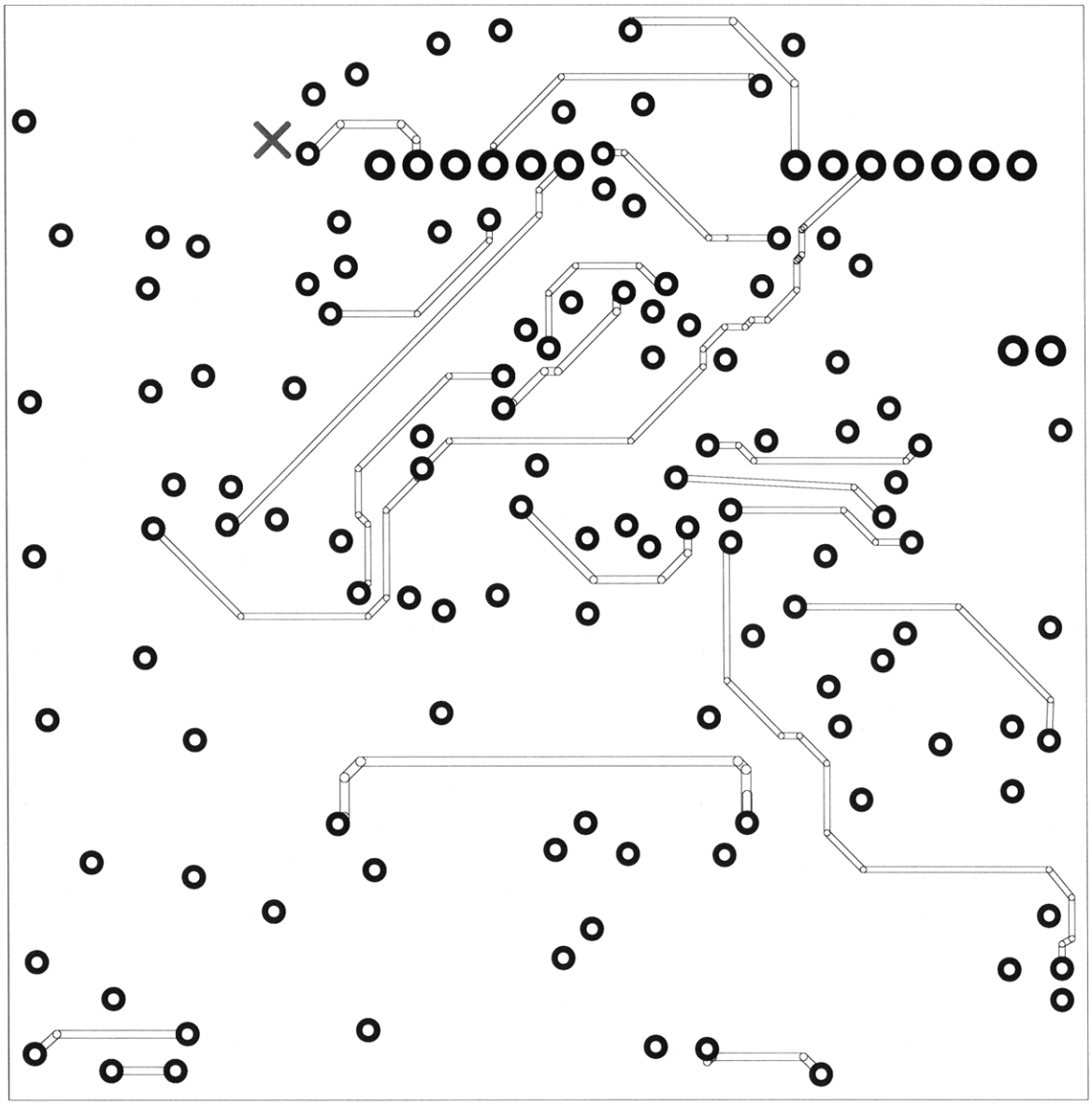


Figure B-9: Printed circuit board layout (internal layer 1)

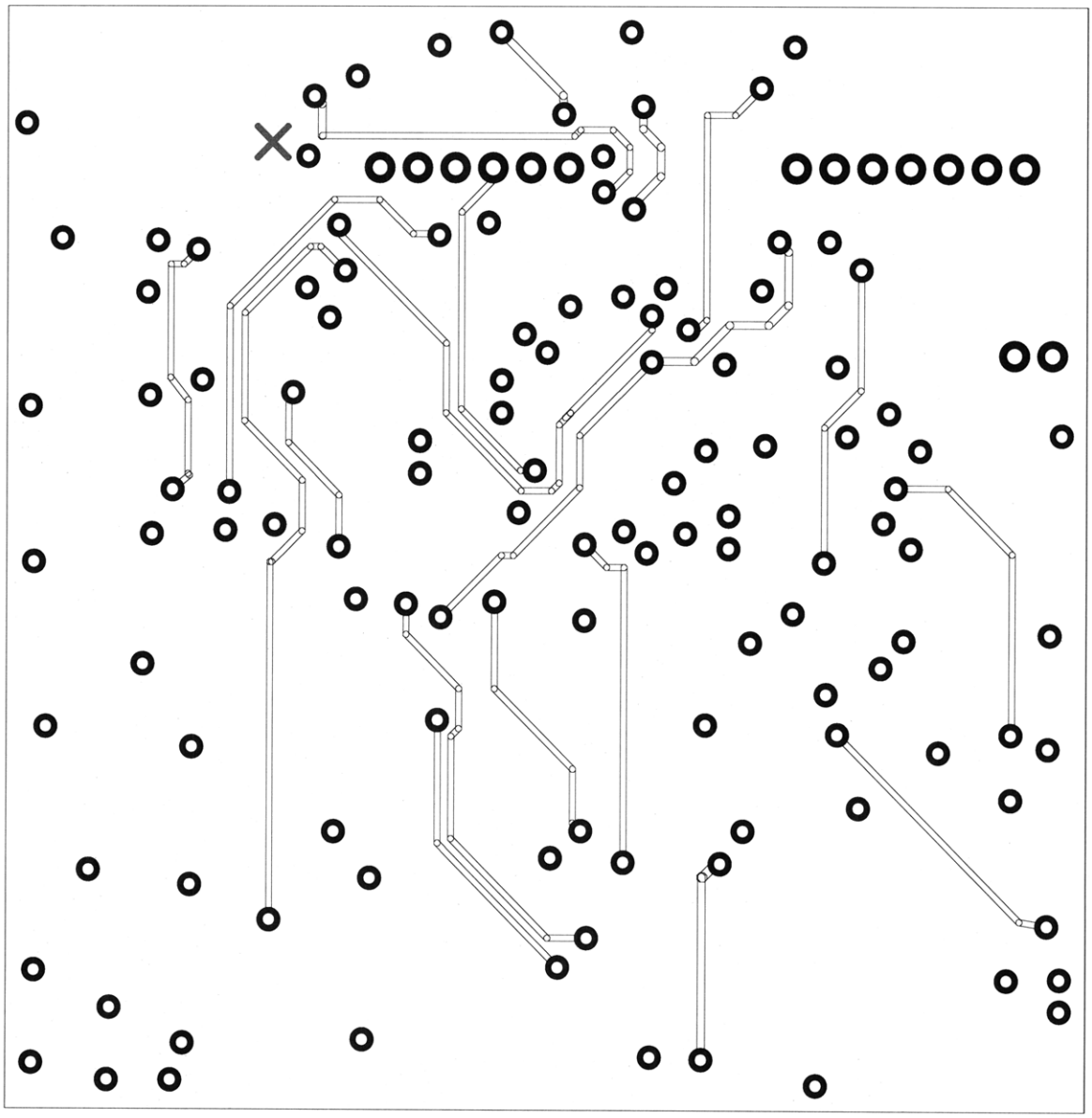


Figure B-10: Printed circuit board layout (internal layer 2)

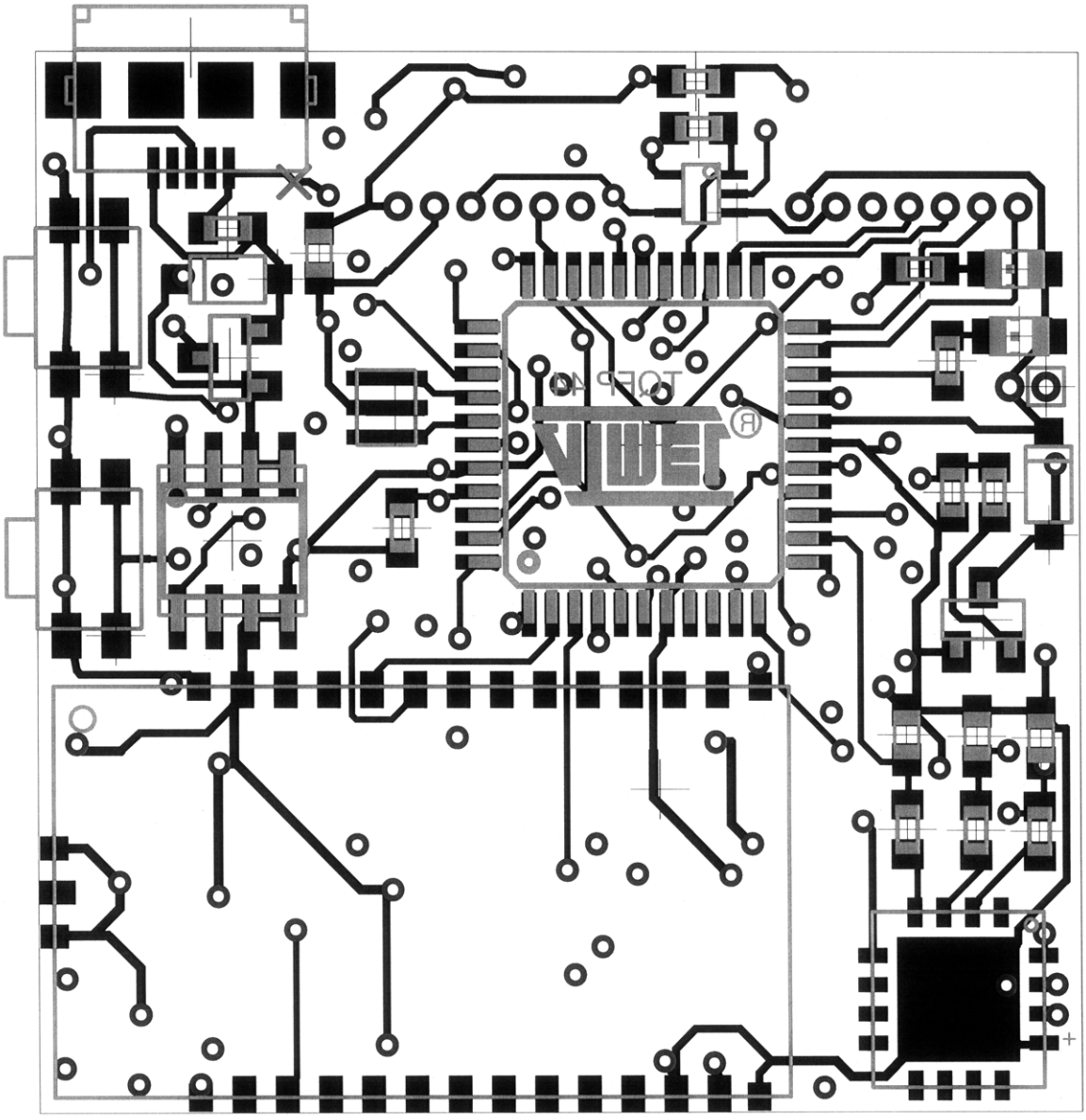


Figure B-11: Printed circuit board layout (bottom layer)

Appendix C

Siftable Flash Memory Organization

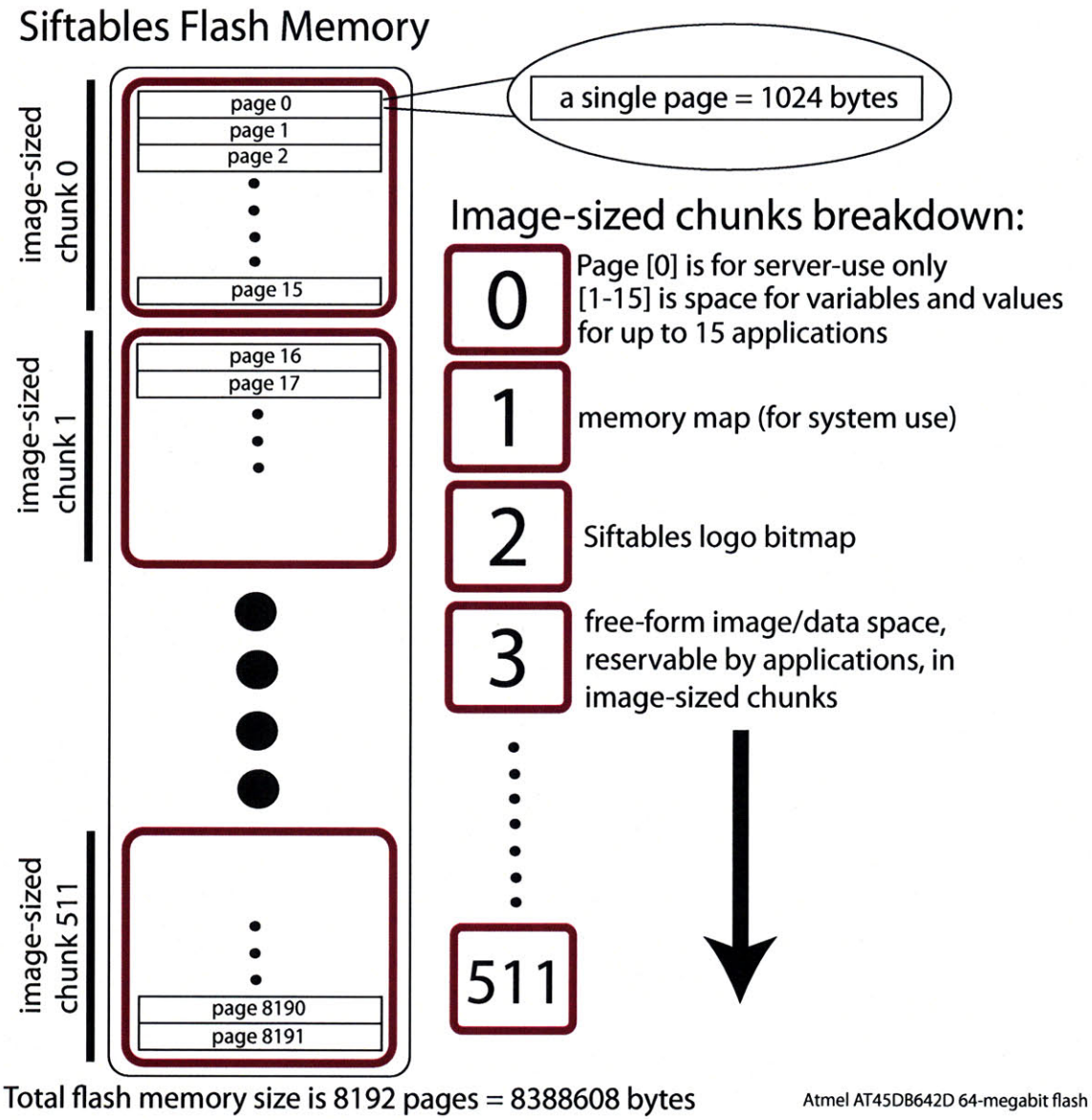


Figure C-1: Organization of the 64-mbit flash memory

Bibliography

[1] Ableton AG. Ableton live. `http://www.ableton.com`.

[2] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[3] Jussi Ängeslevä, Sile O'Modhrain, Ian Oakley, and Stephen Hughes. Body mnemonics: Portable device interaction design concept. In *UIST '03*, Vancouver, Canada, 2003.

[4] Infrared Data Association. `http://www.irda.org/`.

[5] Atmel. Atmel website. `http://atmel.com/`.

[6] Rafael Ballagas, Faraz Memon, Rene Reiners, and Jan Borchers. istuff mobile: rapidly prototyping new mobile phone interfaces for ubiquitous computing. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1107–1116, New York, NY, USA, 2007. ACM.

[7] Bandai. Tamagotchi. `http://www.tamagotchi.com/`.

[8] Dave Baum. *Dave Baum's Definitive Guide to LEGO Mindstorms (Technology In Action)*. APress LP, 2002.

[9] Ari Benbasat. An inertial measurement unit for user interfaces. Master's thesis, Massachusetts Institute of Technology, School of Architecture and Planning, Program in Media Arts and Sciences, 2000.

[10] Jeffrey Traer Bernstein. Tangible sequencer, 2007. `http://www.tangiblesequencer.com/`.

[11] Jonathan Bernstein. An overview of mems inertial sensing technology. *Sensors Online*, 2003. `http://www.sensorsmag.com/articles/0203/14/`.

[12] Rick Borovoy, Brian Silverman, Tim Gorton, Matt Notowidigdo, Brian Knep, Mitchel Resnick, and Jeff Klann. Folk computing: revisiting oral tradition as a scaffold for co-present communities. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 466–473, 2001.

[13] Joel Brandt, Noah Weiss, and Scott R. Klemmer. Designing for situations with limited attention. Technical report, Stanford University, 2007.

[14] Jeff Burke, Deborah Estrin, Mark Hansen, Andrew Parker, Nithya Ramanathan, Sasank Reddy, and Mani Srivastava. Participatory sensing. *ACM Sensys World Sensor Web Workshop*, 2006.

[15] William Butera. *Programming a paintable computer.* PhD thesis, Massachusetts Institute of Technology, School of Architecture and Planning, Program in Media Arts and Sciences, 2002.

[16] Bill Buxton. *Sketching user experiences: getting the design right and the right design.* Morgan Kaufmann, 2007.

[17] Bill Buxton. Multi-touch systems that i have known and loved, August 2008. `http://www.billbuxton.com/multitouchOverview.html`.

[18] John Horton Conway. The game of life. *Scientific American*, 223:120–123, 1970.

[19] Enrico Costanza, S. B. Shelley, and J. Robinson. d-touch: A consumer grade tangible interface module and musical applications. In *Conference on Human-Computer Interaction (HCI03)*, Bath, 2003.

[20] John W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches.* Sage Publications, Inc., 2008.

[21] Delphi. Delphi bi-directional key fob. `http://delphi.com/manufacturers/auto/controls/security/bi-directional_key_fob/`.

[22] Analog Devices. Analog devices. `http://www.analog.com/`.

[23] EInk. Electronic paper displays. `http://www.eink.com/`.

[24] Engadget. Hitachi h001 with 3d display leads up kddi au's spring 2009 lineup. `http://www.engadget.com/2009/01/29/hitachi-h001-with-3d-display-leads-up-kddi-aus-spring-2009-line/`.

[25] Douglas C. Engelbart and William K. English. A research center for augmenting human intellect. *AFIPS Conference Proceedings of the 1968 Fall Joint Computer Conference*, 33:395–410, 1968.

[26] Irfan A. Essa. Ubiquitous sensing for smart and aware environments. *Personal Communications, IEEE*, 7(5):47–49, 2000.

[27] Assaf Feldman, Emmanuel Munguia Tapia, Sajid Sadi, Pattie Maes, and Chris Schmandt. Reachmedia: On-the-move interaction with everyday objects. *Proceedings of the Ninth IEEE International Symposium on Wearable Computers*, 2005.

[28] Kenneth P. Fishkin, Anuj Gujar, Beverly L. Harrison, Thomas P. Moran, and Roy Want. Embodied user interfaces for really direct manipulation. *Communications of the ACM*, 43(9):74–80, 2000.

[29] George W. Fitzmaurice. *Graspable User Interfaces.* PhD thesis, University of Toronto, 1997.

[30] George W. Fitzmaurice, Hiroshi Ishii, and William Buxton. Bricks: Laying the foundations for graspable user interfaces. *Proceedings of CHI'05*, pages 422–449, 1995.

[31] Free Software Foundation. Gcc, the gnu compiler collection. `http://gcc.gnu.org/`.

[32] Ben Fry and Casey Reas. Processing. `http://processing.org/`.

[33] Radica Games. Cube world. `http://www.radicagames.com/cubeworld/`.

[34] Albert Vincent Glinsky. *The Theremin in the Emergence of Electronic Music.* PhD thesis, New York University, 1992.

[35] Matthew G. Gorbet, Maggie Orth, and Hiroshi Ishii. Triangles: tangible interface for manipulation and exploration of digital information topography. In *CHI '98: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 49–56, New York, NY, USA, 1998. ACM Press/Addison-Wesley Publishing Co.

[36] Wayne D. Gray and Deborah A. Boehm-Davis. Milliseconds matter: an introduction to microstrategies and to their use in describing and predicting interactive behavior. *Journal of Experimental Psychology: Applied*, 6(4):322–35, 2000.

[37] Saul Greenberg and Bill Buxton. Usability evaluation considered harmful (some of the time). In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 111–120, New York, NY, USA, 2008. ACM.

[38] Adam Greenfield. *Everyware: The dawning age of ubiquitous computing.* Peachpit Press Berkeley, CA, USA, 2006.

[39] Jefferson Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In *Proceedings of UIST'05*, pages 115–118, New York, NY, USA, 2005. ACM Press.

[40] Beverly L. Harrison, Kenneth P. Fishkin, Anuj Gujar, Carlos Mochon, and Roy Want. Squeeze me, hold me, tilt me! an exploration of manipulative user interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1998.

[41] Björn Hartmann, Scott R. Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. Reflective physical prototyping through integrated design, test, and analysis. *Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 299–308, 2006.

[42] Björn Hartmann, Leslie Wu, Kevin Collins, and Scott R. Klemmer. Programming by a sample: Rapidly prototyping web applications with d. mix. *In Proceeding of the 20th Symp. on User Interface Software and Technology (UIST07). Newport, RI, USA*, 2007.

[43] Björn Hartmann, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R. Klemmer. Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. *Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 91–100, 2008.

[44] Ken Hinckley. *Haptic issues for virtual manipulation.* PhD thesis, University of Virginia, 1997.

[45] Ken Hinckley. Synchronous gestures for multiple persons and computers. In *Proceedings of UIST'03*, pages 149–158, New York, NY, USA, 2003. ACM Press.

[46] Ken Hinckley, Randy Pausch, John C. Goble, and Neal F. Kassell. Passive real-world interface props for neurosurgical visualization. *Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence*, pages 452–458, 1994.

[47] Lars Erik Holmquist, Friedemann Mattern, Bernt Schiele, Petteri Alahuhta, Michael Beigl, and Hans Gellersen. Smart-its friends: A technique for users to easily establish connections between smart artefacts. *Proceedings of UbiComp'01*, pages 116–122, 2001.

[48] Michael S. Horn and Robert J. K. Jacob. Tangible programming in the classroom with tern. In *CHI '07: CHI '07 extended abstracts on Human factors in computing systems*, pages 1965–1970, New York, NY, USA, 2007. ACM.

[49] Edwin Hutchins. *Cognition in the wild.* MIT Press, 1995.

[50] Adobe Inc. Adobe flash. `http://www.adobe.com/products/flash/`.

[51] Nintendo Inc. Nintendo wii website. `http://www.nintendo.com/wii`.

[52] Adobe Systems Incorporated. Adobe photoshop. `http://www.adobe.com/`.

[53] Apple Incorporated. iphone. `http://www.apple.com/iphone/`.

[54] Hiroshi Ishii and Brygg Ullmer. Tangible bits: Towards seamless interfaces between people, bits, and atoms. *Proceedings of CHI'97*, pages 234–241, 1997.

[55] S. Jordà. Sonigraphical instruments: from fmol to the reactable. *Proceedings of the 2003 conference on New interfaces for musical expression*, pages 70–76, 2003.

[56] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: mobile networking for "smart dust". *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 271–278, 1999.

[57] David Kirsh. The intelligent use of space. *Artificial Intelligence*, 73(1-2):31–68, 1995.

[58] Scott R. Klemmer, Björn Hartmann, and Leila Takayama. How bodies matter: five themes for interaction design. *Proceedings of the 6th conference on Designing Interactive systems*, pages 140–149, 2006.

[59] Scott R. Klemmer, Jack Li, James Lin, and James A. Landay. Papier-mache: toolkit support for tangible input. *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 399–406, 2004.

[60] Scott R. Klemmer, Mark W. Newman, Ryan Farrell, Mark Bilezikjian, and James A. Landay. The designers' outpost: a tangible interface for collaborative web site. *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 1–10, 2001.

[61] Kwindla Hultman Kramer. Moveable objects, mobile code. Master's thesis, Massachusetts Institute of Technology, School of Architecture and Planning, Program in Media Arts and Sciences, 1998.

[62] Bug Labs. Bug. `http://www.buglabs.net`.

[63] Jonathan Lester, Blake Hannaford, and Gaetano Borriello. "are you with me?" - using accelerometers to determine if two devices are carried by the same person. *Pervasive Computing*, pages 33–50, 2004.

[64] Henry Lieberman. The tyranny of evaluation. *ACM CHI Fringe*, 2003. http://web.media.mit.edu/ lieber/Misc/Tyranny-Evaluation.html.

[65] Joshua Lifton, Michael Broxton, and Joseph A. Paradiso. Experiences and directions in pushpin computing. *Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005.

[66] Joshua Lifton, Deva Seetharam, Michael Broxton, and Joseph A. Paradiso. Pushpin computing system overview: A platform for distributed, embedded, ubiquitous sensor networks. *Pervasive Computing*, pages 139–151, 2002.

[67] Benny P.L. Lo, Surapa Thiemjarus, Rachel King, and Guang-Zhong Yang. Body sensor network–a wireless sensor platform for pervasive healthcare monitoring. *The 3rd International Conference on Pervasive Computing*, 2005.

[68] ALPS Electric Company LTD. Alps glidepoint. `http://www.alps.com/`.

[69] Wendy E. Mackay. Is paper safer? the role of paper flight strips in air traffic control. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 6(4):311–340, 1999.

[70] Paul P. Maglio and David Kirsch. Epistemic action increases with skill. *Proceedings of the Eighteenth Annual Conference of the Cognitive Science Society*, pages 391–396, 1996.

[71] Paul P. Maglio, Teenie Matlock, Dorth Raphaely, Brian Chernicky, and David Kirsch. Interactive skill in scrabble. *Proceedings of the Twenty-first Annual Conference of the Cognitive Science Society*, pages 326–330, 1999.

[72] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, 2002.

[73] Raluca Marin-Perianu, Mihai Marin-Perianu, Paul Havinga, and Hans Scholten. Movement-based group awareness with wireless sensor networks. *Pervasive Computing*, 4480:298, 2007.

[74] MatthewWright and Adrian Freed. Open sound control: A new protocol for communicating with sound synthesizers, 1997.

[75] Scott McMillan. Upper body tracking using the polhemus fastrak. Technical Report NPSCS-96-002, January 1996.

[76] Nimish Mehta. Flexible machine interface. Master's thesis, University of Toronto, 1982.

[77] David A. Mellis, Massimo Banzi, David Cuartielles, and Tom Igoe. Arduino: An open electronic prototyping platform. *CHI: ACM Conference on Human Factors in Computing Systems*, 2007.

[78] David Merrill. Flexigesture: A sensor-rich real-time adaptive gesture and affordance learning platform for electronic music control. Master's thesis, Massachusetts Institute of Technology, School of Architecture and Planning, Program in Media Arts and Sciences, 2004.

[79] David Merrill, Jeevan Kalanithi, and Pattie Maes. Siftables: towards sensor network user interfaces. *Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 75–78, 2007.

[80] David Merrill and Joseph A. Paradiso. Personalization, expressivity, and learnability of an implicit mapping strategy for physical interfaces. *Extended Abstracts of the Conference on Human Factors in Computing Systems (CHI05)*, 2005.

[81] MITS. Altair 8800. `http://en.wikipedia.org/wiki/Altair_8800`.

[82] Stacy J. Morris and Joseph A. Paradiso. Shoe-integrated sensor system for wireless gait analysis and real-time feedback. *Proceedings of the 2nd Joint IEEE EMBS (Engineering in Medicine and Biology Society) and BMES (the Biomedical Engineering Society) Conference*, pages 2468–2469, 2002.

[83] Roderick Murray-Smith, John Williamson, Stephen Hughes, and Torben Quaade. Stane: synthesized surfaces for tactile input. *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1299–1302, 2008.

[84] Henry Newton-Dunn, Hiroaki Nakano, and James Gibson. Block jam. In *SIGGRAPH '02: ACM SIGGRAPH 2002 conference abstracts and applications*, pages 67–67, New York, NY, USA, 2002. ACM.

[85] Nintendo. Dsi. `http://www.nintendodsi.com/`.

[86] D.A. Norman. *Things That Make Us Smart: Defending Human Attributes In The Age Of The Machine.* Basic Books, 1994.

[87] Ian Oakley and Sile O'Modhrain. Tilt to scroll: evaluating a motion based vibrotactile mobile interfaces. *Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005. World Haptics 2005. First Joint*, pages 40–49, 2005.

[88] Maura Sile O'Modhrain. *Incorporating Haptic Feedback into Computer-Based Musical Instruments.* PhD thesis, Stanford University, 2000.

[89] Leysia Palen, Marilyn Salzman, and Ed Youngs. Going wireless: behavior & practice of new mobile phone users. *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 201–210, 2000.

[90] Joseph A. Paradiso. Electronic music: new ways to play. *Spectrum, IEEE*, 34(12):18–30, 1997.

[91] Amanda Parkes, Vincent LeClerc, and Hiroshi Ishii. Glume: exploring materiality in a soft augmented modular modeling system. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 1211–1216, New York, NY, USA, 2006. ACM.

[92] J. Patten and H. Ishii. Mechanical constraints as computational constraints in tabletop tangible interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 809–818, 2007.

[93] James Patten. *Mechanical constraints as common ground between people and computers.* PhD thesis, Massachusetts Institute of Technology, School of Architecture and Planning, Program in Media Arts and Sciences, 2004.

[94] James Patten, Hiroshi Ishii, Jim Hines, and Gian Pangaro. Sensetable: A wireless object tracking platform for tangible user interfaces. In *Proceedings of CHI'01*, pages 253–260, Seattle, WA, 2001. ACM Press.

[95] Percussa. Audiocubes. `http://www.percussa.com/`.

[96] Antti Pirhonen, Stephen Brewster, and Christopher Holguin. Gestural and audio metaphors as a means of control for mobile devices. *Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves*, 2002.

[97] Polhemus. Polhemus patriot. `http://www.polhemus.com/?page=Motion_Patriot`.

[98] Ivan Poupyrev, Shigeaki Maruyama, and Jun Rekimoto. Ambient touch: designing tactile interfaces for handheld devices. *Proceedings of the 15th annual ACM symposium on User interface software and technology*, 2002.

[99] Miller Puckette. Pure data: another integrated computer music environment. *Proceedings of the Second Intercollege Computer Music Concerts*, pages 37–41, 1996.

[100] Hayes Solos Raffle, Amanda J. Parkes, and Hiroshi Ishii. Topobo: a constructive assembly system with kinetic memory. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 647–654, New York, NY, USA, 2004. ACM.

[101] Jun Rekimoto. Gesturewrist and gesturepad: Unobtrusive wearable interaction devices. In *5th IEEE International Symposium on Wearable Computers*, 2001.

[102] Jun Rekimoto, Brygg Ullmer, and Haruo Oba. Datatiles: a modular platform for mixed physical and graphical interactions. In *Proceedings of CHI '01*, pages 269–276, New York, NY, USA, 2001. ACM Press.

[103] Edward Sazonov, Kerop Janoyan, and Ratan Jha. Wireless intelligent sensor network for autonomous structural health monitoring. *Smart Structures/NDE 2004*, 2004.

[104] R.R. Schaller. Moore's law: past, present and future. *Spectrum, IEEE*, 34(6):52–59, 1997.

[105] Carsten Schwesig, Ivan Poupyrev, and Eijiro Mori. Gummi: a bendable computer. *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 263–270, 2004.

[106] Freescale Semiconductor. Accelerometers. `http://www.freescale.com/`.

[107] Dr. Seuss and Theodore Geisel. *Hop on Pop.* Random House Childrens Books, 2003.

[108] Claude E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[109] Scott S. Snibbe, Karon E. MacLean, Rob Shaw, Jayne Roderick, William L. Verplank, and Mark Scheeff. Haptic techniques for media control. *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 199–208, 2001.

[110] Steve Stiles. "dial-up" icd interrogation keeps the doctor away, May 2004. `http://www.theheart.org/article/142905.do`.

[111] Ivan Edward Sutherland. *Sketchpad: A man-machine graphical communication system.* PhD thesis, Massachusetts Institute of Technology, 1963.

[112] Synaptics. Touchpad. `http://www.synaptics.com/solutions/products/touchpad`.

[113] DeLiang Wang and David Terman. Image segmentation based on oscillatory correlation. *Neural Computation*, 9(4):805–836, 1997.

[114] Roy Want, Andy Hopper, Veronica Falcao, and Jon Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1), 1992.

[115] Reynold Weidenaar, Brian Lehrer, and Barbara Blegen. *Magic Music from the Telharmonium.* Scarecrow Press, 1995.

[116] Marc Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, 1991.

[117] P. Wellner. Interacting with paper on the digitaldesk. *Communications of the ACM*, 36(7):87–96, 1993.

[118] Stephen Wolfram. Theory and applications of cellular automata. *Advanced Series on Complex Systems, Singapore: World Scientific Publication, 1986*, 1986.

[119] J. Zhang, G. Harbottle, C. Wang, and Z. Kong. Oldest playable musical instruments found at jiahu early neolithic site in china. *Nature*, 401(6751):366–368, 1999.

[120] Jiajie Zhang and Donald A. Norman. Representations in distributed cognitive tasks. *Cognitive Science*, 18(1):87–122, 1994.

[121] Jamie Zigelbaum, Michael S. Horn, Orit Shaer, and Robert J.K. Jacobs. The tangible video editor: collaborative video editing with active tokens. *Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 43–46, 2007.

[122] Oren Zuckerman, Tina Grotzer, and Kelly Leahy. Flow blocks as a conceptual bridge between understanding the structure and behavior of a complex causal system, 2006.

[123] Orit Zuckerman. Spotlight. http://web.media.mit.edu/~orit/.